ALIBABA CLOUD

阿里云

容器服务Kubernetes版
Kubernetes集群用户指南

文档版本：20211202

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.ACK集群概述 23
2.首次使用容器服务Kubernetes版 28
3.授权 31
3.1. 授权概述 31
3.2. 自定义RAM授权策略 33
3.3. 自定义Kubernetes授权策略 40
3.4. 配置RAM用户RBAC权限 42
3.5. 容器服务默认角色 47
3.6. 授权管理FAQ 72
4.集群 79
4.1. 迁移至Pro集群 79
4.1.1. 热迁移ACK标准托管集群至Pro托管集群 79
4.1.2. 热迁移ACK专有版集群至Pro托管集群 80
4.2. 操作系统 82
4.2.1. ACK等保加固使用说明 82
4.2.2. ACK CIS加固使用说明 83
4.2.3. 使用操作系统Alibaba Cloud Linux 2 89
4.2.4. ContainerOS概述 91
4.2.5. ContainerOS运维容器使用指导 92
4.3. 创建集群 94
4.3.1. 创建Kubernetes专有版集群 95
4.3.2. 创建Kubernetes托管版集群 104
4.3.3. 使用自定义镜像创建Kubernetes集群 115
4.3.4. 通过生成OpenAPI参数创建集群 119
4.4. 连接集群 120
4.4.1. 通过kubectl工具连接集群 120

4.4.2. 在CloudShell上通过kubectl管理Kubernetes集群 123
4.4.3. 集群访问控制 124
4.4.3.1. 配置集群API Server的访问控制策略 124
4.4.3.2. 控制集群API Server的公网访问能力 125
4.4.3.3. 吊销集群的KubeConfig凭证 126
4.4.4. 更多连接方式 127
4.4.4.1. 通过SSH连接ACK专有版集群的Master节点 127
4.4.4.2. 使用ServiceAccount Token访问Kubernetes集群 129
4.5. 管理集群 131
4.5.1. 查看集群信息 131
4.5.2. 集群证书 132
4.5.2.1. 集群证书更新说明 132
4.5.2.2. 更新Kubernetes集群即将过期的证书 135
4.5.2.3. 更新Kubernetes集群已过期的证书 141
4.5.3. 升级ACK集群K8s版本 141
4.5.4. 扩容ACK集群的节点 145
4.5.5. 升级Kubernetes集群的metrics-server组件 147
4.5.6. 删除集群 151
4.6. 通过集群检查功能定位集群问题 152
4.7. 通过集群故障诊断功能定位集群问题 154
4.8. 集群管理FAQ 156
5.ACK Pro集群 158
5.1. ACK Pro版集群概述 158
5.2. 创建ACK Pro版集群 159
5.3. CPU拓扑感知调度 168
5.4. GPU拓扑感知调度 168
5.5. 共享GPU调度专业版 169
5.6. 任务调度 169

5.7. 使用负载感知调度 169
5.8. 使用阿里云KMS进行Secret的落盘加密 174
5.9. 自定义ACK Pro集群的管控面参数 176
6.节点与节点池 178
6.1. 节点 178
6.1.1. 添加已有节点 178
6.1.2. 最小化集群访问规则 181
6.1.3. 监控节点 182
6.1.4. 管理节点标签 183
6.1.5. 设置节点调度 184
6.1.6. 批量运维节点 185
6.1.7. Master节点的资源升配 185
6.1.8. Worker节点的资源升配 188
6.1.9. 为容器服务的Docker增加数据盘 190
6.1.10. 集群节点挂载数据盘 192
6.1.11. 管理污点 194
6.1.12. 一键采集节点的诊断日志 195
6.1.13. 查看节点资源请求量和使用量 197
6.1.14. 查看节点列表 198
6.1.15. 移除节点 198
6.2. 节点池 199
6.2.1. 节点池概述 199
6.2.2. 管理节点池 202
6.2.3. 调度应用至指定节点池 206
6.2.4. 配置节点池内抢占式实例和存量实例的比例 208
6.2.5. 查看Spot实例到期状态 210
6.2.6. 查看节点池弹性强度 211
6.3. 托管节点池 214

6.3.1. 托管节点池概述 214
6.3.2. 管理托管节点池 215
6.3.3. 调度应用至指定节点池 218
6.4. 节点资源预留策略 221
6.5. 节点管理FAQ 222
7.网络 226
7.1. 网络概述 226
7.2. Kubernetes集群网络规划 230
7.3. 容器网络CNI 232
7.3.1. 使用Terway网络插件 232
7.3.2. 扩容Terway场景下的虚拟交换机 236
7.3.3. 使用VPC附加网段扩充集群的虚拟交换机 238
7.3.4. 使用网络策略Network Policy 240
7.3.5. 使用Host网络 257
7.3.6. 为Pod挂载独立公网EIP 258
7.3.7. 使用VPC的多路由表功能 261
7.3.8. 为已有集群开启SNAT公网访问能力 262
7.3.9. 容器网络FAQ 265
7.4. Service管理 273
7.4.1. Service的负载均衡配置注意事项 273
7.4.2. 通过Annotation配置负载均衡 276
7.4.3. 通过使用已有SLB的服务公开应用 300
7.4.4. 通过使用自动创建SLB的服务公开应用 306
7.4.5. 通过CCM跨集群部署服务 312
7.4.6. 管理服务 316
7.4.7. Service FAQ 321
7.5. Ingress管理 327
7.5.1. Ingress概述 327

7.5.2. 创建Ingress路由 329
7.5.3. Nginx Ingress管理 335
7.5.3.1. Ingress高级用法 335
7.5.3.2. 通过Ingress实现灰度发布和蓝绿发布 348
7.5.3.3. 使用Ingress-sentinel实现流控 357
7.5.3.4. 使用Ingress-tracing实现链路追踪 359
7.5.3.5. Ingress访问日志分析与监控 361
7.5.3.6. 部署高负载场景的Ingress Controller 368
7.5.3.7. 部署高可靠Ingress Controller 373
7.5.3.8. 部署Ingress Controller使用私网SLB 375
7.5.3.9. 部署多个Ingress Controller 378
7.5.3.10. 通过Ingress Controller来实现应用的流量复制 382
7.5.3.11. 通过Ingress Controller实现gRPC服务访问 388
7.5.4. ALB Ingress管理 394
7.5.4.1. 通过ALB Ingress访问服务 394
7.5.4.2. ALB Ingress服务高级用法 398
7.5.4.3. 为ACK专有版集群中的ALB Ingress Controller授权 410
7.5.4.4. 配置Albconfig 413
7.5.4.5. 为ALB Ingress开启WAF防护暂时只支持中国站 416
7.5.5. Ingress FAQ 417
7.6. 服务发现DNS 424
7.6.1. 基于DNS的服务发现概述 424
7.6.2. ACK集群内DNS原理和配置说明 426
7.6.3. 优化集群DNS性能 434
7.6.4. 在ACK集群中使用NodeLocal DNSCache 436
7.6.5. 使用ACK CoreDNS DNSTAP Analyser诊断CoreDNS域名解析.. 445
7.6.6. CoreDNS升级说明 448
7.6.7. Kubernetes域名解析异常问题排查 452

7.6.7.1. 诊断流程与排查思路 452
7.6.7.2. 常见检查方法 455
7.6.7.3. 常见问题及解决方案 459
7.7. 网络管理FAQ 464
8.应用 466
8.1. 工作负载 466
8.1.1. 创建无状态工作负载Deployment 466
8.1.2. 创建有状态工作负载StatefulSet 482
8.1.3. 创建守护进程集工作负载DaemonSet 492
8.1.4. 创建任务工作负载Job 495
8.1.5. 创建定时任务工作负载CronJob 499
8.1.6. 管理容器组（Pod） 503
8.1.7. 管理自定义资源 504
8.2. 镜像 505
8.2.1. kritis-validation-hook组件介绍 505
8.2.2. 管理镜像 509
8.2.3. 使用免密组件拉取容器镜像 509
8.2.4. 使用kritis-validation-hook组件实现自动验证容器镜像签名 516
8.3. 配置项及保密字典 519
8.3.1. 管理配置项 519
8.3.2. 在容器组中使用配置项 520
8.3.3. 管理保密字典 524
8.3.4. 在容器组中使用保密字典 525
8.4. 应用调度部署 529
8.4.1. 调度应用Pod至指定节点 529
8.4.2. 使用Descheduler组件对Pod进行调度优化 531
8.4.3. 使用应用触发器重新部署应用 535
8.4.4. 使用Helm简化应用部署 536

8.4.5. 通过控制台部署、发布和监控应用 539
8.5. 使用AHAS对应用进行高可用防护 542
8.6. 应用FAQ 544
9.组件 548
9.1. 组件概述 548
9.2. 管理组件 550
9.3. metrics-server组件常见问题 551
9.4. ack-arms-prometheus组件常见问题 553
10.存储-CSI 555
10.1. 存储CSI概述 555
10.2. 存储基础知识 562
10.3. 安装与升级CSI组件 565
10.4. 云盘存储卷 566
10.4.1. 云盘存储卷概述 567
10.4.2. 使用云盘静态存储卷 570
10.4.3. 使用云盘动态存储卷 576
10.4.4. 在线扩容云盘数据卷 584
10.4.5. 手动扩容云盘数据卷 586
10.4.6. 自动扩容云盘数据卷（公测） 592
10.4.7. 使用云盘存储快照 597
10.4.8. 加密云盘存储卷 606
10.4.9. 云盘存储卷FAQ 608
10.5. 容器网络文件系统 611
10.5.1. 容器网络文件系统CNFS概述 611
10.5.2. 使用CNFS托管NAS文件系统 612
10.5.3. 使用CNFS自动扩容NAS存储卷 623
10.5.4. 使用CNFS动态创建NAS共享存储卷 628
10.5.5. 使用CNFS的回收站功能恢复数据 633

10.6. NAS存储卷 638
10.6.1. NAS存储卷概述 638
10.6.2. 使用NAS静态存储卷 639
10.6.3. 使用NAS动态存储卷 646
10.6.4. 配置NAS存储卷容量 658
10.6.5. NAS存储卷FAQ 661
10.7. OSS存储卷 662
10.7.1. OSS存储卷概述 662
10.7.2. 使用OSS静态存储卷 663
10.7.3. 加密OSS存储卷 672
10.7.4. OSS存储卷FAQ 675
10.8. CPFS存储卷 675
10.8.1. CPFS存储卷概述 675
10.8.2. CPFS静态卷 676
10.8.3. CPFS动态卷 680
10.9. 本地存储卷 684
10.9.1. 本地存储卷概述 684
10.9.2. HostPath数据卷 685
10.9.3. LocalVolume数据卷 688
10.9.4. LVM数据卷 689
10.9.5. QuotaPath数据卷 695
10.9.6. 内存数据卷 699
10.9.7. 自动初始化本地存储资源 700
10.10. AEP非易失性存储卷 705
10.10.1. AEP非易失性数据卷概述 705
10.10.2. 使用AEP非易失性存储提升读写性能 711
10.10.3. 使用AEP的直接内存访问方式部署Redis数据库 717
10.11. 容器存储监控 720

10.11.1. 使用csi-plugin组件监控节点侧存储资源 720
10.11.2. 使用storage-operator组件监控集群侧存储资源 724
10.12. 容器存储运维 725
10.12.1. 使用storage-operator进行存储组件的部署与升级 725
10.13. 存储FAQ-CSI 729
11.存储-Flexvolume 733
11.1. 存储Flexvolume概述 733
11.2. 安装与升级Flexvolume组件 733
11.3. 云盘存储卷 735
11.3.1. 云盘存储卷概述 735
11.3.2. 使用云盘静态存储卷 739
11.3.3. 通过命令行使用动态云盘卷 741
11.3.4. 通过控制台使用动态云盘卷 744
11.3.5. 使用云盘静态存储卷实现持久化存储-Flexvolume 746
11.3.6. 使用云盘动态存储卷实现持久化存储-Flexvolume 751
11.3.7. 在ACK中实现Flexvolume云盘在线扩容 755
11.3.8. 手动扩容云盘数据卷 759
11.3.9. 云盘存储卷FAQ 765
11.4. NAS存储卷 766
11.4.1. NAS存储卷概述 766
11.4.2. 使用NAS静态存储卷 767
11.4.3. 使用NAS动态存储卷 770
11.4.4. 使用NAS实现共享存储和持久化存储-Flexvolume 772
11.4.5. NAS存储卷FAQ 777
11.5. OSS存储卷 779
11.5.1. OSS存储卷概述 779
11.5.2. 使用OSS静态存储卷 780
11.5.3. 使用OSS实现持久化存储-Flexvolume 783

11.5.4. OSS存储卷FAQ 788
11.6. CPFS存储卷 789
11.6.1. 如何在ACK集群中使用CPFS存储卷服务 789
11.7. 创建持久化存储卷声明 795
11.8. 使用持久化存储卷声明 796
11.9. 存储FAQ-Flexvolume 797
12.安全 801
12.1. 安全体系概述 801
12.2. 基础设施安全 803
12.2.1. 在Kubernetes中实现HTTPS安全访问 803
12.2.2. 使用集群审计功能 808
12.2.3. 部署服务账户令牌卷投影 815
12.2.4. 使用Pod安全策略 817
12.2.5. 安全巡检 818
12.2.6. 使用ack-kubernetes-webhook-injector动态配置阿里云产品... 820
12.2.7. 自定义集群API Server证书SAN 823
12.3. 容器应用安全 824
12.3.1. 使用PSP安全策略 824
12.3.2. 使用配置巡检检查集群Workload安全隐患 829
12.3.3. 使用运行时刻安全监控 836
12.4. 容器安全FAQ 837
13.可观测性 843
13.1. 可观测性体系概述 843
13.2. 日志管理 843
13.2.1. 日志概述 843
13.2.2. 通过日志服务采集Kubernetes容器日志 844
13.2.3. 为Kubernetes和日志服务配置Log4JAppender 850
13.2.4. 收集托管集群控制平面组件日志 855

13.2.5. 分析和监控CoreDNS日志 857
13.2.6. 收集系统插件日志 862
13.3. 监控管理 865
13.3.1. 基础资源监控 865
13.3.2. 应用性能监控 870
13.3.3. 架构感知监控 875
13.3.4. 事件监控 876
13.3.5. 开源Prometheus监控 889
13.3.6. 阿里云Prometheus监控 901
13.3.7. Ingress Dashboard监控 911
13.3.8. 使用ACK Terway Hubble实现网络可观测性 915
13.3.9. 卸载ARMS Prometheus和ACK Prometheus 919
13.4. 容器服务报警管理 919
13.5. 可观测性FAQ 937
14.成本分析 938
14.1. 集群成本分析 938
15.弹性伸缩 942
15.1. 弹性伸缩概述 942
15.2. 节点自动伸缩 943
15.3. 容器水平伸缩（HPA） 954
15.4. 容器定时伸缩（CronHPA） 957
15.5. 容器垂直伸缩（VPA） 963
15.6. 基于阿里云组件指标的容器水平伸缩 967
15.7. 基于阿里云Prometheus指标的容器水平伸缩 971
15.8. 使用ECI弹性调度 977
15.9. 事件驱动弹性 977
15.10. 弹性伸缩FAQ 982
16.发布 988

16.1. 灰度发布 988
16.1.1. 灰度发布概述 988
16.1.2. 创建灰度发布 989
16.1.3. 灰度发布的高级选项 989
16.1.4. 使用手动发布模式创建灰度发布 990
16.1.5. 使用自动发布模式创建灰度发布 993
16.2. 使用OpenKruise部署云原生应用 995
16.3. 通过Ingress实现灰度发布和蓝绿发布 1002
16.4. Kubernetes集群中使用阿里云SLB实现四层金丝雀发布 1011
16.5. 基于Helm的发布管理 1015
17.Knative 1019
17.1. Knative概述 1019
17.2. Knative版本发布说明 1020
17.2.1. Knative发布0.18.3版本说明 1020
17.3. Knative组件管理 1021
17.3.1. 一键部署Knative 1021
17.3.2. 部署Knative组件 1022
17.3.3. 升级组件 1023
17.3.4. 采集组件日志 1023
17.3.5. 设置组件监控告警 1025
17.3.6. 卸载Knative组件 1028
17.3.7. 卸载Knative 1028
17.4. Knative服务管理 1029
17.4.1. 快速部署Serverless应用 1029
17.4.2. 创建修订版本 1031
17.4.3. 在Knative使用自定义域名 1032
17.4.4. 删除修订版本 1033
17.4.5. 删除Knative服务 1033

17.4.6. 在Knative中基于流量灰度发布服务 1033
17.5. Knative事件驱动 1036
17.5.1. 事件驱动概述 1036
17.5.2. 在Knative上实现GitHub事件处理 1037
17.5.3. 在Knative上使用MnsOss事件源 1040
17.6. Knative最佳实践 1043
17.6.1. 服务监控告警 1043
17.6.2. 在Knative上实现日志采集 1047
17.6.3. 在Knative中使用ECI 1050
17.6.4. 在Knative中通过Prometheus采集Pod并发请求数 1053
17.6.5. 在Knative中使用HPA 1055
17.6.6. 基于流量请求数实现服务自动扩缩容 1056
18.GPU/NPU 1062
18.1. 创建异构计算集群 1062
18.1.1. 创建托管GPU集群 1062
18.1.2. 创建专有GPU集群 1070
18.1.3. 创建NPU集群 1079
18.1.4. 创建托管FPGA集群 1081
18.2. GPU调度 1091
18.2.1. 使用Kubernetes默认GPU调度 1091
18.2.2. 使用节点标签自定义GPU调度 1094
18.2.3. ACK调度GPU使用的节点标签说明 1098
18.2.4. GPU拓扑感知调度 1100
18.2.4.1. GPU拓扑感知调度概述 1100
18.2.4.2. 安装GPU拓扑感知组件 1101
18.2.4.3. Tensorflow分布式训练使用GPU拓扑感知调度 1103
18.2.4.4. Pytorch分布式训练使用GPU拓扑感知调度 1108
18.3. 共享GPU调度 1113

18.3.1. 共享GPU调度概述 1114
18.3.2. 基础版 1115
18.3.2.1. 安装共享GPU组件 1115
18.3.2.2. 运行共享GPU示例 1117
18.3.2.3. 监控和隔离GPU资源 1121
18.3.2.4. 关闭共享GPU显存隔离能力 1128
18.3.2.5. 基于节点池管理共享GPU 1130
18.3.3. 专业版 1135
18.3.3.1. 安装并使用共享GPU组件和资源工具 1136
18.3.3.2. 运行共享GPU示例 1138
18.3.3.3. 通过共享GPU实现多卡共享策略 1141
18.3.3.4. 基于节点池管理共享GPU 1143
18.3.3.5. 在ACK Pro版集群中将共享GPU基础版升级为共享GPU专... 1149
18.4. GPU监控 1150
18.4.1. 使用Kubernetes事件中心监控GPU异常 1150
18.4.2. 使用ARMS Prometheus监控集群GPU资源 1151
18.5. GPU弹性伸缩 1153
18.5.1. 基于GPU指标实现弹性伸缩 1154
18.6. GPU运维管理 1159
18.6.1. 升级GPU节点的Docker 1159
18.6.2. 升级节点cGPU版本 1161
18.6.3. 排查GPU监控常见问题 1164
18.6.4. 通过节点池创建新NVIDIA驱动版本的节点 1166
18.6.5. 通过节点池升级已有节点的NVIDIA驱动 1171
18.6.6. 通过节点池开启NVIDIA A100多实例GPU能力 1176
18.6.7. 手动升级GPU节点驱动 1180
18.6.8. 修复GPU实例重启或被置换后设备ID变更问题 1182
18.6.9. 更新ACK集群中GPU虚拟化型（vGPU）实例的NVIDIA驱动Li... 1187

18.7. NPU调度 1188
18.7.1. Kubernetes集群支持NPU调度 1188
18.8. GPU/NPU FAQ 1190
19.调度 1191
19.1. CPU拓扑感知调度 1191
19.1.1. CPU拓扑感知调度 1191
19.1.2. 通过resource-controller组件动态修改Pod资源上限 1200
19.1.3. 通过控制L3 cache和MBA提高不同优先级任务的隔离能力 1205
19.1.4. 通过控制动态水位提高不同优先级负载的资源利用效率 1209
19.2. GPU调度 1213
19.3. FPGA调度 1213
19.3.1. 利用阿里云Kubernetes的FPGA节点标签进行调度 1214
19.4. 任务调度 1216
19.4.1. Gang scheduling 1216
19.4.2. Capacity Scheduling 1220
19.5. 弹性调度 1228
19.5.1. 使用ECI弹性调度 1228
19.6. 负载感知调度 1229
20.命名空间与配额 1230
20.1. 管理命名空间 1230
20.2. 设置资源配额和限制 1231
21.服务网格ASM 1234
22.应用市场 1235
22.1. 容器镜像服务ACR 1235
22.2. 模板管理 1235
22.2.1. 创建编排模板 1235
22.2.2. 更新编排模板 1236
22.2.3. 另存编排模板 1236

22.2.4. 下载编排模板 1237
22.2.5. 删除编排模板 1237
22.3. 应用目录管理 1238
22.3.1. 应用目录概述 1238
22.3.2. 查看应用目录列表 1239
23.弹性容器实例ECI 1240
23.1. 通过部署ACK虚拟节点组件创建ECI Pod 1240
23.2. 配置ECI Profile 1244
23.3. 基于ECI运行Job任务 1247
23.4. 部署Ingress应用 1249
23.5. 使用ECI弹性调度 1252
23.6. 虚拟节点基于云解析PrivateZone的服务发现 1252
23.7. 安装virtual-kubelet-autoscaler 1255
23.8. 安装Elastic Workload 1256
24.Windows容器 1262
24.1. 创建Windows节点池 1262
24.2. 创建Windows应用 1263
24.3. 使用Logtail收集Windows容器日志 1265
24.4. 在Windows容器中使用云盘及基于SMB的文件存储 1273
24.5. 在Windows容器中使用基于DirectX的GPU加速 1279
25.注册集群 1284
25.1. 注册集群概述 1284
25.2. 混合集群部署 1291
25.2.1. 创建注册集群并接入本地数据中心集群 1291
25.2.2. 创建混合集群 1294
25.2.3. 通过SAG vCPE实现云上ACK与云下K8s集群互通 1294
25.3. 配置混合集群网络 1301
25.4. 容器镜像 1307

25.4.1. 通过自建Kubernetes集群的免密组件拉取容器镜像 1308
25.4.2. 通过image-syncer工具迁移容器镜像 1312
25.5. 弹性伸缩 1314
25.5.1. 创建节点池并扩容 1314
25.5.2. 配置自动弹性伸缩 1316
25.5.3. 使用自定义节点添加脚本 1318
25.5.4. 扩容弹性容器实例ECI 1320
25.6. 运维 1324
25.6.1. 可观测性 1324
25.6.1.1. 将日志服务接入注册集群 1324
25.6.1.2. 将事件中心接入注册集群 1326
25.6.1.3. 将报警配置功能接入注册集群 1330
25.6.1.4. 将应用实时监控服务ARMS接入注册集群 1341
25.6.1.5. 将arms-prometheus接入注册集群 1342
25.6.1.6. 将alibaba-cloud-metrics-adapter接入注册集群 1343
25.6.2. 微服务 1344
25.6.2.1. 将MSE接入注册集群 1345
25.6.2.2. 将AHAS-架构感知接入注册集群 1346
25.6.2.3. 将AHAS-应用限流接入注册集群 1347
25.6.3. 集群成本分析 1348
25.7. 使用Git源仓库创建应用 1352
25.8. 在注册集群中使用阿里云CSI存储 1354
25.9. 安全 1357
25.9.1. 在注册集群中使用集群审计功能 1357
25.9.2. 在注册集群中使用安全巡检功能 1361
25.9.3. 使用配置巡检功能检查注册集群Workload安全隐患 1365
25.10. 注册集群FAQ 1372
26.多集群管理 1373

26.1. 应用中心 1373
26.1.1. 应用中心概述 1373
26.1.2. 应用管理 1374
26.1.2.1. 使用编排模板创建应用 1374
26.1.2.2. 使用Git源仓库创建应用 1378
26.1.2.3. 管理应用 1381
26.1.3. 触发器 1382
26.1.3.1. 触发器概述 1382
26.1.3.2. 使用触发器自动更新应用 1382
26.1.3.3. 使用Jenkins持续集成自动更新应用 1384
26.1.3.4. 使用容器镜像触发器自动更新应用 1386
26.1.4. 多集群发布 1390
26.1.4.1. 使用编排模板部署应用到多集群 1390
26.1.4.2. 多地域集群应用灰度发布 1394
26.2. 备份中心 1396
26.2.1. 备份中心概述 1396
26.2.2. 开启集群备份 1400
26.2.3. 备份和恢复应用 1403
26.2.4. 跨集群迁移应用 1408
26.3. 服务网格 1415
27.安全沙箱 1417
27.1. 安全沙箱概述 1417
27.2. runC容器和安全沙箱（runV）容器的区别 1418
27.3. 如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时.. 1420
27.4. 为什么选择安全沙箱? 1424
27.5. 创建安全沙箱集群 1428
27.5.1. 创建安全沙箱托管版集群 1428
27.5.2. 创建安全沙箱专有版集群 1436

27.6. 扩容安全沙箱集群 1446
27.7. 创建安全沙箱应用 1448
27.8. 升级安全沙箱运行时 1458
27.9. 安全沙箱配置 1460
27.9.1. 配置安全沙箱节点和Docker节点的混合集群 1460
27.9.2. 配置安全沙箱Pod内核参数 1462
27.10. 安全沙箱存储 1464
27.10.1. 安全沙箱的NAS直挂 1464
27.10.2. 安全沙箱的云盘直挂 1468
27.11. 安全沙箱兼容性说明 1471
28.ACK-TEE机密计算 1473
28.1. ACK-TEE机密计算介绍 1473
28.2. 创建加密计算托管集群 1476
28.3. 创建加密计算节点池 1486
28.4. 通过SDK开发和构建SGX2.0应用 1487
28.5. 在ACK-TEE集群中使用Inclavare Containers机密容器 1497
28.6. 在ACK-TEE集群中使用Inclavare Containers机密容器实现远程证... 1502

1.ACK集群概述

Kubernetes是主流的开源容器编排平台，用于管理容器化应用和服务。本文介绍ACK集群的功能、类型和使用限制等。

背景信息

容器服务ACK面向多种业务场景提供多样化的Kubernetes集群：

- ACK集群：适合大多数业务场景，是一种最通用的Kubernetes集群。
- ASK集群：适合快速伸缩的敏捷业务场景以及单个或多个任务处理的场景。更多信息，请参见ASK概述。
- 边缘集群：是IoT、CDN等边缘业务的必选。更多信息，请参见ACK@Edge概述。

此外，容器服务ACK在基因计算、AI大数据等领域提供了高度集成的解决方案，结合IaaS高性能计算、网络能力，发挥容器的最佳性能。在多云混合云领域，容器服务ACK提供了多集群统一管理能力，您可在容器服务控制台，统一管理来自线下IDC，或者其他云上的Kubernetes集群。

使用限制

关于ACK集群的使用限制，请参见使用限制。

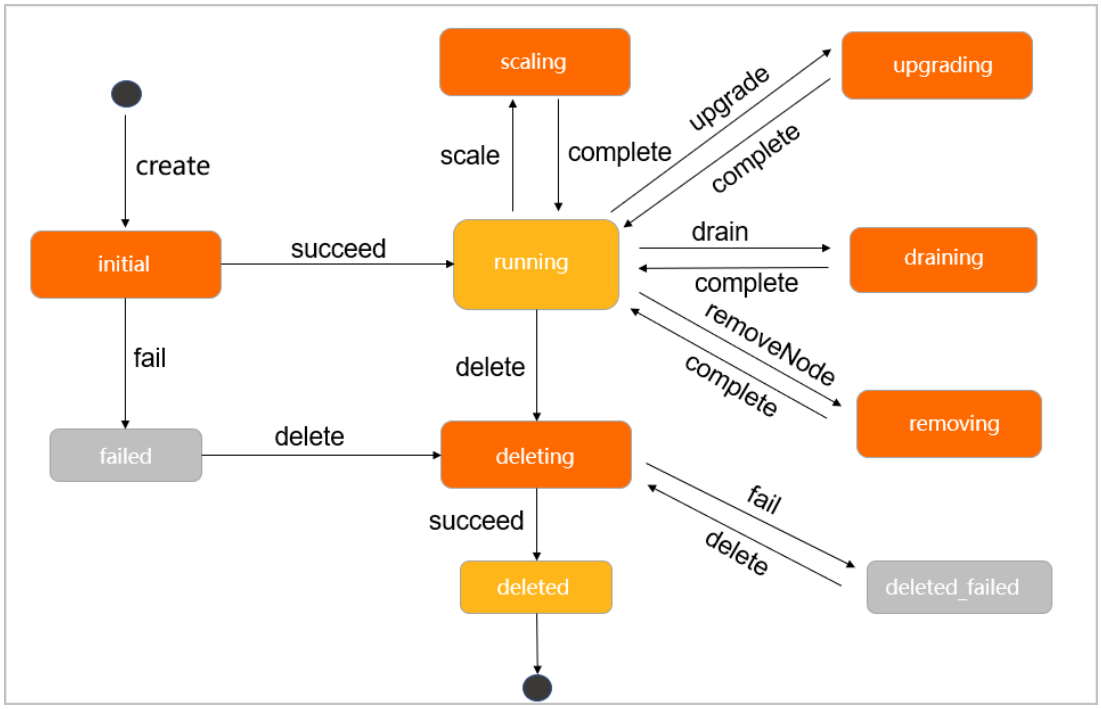
集群生命周期

ACK集群在不同状态下的含义和集群的状态流转图如下。

集群状态说明

状态	说明
初始化中（initial）	正在申请集群相应的云资源。
创建失败（failed）	申请集群云资源失败。
运行中（running）	成功申请集群云资源。
更新中（updating）	更新集群元信息。
扩容中（scaling）	扩容或添加集群节点。
移除节点中（removing）	移除集群节点中。
升级中（upgrading）	升级集群中。
驱逐节点中（draining）	正在驱逐节点中的Pod资源到其他节点中重新创建，之后该节点将不可调度。
删除中（deleting）	删除集群中。
删除失败（delete_failed）	删除集群失败。
已删除（deleted，该状态用户不可见）	成功删除集群。

集群状态流转



集群类型

ACK集群包括Pro版本、标准版、专有版三种类型。

比较项	ACK Pro版	ACK标准版	ACK专有版
主要特点	您只需创建Worker节点，Master节点由ACK创建并托管。		您需要自行创建Master节点及Worker节点。
	简单、低成本、高可用，无需管理Master节点。 关于ACK集群标准版和Pro版本的具体区别，请参见对比。	简单、低成本、高可用，无需管理Master节点。	可以对集群基础设施进行更细粒度的控制，需要自行规划、维护、升级服务器集群。
收费方式	收取集群管理费用，按照集群数量方式收费，同时但需要承担Worker节点以及其他基础资源的费用。更多信息，请参见产品计费。	集群管理免费，但需要承担Worker节点以及其他基础资源的费用。更多信息，请参见产品计费。	集群管理免费，但需要承担Master节点、Worker节点以及其他基础资源的费用。更多信息，请参见产品计费。
应用场景	适用于企业生产与测试环境。	适用于个人学习与测试。	适用于K8s研究与定制化场景。
用户画像			

比较项	ACK Pro版	ACK标准版	ACK专有版
集群创建			

产品功能

ACK集群支持的功能如下表。

功能	描述
集群管理	• 集群创建：您可根据需求创建多种形态集群，选择类型丰富的工作节点，并进行灵活的自定义配置。更多信息，请参见创建ACK Pro版集群、创建Kubernetes托管版集群和创建Kubernetes专有版集群。 • 集群升级：一键升级K8s版本，统一管理系统组件升级。更多信息，请参见升级ACK集群K8s版本。 • 弹性伸缩：通过控制台一键垂直扩缩容来快速应对业务波动，同时支持服务级别的亲和性策略和横向扩展。 • 多集群管理：支持线下IDC和多云多区域的集群统一接入实现混合云应用管理。 • 授权管理：支持RAM授权和RBAC权限管理。
节点池	支持节点池生命周期管理，支持在同一集群中配置不同规格的节点池，例如交换机、运行时、OS、安全组等。更多信息，请参见节点池概述。

功能	描述
应用管理	• 应用创建：支持多种类型应用，从镜像、模版的创建，支持环境变量、应用健康、数据盘、日志等相关配置。 • 应用全生命周期：支持应用查看、更新、删除，应用历史版本回滚、应用事件查看、应用滚动升级、应用替换升级以及通过触发器重新部署应用。 • 应用调度：支持节点间亲和性调度、应用间亲和性调度、应用间反亲和性调度三种策略。 • 应用伸缩：支持手动伸缩应用容器实例，HPA自动伸缩策略。 • 应用发布：支持灰度发布和蓝绿发布。 • 应用目录：支持应用目录，简化云服务集成。 • 应用中心：应用部署后，以统一的视角展现整体应用的拓扑结构，同时对于持续部署等场景进行统一的版本管理与回滚。 • 应用备份和恢复：支持对Kubernetes应用进行备份和恢复。更多信息，请参见备份和恢复应用。

存储	• 存储插件：支持Flexvolume以及CSI存储插件。更多信息，请参见存储CSI概述和存储Flexvolume概述。
	• 存储卷和存储声明：
	  ◦ 支持创建块存储、NAS、OSS、CPFS类型的存储卷。
	  ◦ 支持持久化存储卷声明（PVC）挂接存储卷。
	  ◦ 支持存储卷的动态创建和迁移。
	  ◦ 支持以脚本方式查看和更新存储卷和存储声明。

网络	• 支持Flannel容器网络和Terway容器网络。更多信息，请参见网络概述。 • 支持定义Sevice和Pod的CIDR。 • 支持NetworkPolicy。更多信息，请参见使用网络策略Network Policy。 • 支持路由Ingress。 • 支持服务发现DNS。更多信息，请参见基于DNS的服务发现概述。

运维与安全	• 可观测性：
	  ◦ 监控：支持集群、节点、应用、容器实例层面的监控；支持prometheus插件。
	  ◦ 日志：支持集群日志查看；支持应用日志采集；支持容器实例日志查看。
	  ◦ 报警：支持容器服务异常事件报警，以及容器场景指标报警。更多信息，请参见容器服务报警管理。
	• 成本分析：支持可视化集群资源使用量及成本分布，以提升集群资源利用率。
	• 安全中心：支持运行时刻的安全策略管理，应用安全配置巡检和运行时刻的安全监控和告警，提升容器安全整体纵深防御能力。
	• 安全沙箱：可以让应用运行在一个轻量虚拟机沙箱环境中，拥有独立的内核，具备更好的安全隔离能力。适用于不可信应用隔离、故障隔离、性能隔离、多用户间负载隔离等场景。
	• 机密计算：基于Intel SGX提供的可信应用或用于交付和管理机密计算应用的云原生一站式机密计算平台，帮助您保护数据使用中的安全性、完整性和机密性。机密计算可以让您把重要的数据和代码放在一个特殊的可信执行加密环境。

功能	描述
异构计算	• GPU和NPU：支持创建以GPU或NPU型实例作为工作节点的集群，并支持GPU调度、GPU监控、GPU弹性伸缩、GPU运维管理等。更多信息，请参见创建托管GPU集群。 • 共享GPU：支持在云平台和自己的数据中心的集群中通过GPU共享调度框架实现多个容器运行在同一个GPU设备。更多信息，请参见共享GPU调度概述。 • 云原生AI：提供了云原生AI能力，支持编排、管理数据计算类任务。更多信息，请参见云原生AI套件概述。
开发者服务	• API：提供OpenAPI和社区原生API。更多信息，请参见使用Kubernetes API和API概览。 • Cloudshell：提供阿里云Cloudshell和社区原生Kubectl工具。更多信息，请参见在CloudShell上通过kubectl管理Kubernetes集群和通过kubectl工具连接集群。

开源项目

有关容器服务ACK使用的开源项目，请参见开源项目。

常见问题

- 授权管理FAQ
- 集群管理FAQ
- 节点管理FAQ
- 容器网络FAQ
- Service FAQ
- Ingress FAQ
- 注册集群FAQ
- 应用FAQ
- 容器安全FAQ
- 存储FAQ-CSI
- 存储FAQ-Flexvolume
- GPU/NPU FAQ
- 弹性伸缩FAQ

2.首次使用容器服务Kubernetes版

您首次使用容器服务Kubernetes版（Alibaba Cloud Container Service for Kubernetes，简称容器服务ACK）时，需要为服务账号授予系统默认角色。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。本文介绍在首次使用ACK时如何授权容服务默认角色以及开通相关云产品。

步骤一：开通容器服务ACK

容器服务ACK现已正式商用，在创建ACK集群前您需要开通相应服务。操作步骤如下：

1. 登录容器服务ACK开通页面。
2. 阅读并选中**容器服务ACK服务协议**。
3. 单击**立即开通**。

> **说明** 如果您在创建ACK集群前未进行开通容器服务ACK的操作，在创建集群页面的**依赖检查**区域，系统会提示您进行开通容器服务ACK的操作。

步骤二：一键授权默认角色

首次登录容器服务ACK时，需要为服务账号授予系统默认角色。步骤如下：

> **说明** 阿里云账号（主账号）和拥有管理权限的RAM用户（子账号）都可以授权系统默认角色。

1. 登录容器服务管理控制台。
2. 单击**前往RAM进行授权**进入云资源访问授权页面，然后单击**同意授权**。
3. 完成以上授权后，刷新控制台即可使用容器服务ACK。

步骤三：开通相关云产品

容器服务ACK相关功能依赖或关联其他的云资源服务，因此您需要开通相关云产品。

> **说明** 只有阿里云账号（主账号）才可以开通云产品，RAM用户暂不支持开通云产品操作。

使用阿里云账号（即主账号）登录阿里云官网，根据需要开通以下相关云产品。

- 必选项：使用ACK集群的强依赖服务，必须开通。
- 建议项：集群创建和应用管理中常见的依赖服务，建议开通。
- 可选项：根据业务架构和运维需求选择性开通。

产品名称	开通链接	开通类型	产品说明
VPC专有网络	https://www.aliyun.com/product/vpc	必选项	用于构建集群网络环境和路由规则
NAT网关服务	https://www.aliyun.com/product/nat	必选项	用于为集群开启公网访问和公网镜像拉取
SLB负载均衡	https://www.aliyun.com/product/slb	必选项	用于为集群创建负载均衡
ESS弹性伸缩服务	https://www.aliyun.com/product/ess	必选项	用于为集群创建Worker节点和实现自动伸缩

产品名称	开通链接	开通类型	产品说明
ACR容器镜像服务	https://www.aliyun.com/product/acr	建议项	用于云原生资产的安全托管和全生命周期管理
ECI弹性容器实例	https://www.aliyun.com/product/eci	建议项	用于部署ASK Serverless集群
ASM服务网格	https://www.aliyun.com/product/servicemesh	建议项	基于服务网格实现多个ACK集群应用的统一流量管理
SLS日志服务	https://www.aliyun.com/product/sls	建议项	用于ACK集群组件和应用的日志采集和检索
CMS云监控服务	https://www.aliyun.com/product/jiankong	建议项	用于监控集群节点和应用运行状态
阿里云Prometheus	https://www.aliyun.com/product/developerservices/prometheus	建议项	基于Prometheus实现对ACK集群的监控和告警
SAS云安全中心	https://www.aliyun.com/product/sas	可选项	用于监控集群应用运行时的安全事件和告警
NAS文件存储	https://www.aliyun.com/product/nas	可选项	基于NAS实现集群应用数据的文件存储方案
OSS对象存储	https://www.aliyun.com/product/oss	可选项	基于OSS实现集群应用数据的对象存储方案
KMS密钥管理服务	https://www.aliyun.com/product/kms	可选项	用于集群应用密钥的管理以及Pro集群开启密钥的落盘加密能力
PrivateZone服务	https://www.aliyun.com/product/pvtz	可选项	基于内网DNS服务实现ASK集群应用的域名访问

容器服务默认角色说明

角色名称	角色说明
AliyunCSDefaultRole	ACK在集群管控操作中使用该角色访问您在ECS、VPC、SLB、ROS、ESS等服务中的资源。
AliyunCSManagedKubernetesRole	ACK托管版集群默认使用该角色来访问您在ECS、VPC、SLB、ACR等服务中的资源。
AliyunCSServerlessKubernetesRole	ASK集群默认使用该角色来访问您在ECS、VPC、SLB、PVTZ等服务中的资源。
AliyunCSKubernetesAuditRole	ACK托管版集群和ASK集群的审计功能使用该角色来访问您在SLS服务中的资源。
AliyunCSManagedNetworkRole	ACK托管版集群和ASK集群的网络组件使用该角色访问您在ECS、VPC服务中的资源。
AliyunCSManagedCsiRole	ACK托管版集群和ASK集群的存储组件使用该角色访问您在ECS、NAS服务中的资源。

角色名称	角色说明
AliyunCSManagedCmsRole	ACK托管版集群和ASK集群的监控组件使用该角色访问您在CMS、SLS服务中的资源。
AliyunCSManagedLogRole	ACK托管版集群和ASK集群的日志组件使用该角色访问您在SLS服务中的资源。
AliyunCSManagedVKRole	ASK集群的Virtual Kubelet组件使用该角色访问您在ECS、VPC、ECI等服务中的资源。
AliyunCSManagedArmsRole	ACK托管版集群和ASK集群的应用实时监控组件使用该角色访问您在ARMS服务中的资源。

相关文档

- 入门概述
- 容器服务默认角色
- 快速创建Kubernetes托管版集群
- 使用镜像快速创建无状态Deployment应用
- 创建有状态工作负载StatefulSet

3.授权

3.1. 授权概述

容器服务ACK的授权体系包含RAM授权和RBAC授权两部分。本文介绍容器服务ACK集群访问控制授权的组成及方式。

RAM授权

在企业对接RAM的账号系统中，运维人员通过RAM用户管理云服务资源是一个常见场景。然而默认情况下RAM用户没有使用云服务OpenAPI的任何权限，为了保证RAM用户的正常使用，需要对RAM用户授权。

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。更多信息，请参见自定义RAM授权策略。

RAM授权支持以下两种方式：

- RAM系统策略授权：用于指定全局资源的读写访问控制。当RAM用户具有阿里云账号下所有集群的运维管理权限时，建议使用系统策略进行快捷授权。为RAM用户添加系统策略权限，请参见为RAM用户授权。
  容器服务的常用系统策略如下表所示，您可以根据业务需求添加对应的系统策略。

系统策略名称	说明
AliyunCSFullAccess	当RAM用户需要管理容器服务时的所有权限。
AliyunVPCReadOnlyAccess	当RAM用户在创建集群时选择指定VPC。
AliyunECSReadOnlyAccess	当RAM用户为集群添加已有节点或查看节点详细信息。
AliyunContainerRegistryFullAccess	当RAM用户需要全局管理阿里云账号内的业务镜像。
AliyunLogReadOnlyAccess	当RAM用户在创建集群时选择已有Log Project存储审计日志，或查看指定集群的配置巡检。
AliyunAHASReadOnlyAccess	当RAM用户需要使用集群拓扑功能。
AliyunRAMFullAccess	当RAM用户需要负责阿里云账号内的全局授权管理。
AliyunYundunSASReadOnlyAccess	当RAM用户需要查看指定集群的运行时安全监控。
AliyunARMSReadOnlyAccess	当RAM用户需要查看集群阿里云Prometheus插件的监控状态。
AliyunKMSReadOnlyAccess	当RAM用户在创建Pro集群时启用Secret落盘加密能力。

- RAM自定义策略授权：用于对目标RAM用户实现细粒度的云资源访问控制。另外，如果目标RAM用户有基于SDK的二次开发需求，还可实现API级别的权限控制。
  ACK集群维度的RAM自定义策略授权流程，请参见自定义RAM授权策略。
  例如，当RAM用户需要具有对指定OSS Bucket的读取权限时，可以参考以下策略。

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
            "oss:ListBuckets",
            "oss:GetBucketStat",
            "oss:GetBucketInfo",
            "oss:GetBucketTagging",
            "oss:GetBucketAcl"
            ],
      "Resource": "acs:oss:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "oss:ListObjects",
        "oss:GetBucketAcl"
      ],
      "Resource": "acs:oss:*:*:myphotos"
    },
    {
      "Effect": "Allow",
      "Action": [
        "oss:GetObject",
        "oss:GetObjectAcl"
      ],
      "Resource": "acs:oss:*:*:myphotos/*"
    }
  ]
}
```

Role和ClusterRole

RBAC的Role或ClusterRole中包含一组代表相关权限的规则。这些权限是纯粹累加的（不存在拒绝某操作的规则）。

Role总是用来在某个命名空间内设置访问权限；在您创建Role时，您必须指定该Role所属的命名空间。

与之相对，ClusterRole则是一个集群作用域的资源。这两种资源的名字不同（Role和ClusterRole）是因为Kubernetes对象是命名空间作用域或集群作用域的，不可两者兼具。

ClusterRole有若干用法。您可以用来：

- 定义对某命名空间域对象的访问权限，并将在各个命名空间内完成授权。
- 为命名空间作用域的对象设置访问权限，并跨所有命名空间执行授权。
- 为集群作用域的资源定义访问权限。

如果您需要在命名空间内定义角色，应该使用Role；如果您需要定义集群范围的角色，应该使用ClusterRole。关于如何编写Kubernetes的ClusterRole和Role，请参见自定义Kubernetes授权策略。

关于如何授予RAM用户自定义Kubernetes授权策略，请参见配置RAM用户RBAC权限。

> **注意** 当前容器服务ACK授权管理只支持自定义ClusterRole角色与集群内RBAC权限的绑定，不支持自定义Role角色与集群内RBAC权限的绑定。

RBAC授权

当RAM用户需要操作指定集群内K8s资源时（例如，获取集群Pod和Node信息），需要在容器服务管理控制台的授权管理页面对指定RAM用户进行数据平面资源的授权。

您可以给RAM用户授予以下预置角色。详细操作，请参见配置RAM用户RBAC权限。

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RAM用户获得不符合预期的权限。关于ClusterRole的更多信息，请参见自定义Kubernetes授权策略。

相关文档

- 授权管理FAQ
- 容器服务默认角色
-

3.2. 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RAM用户（即子账号）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。更多信息，请参见授权策略语言描述。

背景信息

容器服务ACK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。

在进行RAM用户的集群RBAC授权前，您需要完成对集群管控能力的RAM授权。您可以根据需要授予RAM用户对于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。

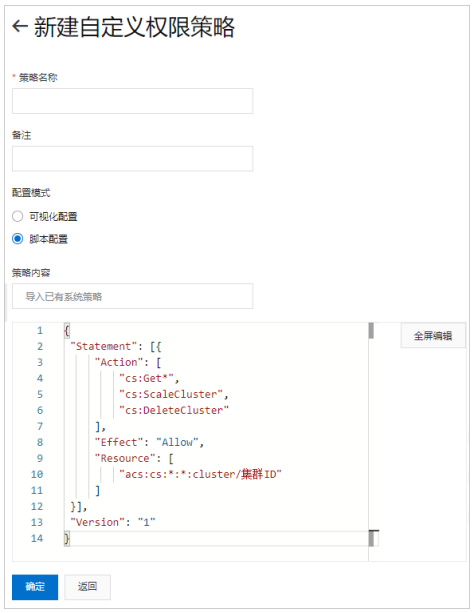
在提交RBAC授权前，您需要确保目标集群已经被授予RAM只读权限，策略参考如下。

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

当您完成RAM授权后，需要完成集群内Kubernetes资源模型访问的RBAC授权。具体操作，请参见配置RAM用户RBAC权限。

创建自定义授权策略

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**。
3. 在**权限策略管理**页面，单击**创建权限策略**。
4. 在**新建自定义权限策略**页面，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:List*",
     "cs:Describe*",
     "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` 有以下配置方式。
  - 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"
    ]
    ```

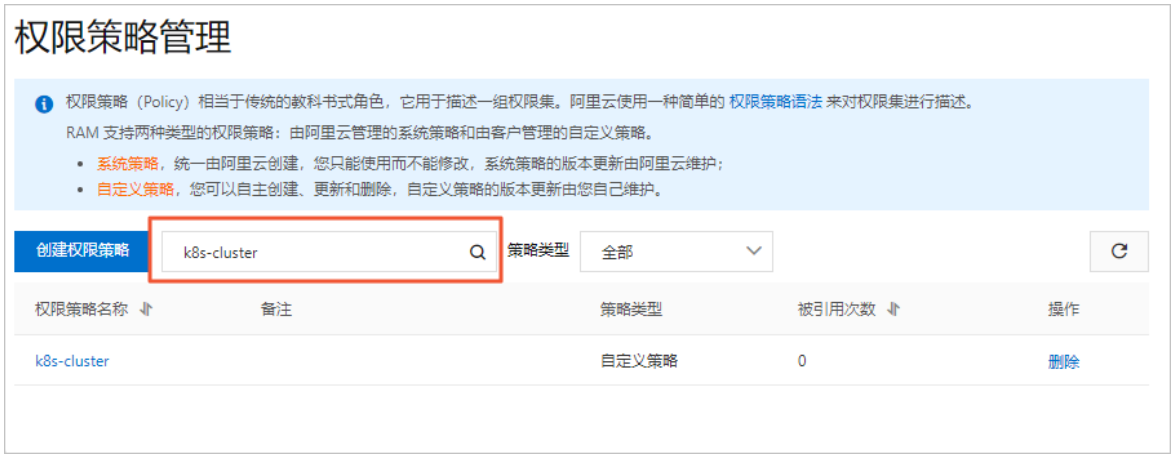
  - 授予所有集群的权限

    ```
    "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**。

   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

RAM Action和API名称的对应关系

API名称	RAM Action	说明	是否支持限制集群
DescribeEvents	cs:DescribeEvents	查询用户操作事件。	不支持
StartAlert	cs:StartAlert	启动报警规则。	不支持
StopAlert	cs:StopAlert	停止报警规则。	不支持
UpdateContactGroupForAlert	cs:UpdateContactGroupForAlert	更新告警联系人组。	不支持
DeleteAlertContact	cs:DeleteAlertContact	删除报警联系人。	不支持
DeleteAlertContactGroup	cs:DeleteAlertContactGroup	删除报警联系人分组。	不支持
DescribeUserPermission	cs:DescribeUserPermission	查询指定RAM用户的集群授权信息。	不支持
OpenAckService	cs:OpenAckService	开通容器服务ACK。	不支持
GrantPermissions	cs:GrantPermissions	全量更新RAM用户集群授权信息。	不支持
CreateCluster	cs:CreateCluster	创建ACK集群（包括ACK 专有版集群、ACK托管版集群、ASK集群，ACK边缘托管版集群以及注册集群）。	不支持
DescribeClusterResources	cs:DescribeClusterResources	根据集群ID查询该集群的所有资源。	支持
DescribeClusterDetail	cs:DescribeClusterDetail	根据集群ID查询该集群的详情。	支持
DescribeUserQuota	cs:DescribeUserQuota	查询集群及节点配额。	不支持
DescribeClustersV1	cs:DescribeClustersV1 和 cs:GetClusters	查询已创建的所有ACK集群的详情。	不支持

API名称	RAM Action	说明	是否支持限制集群
DescribeExternalAgent	cs:DescribeExternalAgent	根据集群ID查询集群中部署注册集群的代理配置。	支持
DescribeClusterLogs	cs:DescribeClusterLogs	根据集群ID查询集群日志。	支持
DescribeTaskInfo	cs:DescribeTaskInfo	根据任务ID查询该任务执行详情。	不支持
DescribeKubernetesVersionMetadata	cs:DescribeKubernetesVersionMetadata	查询ACK支持的Kubernetes版本详情。	不支持
DescribeClusterUserKubeconfig	cs:DescribeClusterUserKubeconfig	根据集群ID查询访问该集群的kubeconfig配置。	支持
DescribeClusterAddonUpgradeStatus	cs:DescribeClusterAddonUpgradeStatus	查询集群Addons升级状态。	支持
DescribeClusters	cs:DescribeClusters 和 cs:GetClusters	查看容器服务中创建的所有集群（包括Swarm和Kubernetes集群）。	不支持
DescribeClusterNamespaces	cs:DescribeClusterNamespaces	获取集群命名空间。	支持
ScaleOutCluster	cs:ScaleOutCluster	根据集群ID扩容集群。	支持
ModifyCluster	cs:ModifyCluster	根据集群ID修改该集群配置。	支持
MigrateCluster	cs:MigrateCluster	迁移集群。	支持
ScaleCluster	cs:ScaleCluster	扩容节点。	支持
UpdateK8sClusterUserConfigExpire	cs:UpdateK8sClusterUserConfigExpire	更新用户自定义配置过期时间。	支持
DeleteCluster	cs:DeleteCluster	根据集群ID删除集群实例，并释放集群所有节点资源。	支持
DescribeClusterNodes	cs:DescribeClusterNodes	根据集群ID查询该集群中所有节点的详情。	支持
AttachInstances	cs:AttachInstances	添加已有ECS实例到ACK集群。	支持
DescribeClusterAttachScripts	cs:DescribeClusterAttachScripts	查询手动添加实例到集群的脚本。	支持
DeleteClusterNodes	cs:DeleteClusterNodes	根据节点名称移除集群中指定节点。	支持
RemoveClusterNodes	cs:RemoveClusterNodes	移除指定集群额外节点。	支持

API名称	RAM Action	说明	是否支持限制集群
CreateClusterNodePool	cs:CreateClusterNodePool	为集群新建节点池。	支持
DescribeClusterNodePools	cs:DescribeClusterNodePools	根据集群ID查询该集群中所有节点池的详情。	支持
DescribeClusterNodePoolDetail	cs:DescribeClusterNodePoolDetail	根据节点池ID查询集群中该节点池的详情。	支持
ScaleClusterNodePool	cs:ScaleClusterNodePool	根据节点池ID扩容节点池节点。	支持
ModifyClusterNodePool	cs:ModifyClusterNodePool	根据节点池ID修改指定集群该节点池的配置。	支持
DeleteClusterNodepool	cs:DeleteClusterNodepool	根据节点池ID删除集群节点池。	支持
GetUpgradeStatus	cs:GetUpgradeStatus	根据集群ID查询该集群的升级状态。	支持
ResumeUpgradeCluster	cs:ResumeUpgradeCluster	根据集群ID恢复升级处于升级暂停状态的集群。	支持
UpgradeCluster	cs:UpgradeCluster	根据集群ID升级指定集群。	支持
PauseClusterUpgrade	cs:PauseClusterUpgrade	暂停集群升级。	支持
CancelClusterUpgrade	cs:CancelClusterUpgrade	取消处于升级状态的ACK集群。	支持
CreateTemplate	cs:CreateTemplate	创建一个编排模板。	不支持
DescribeTemplates	cs:DescribeTemplates	查询已创建的所有编排模板详情。	不支持
DescribeTemplateAttribute	cs:DescribeTemplateAttribute	根据编排模板ID查询该编排模板的详情。	不支持
UpdateTemplate	cs:UpdateTemplate	根据编排模板ID更新编排模板。	不支持
DeleteTemplate	cs:DeleteTemplate	根据编排模板ID删除指定编排模板。	不支持
InstallClusterAddons	cs:InstallClusterAddons	为集群安装组件。	支持
DescribeAddons	cs:DescribeAddons	查询平台支持的所有组件的详情。	不支持
DescribeClusterAddonsUpgradeStatus	cs:DescribeClusterAddonsUpgradeStatus	根据组件名称查询该组件升级状态。	支持

API名称	RAM Action	说明	是否支持限制集群
DescribeClusterAddonsVersion	cs:DescribeClusterAddonsVersion	根据集群ID查询集群中已安装的所有组件的详情。	支持
ModifyClusterConfiguration	cs:ModifyClusterConfiguration	修改托管版集群配置。	支持
UpgradeClusterAddons	cs:UpgradeClusterAddons	根据组件名称将指定组件升级到指定版本。	支持
PauseComponentUpgrade	cs:PauseComponentUpgrade	暂停组件升级。	支持
ResumeComponentUpgrade	cs:ResumeComponentUpgrade	重新开始组件升级。	支持
CancelComponentUpgrade	cs:CancelComponentUpgrade	取消集群组件升级。	支持
UnInstallClusterAddons	cs:UnInstallClusterAddons	根据组件名称卸载指定集群的组件。	支持
ListTagResources	cs:ListTagResources	根据集群ID查询指定集群资源的标签。	不支持
TagResources	cs:TagResources	为资源绑定标签。	不支持
ModifyClusterTags	cs:ModifyClusterTags	根据集群ID修改该集群的集群标签。	支持
UntagResources	cs:UntagResources	删除资源标签。	不支持
CreateTrigger	cs:CreateTrigger	创建触发器。	支持
DescribeTrigger	cs:DescribeTrigger	查询触发器。	支持
DeleteTrigger	cs:DeleteTrigger	删除触发器	支持

为RAM用户授权不支持限制集群的OpenAPI的操作权限

部分OpenAPI不支持限制集群的授权（例如，DescribeEvents，详见上方的表格。），若您需要给RAM用户授权这些OpenAPI不支持限制集群的OpenAPI的操作权限，请不要在Resource中限定集群ID。

例如，当前RAM权限策略如下：

```
{
  "Statement": [
    {
      "Action": [
        "cs:Get*",
        "cs:List*",
        "cs:Describe*"
      ],
      "Effect": "Allow",
      "Resource": [
        "acs:cs:*:*:cluster/c2e63856bcd714197****"
      ]
    }
  ],
  "Version": "1"
}
```

若您需要授权不支持限制集群的OpenAPI DescribeEvents，其对应的RAM Action为 `cs:DescribeEvents` ，修改后的RAM权限策略如下所示：

```
{
  "Statement": [
    {
      "Action": [
        "cs:DescribeEvents"
      ],
      "Effect": "Allow",
      "Resource": [
       "*"
      ]
    },
    {
      "Action": [
        "cs:Get*",
        "cs:List*",
        "cs:Describe*"
      ],
      "Effect": "Allow",
      "Resource": [
        "acs:cs:*:*:cluster/c2e63856bcd714197****"
      ]
    }
  ],
  "Version": "1"
}
```

3.3. 自定义Kubernetes授权策略

角色访问控制RBAC（Role-Based Access Control）的Kubernetes对象Role和ClusterRole中包含一组代表相关权限的规则。Role总是用来在某个命名空间内设置访问权限，而ClusterRole是为集群范围的资源定义访问权限。本文介绍如何自行编写Kubernetes的ClusterRole和Role。

前提条件

目前容器服务管理控制台的**安全管理 > 角色**功能处于灰度阶段，若您无法使用该功能，请提交工单申请添加白名单。

权限策略说明

您可自行编写权限策略，或通过容器服务管理控制台创建自定义策略。

Role：命名空间维度

如果您需要在命名空间内定义角色，则应该使用Role。

下方是一个位于default命名空间的Role的YAML示例，用来授予对Pods的所有权限。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: test-role
 namespace: default
rules:
- apiGroups:
 - ""
 resources:
 - pods
 verbs:
 - create
 - delete
 - deletecollection
 - get
 - list
 - patch
 - update
 - watch
```

ClusterRole：集群维度

如果您需要定义集群范围的角色，则应该使用ClusterRole。

下方是一个ClusterRole的示例，用来为任一特定命名空间中的Pods授予所有权限。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: test-clusterrole
rules:
- apiGroups:
 - ""
 resources:
 - pods
 verbs:
 - create
 - delete
 - deletecollection
 - get
 - list
 - patch
 - update
 - watch
```

关于Role和ClusterRole的更多信息，请参见Role和ClusterRole。

创建Kubernetes自定义授权策略

> **说明** 该步骤以为RAM用户创建自定义ClusterRole为例，与创建Role的步骤基本一致，您可结合实际需求进行操作。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**安全管理 > 角色**。
5. 在**角色**页面，单击**Cluster Role**页签。
6. 在**Cluster Role**页签，单击**创建**。
7. 在**创建YAML**面板输入自定义策略的YAML内容，单击**确定**即可创建ClusterRole。

   此步骤以权限策略说明中**ClusterRole：集群维度**的YAML为例，创建完成后，可在**Cluster Role**页签查看自定义权限test-clusterrole。

后续步骤

关于如何授予RAM用户自定义Kubernetes授权策略，请参见配置RAM用户RBAC权限。

> **注意** 当前容器服务ACK授权管理只支持自定义ClusterRole角色与集群内RBAC权限的绑定，不支持自定义Role角色与集群内RBAC权限的绑定。

3.4. 配置RAM用户RBAC权限

RBAC（Role-Based Access Control）是基于角色的访问控制，通过RBAC可以将权限与集群角色相关联，从而为不同角色成员配置不同的权限策略。本文主要介绍如何配置RAM用户（即子账号）对应的Kubernetes集群内RBAC权限。

配置说明

- 默认情况下RAM用户（非集群创建者）没有集群内任何Kubernetes资源的访问权限。
- 请先确保目标RAM用户在RAM管理控制台中至少已被授予指定集群的RAM只读权限。
- 您可以使用容器服务提供的预置角色即管理员（拥有集群内全部Kubernetes资源访问权限）、运维人员、开发人员、受限用户、和自定义角色来对目标RAM用户进行授权。
- 对于拥有全局权限的管理员RAM用户，可以在所有集群维度进行一键授权，后续新创建的集群也会自动绑定已经授权的ClusterRole。
- 当运维管理RAM用户给其他RAM用户进行RBAC授权时，控制台会过滤其可以授权的集群和命名空间的资源范围，只有当运维管理RAM用户有指定集群或命名空间的管理员或cluster-admin角色时，才可以给其他RAM用户进行RBAC授权。
- 支持对多个目标RAM用户进行批量授权。
- 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RAM用户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RAM用户的手动授权。

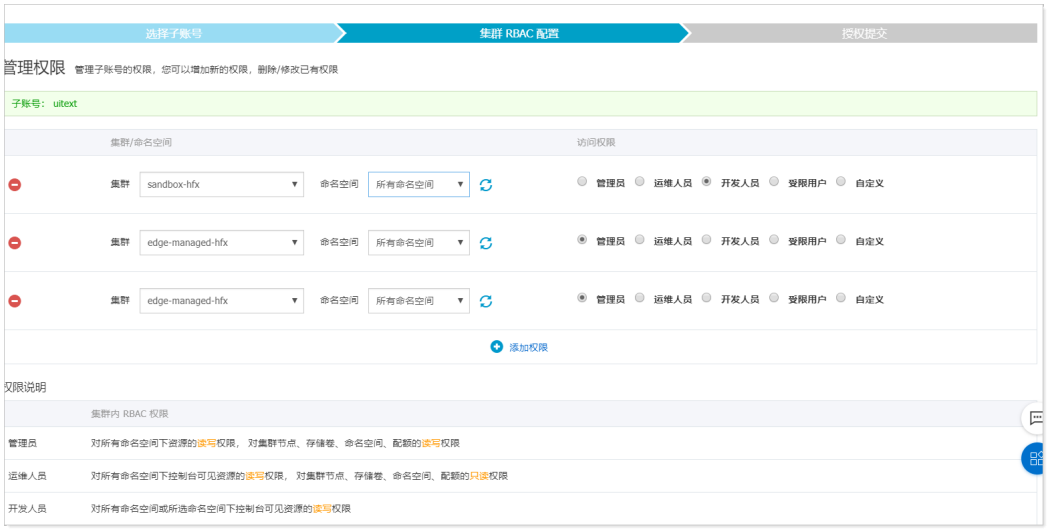
操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**授权管理**。
3. 在**授权管理**页面中的**选择子账号**页签列表中选择需要授权的RAM用户，单击**管理权限**，进入**集群RBAC配置**页签。

   > **说明** 如果您使用RAM用户授权，请先确保该RAM用户已完成集群RBAC配置自定义RAM授权策略文档中的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RAM用户的授权，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。



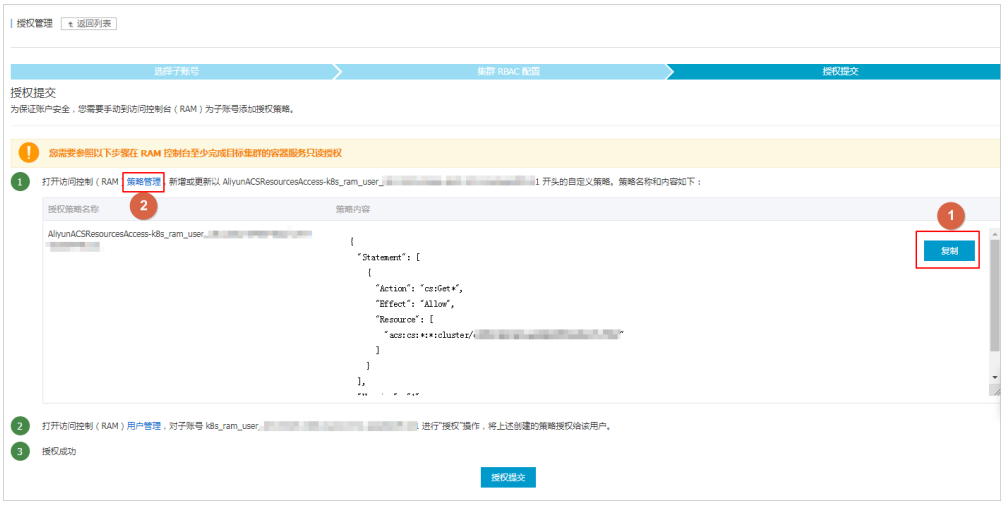
集群和命名空间的预置角色定义可查看下面的角色权限说明：
角色权限说明

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Kubernetes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Kubernetes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Kubernetes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RAM用户获得不符合预期的权限。更多信息，请参见自定义Kubernetes授权策略。

5. 在**授权提交**页签，如果出现**授权成功**，表示该RAM用户之前已被授予RAM权限，此时也已完成RBAC授权，操作结束。如果出现如授权管理所示，表示该RAM用户未被授予RAM权限，请按照页面提示，通过RAM控制台对RAM用户授予指定集群的只读权限。

i. 在**授权提交**页面，单击**复制**后，再单击**策略管理**。

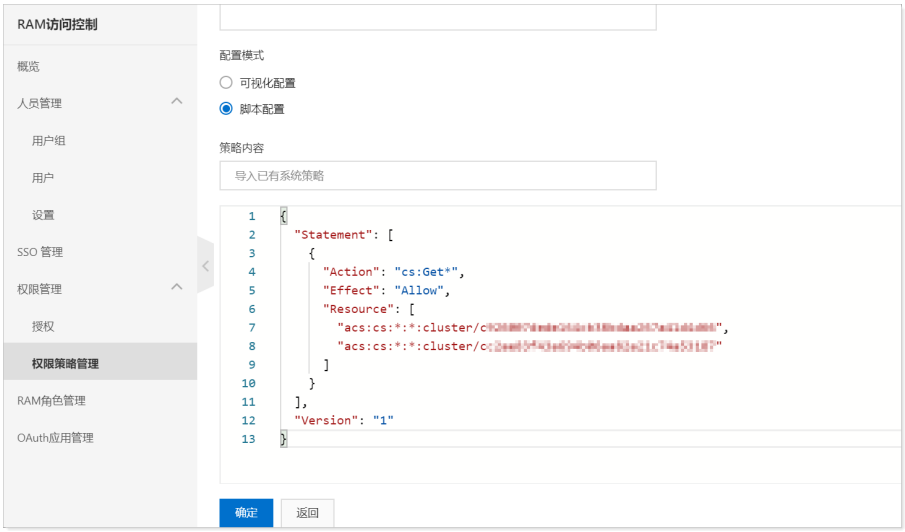
授权管理



ii. 跳转到RAM控制台页面，选择**权限管理 > 权限策略管理**，单击**创建权限策略**。



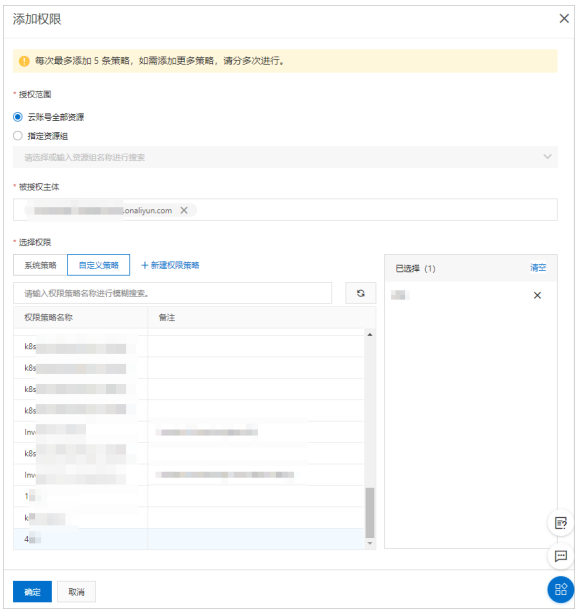
iii. 进入**新建自定义权限策略**页面，自定义**策略名称**，配置模式选择**脚本配置**，使用**Ctrl+V**，将5.i复制的内容粘贴到**策略内容**中，单击**确定**。关于如何新建自定义权限策略，请参见自定义RAM授权策略。



iv. 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**。



v. 在弹出的**添加权限**页面，设置**授权范围**，设置权限为**自定义策略**，搜索或者在下表中找到自定义的策略名称，单击策略名称，移动到右侧**已选择**区域。单击**确定**，完成对RAM用户在指定集群的只读能力的授权。



vi. 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RAM用户RBAC的授权。

6. 配置完成后，您可以使用目标RAM用户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、开发人员和受限用户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。



您可登录到集群节点，执行以下命令，查看自定义权限的详情。

```
kubectl get clusterrole
```

预期输出：

```
NAME                                   AGE
admin                                  13d
alibaba-log-controller                 13d
alicloud-disk-controller-runner        13d
cluster-admin                          13d
cs:admin                               13d
edit                                   13d
flannel                                13d
kube-state-metrics                     22h
node-exporter                          22h
prometheus-k8s                         22h
prometheus-operator                    22h
system:aggregate-to-admin              13d
....
system:volume-scheduler                13d
view                                   13d
```

以超级管理员cluster-admin为例，执行以下命令，查看其权限详情。

> **说明** RAM用户被授予该集群角色后，在该集群内，可视为与阿里云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
```

预期输出：

```
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:
 - '*'
```

3.5. 容器服务默认角色

在您开通容器服务时，需要为服务账号授予系统默认角色，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。本文介绍容器服务ACK默认角色包含的权限。

角色权限内容

本文介绍了以下角色的权限内容：

- AliyunCSManagedLogRole
- AliyunCSManagedCmsRole
- AliyunCSManagedCsiRole
- AliyunCSManagedVKRole
- AliyunCSClusterRole
- AliyunCSServerlessKubernetesRole
- AliyunCSKubernetesAuditRole
- AliyunCSManagedNetworkRole
- AliyunCSDefaultRole
- AliyunCSManagedKubernetesRole
- AliyunCSManagedArmsRole

AliyunCSManagedLogRole

ACK集群日志组件使用AliyunCSManagedLogRole角色来访问您在其他云产品中的资源。

权限名称（Action）	说明
log:CreateProject	创建一个Project。

权限名称（Action）	说明
log:GetProject	根据Project名称查询Project。
log:DeleteProject	删除一个指定的Project。
log:CreateLogStore	在Project下创建Logstore。
log:GetLogStore	查看Logstore属性。
log:UpdateLogStore	更新Logstore的属性。
log:DeleteLogStore	删除Logstore。
log:CreateConfig	创建日志采集配置。
log:UpdateConfig	更新配置内容。
log:GetConfig	获取采集配置的详细信息。
log:DeleteConfig	删除指定的日志采集配置。
log:CreateMachineGroup	根据需求创建一组机器，用以日志收集下发配置。
log:UpdateMachineGroup	更新机器组信息。
log:GetMachineGroup	查看具体的MachineGroup信息。
log:DeleteMachineGroup	删除机器组。
log:ApplyConfigToGroup	将配置应用到机器组。
log:GetAppliedMachineGroups	获得机器组上已经被应用的机器列表。
log:GetAppliedConfigs	获得机器组上已经被应用的配置名称。
log:RemoveConfigFromMachineGroup	从机器组中删除配置。
log:CreateIndex	为指定Logstore创建索引。
log:GetIndex	查询指定Logstore的索引。
log:UpdateIndex	更新指定Logstore的索引。
log:DeleteIndex	删除指定Logstore的索引。
log:CreateSavedSearch	创建快速查询。
log:GetSavedSearch	查看指定快速查询。
log:UpdateSavedSearch	更新快速查询。
log:DeleteSavedSearch	删除快速查询。
log:CreateDashboard	创建仪表盘。
log:GetDashboard	查看指定仪表盘。

权限名称（Action）	说明
log:UpdateDashboard	更新仪表盘。
log:DeleteDashboard	删除仪表盘。
log:CreateJob	创建任务。例如创建告警、订阅。
log:GetJob	查询任务。
log:DeleteJob	删除任务。
log:UpdateJob	更新任务。
log:PostLogStoreLogs	向指定的Logstore写入日志数据。
log:CreateSortedSubStore	创建排序子存储。
log:GetSortedSubStore	获取排序子存储。
log:ListSortedSubStore	列举排序子存储。
log:UpdateSortedSubStore	更新排序子存储。
log:DeleteSortedSubStore	删除排序子存储。
log:CreateApp	日志服务APP（成本管家、日志审计）的创建权限。
log:UpdateApp	日志服务APP（成本管家、日志审计）的更新权限。
log:GetApp	日志服务APP（成本管家、日志审计）的查看权限。
log:DeleteApp	日志服务APP（成本管家、日志审计）的删除权限。
cs:DescribeTemplates	获取容器模板。
cs:DescribeTemplateAttribute	获取容器模板属性。

AliyunCSManagedCmsRole

ACK集群云监控组件使用AliyunCSManagedCmsRole角色来访问您在其他云产品中的资源。

权限名称（Action）	说明
cms:DescribeMonitorGroups	查询应用分组列表。
cms:DescribeMonitorGroupInstances	查询指定应用分组内包含的资源列表。
cms:CreateMonitorGroup	创建一个应用分组。
cms:DeleteMonitorGroup	删除指定的应用分组。
cms:ModifyMonitorGroupInstances	修改应用分组中的资源。
cms:CreateMonitorGroupInstances	添加资源到应用分组。
cms:DeleteMonitorGroupInstances	删除应用分组内的资源实例。

权限名称（Action）	说明
cms:TaskConfigCreate	创建监控任务配置。
cms:TaskConfigList	列举监控任务配置。
cms:DescribeMetricList	查询指定时间段内的云产品时序指标监控数据。
cs:DescribeMonitorToken	获取容器服务监控令牌。
ahas:GetSentinelAppSumMetric	获取AHAS Sentinel应用监控指标。
log:GetLogStoreLogs	查看Logstore日志。
slb:DescribeMetricList	查询指定时间段内的云产品时序指标监控数据。
sls:GetLogs	获取SLS服务中指定Project下某个Logstore中的日志数据。
sls:PutLogs	更新SLS服务中指定Project下某个Logstore中的日志数据。

AliyunCSManagedCsiRole

ACK集群存储插件使用AliyunCSManagedCsiRole角色来访问您在其他云产品中的资源。

- ECS相关权限

权限名称（Action）	说明
ecs:AttachDisk	为一台ECS实例挂载一块按量付费数据盘，或者挂载一块系统盘。
ecs:DetachDisk	从一台实例上卸载一块按量付费磁盘。
ecs:DescribeDisks	查询一块或多块您已经创建的云盘以及本地盘。
ecs:CreateDisk	创建一块按量付费或包年包月数据盘。
ecs:ResizeDisk	扩容一块云盘，支持扩容系统盘和数据盘。
ecs:CreateSnapshot	为一块云盘创建一份快照。
ecs:DeleteSnapshot	删除指定的快照。如果需要取消正在创建的快照，也可以调用该接口删除快照，即取消创建快照任务。
ecs:CreateAutoSnapshotPolicy	创建一条自动快照策略。
ecs:ApplyAutoSnapshotPolicy	为一块或者多块云盘应用自动快照策略。
ecs:CancelAutoSnapshotPolicy	取消一块或者多块云盘的自动快照策略。
ecs:DeleteAutoSnapshotPolicy	删除一条自动快照策略。
ecs:DescribeAutoSnapshotPolicyEX	查询您已创建的自动快照策略。
ecs:ModifyAutoSnapshotPolicyEx	修改一条自动快照策略。
ecs:AddTags	为ECS节点添加标签。

权限名称（Action）	说明
ecs:DescribeTags	查询标签。
ecs:DescribeSnapshots	查询一台ECS实例或一块云盘所有的快照列表。
ecs:ListTagResources	查询一个或多个ECS资源已经绑定的标签列表。
ecs:TagResources	为指定的ECS资源列表统一创建并绑定标签。
ecs:UntagResources	为指定的ECS资源列表统一解绑并删除标签。
ecs:ModifyDiskSpec	升级一块ESSD云盘的性能等级。
ecs:CreateSnapshot	为一块云盘创建一份快照。
ecs:DeleteDisk	释放一块按量付费数据盘。
ecs:DescribeInstanceAttribute	查询实例所有的属性信息。
ecs:DescribeInstances	查询一台或多台ECS实例的详细信息。

- NAS相关权限

权限名称（Action）	说明
nas:DescribeFileSystems	返回文件系统描述信息。
nas:DescribeMountTargets	返回挂载点描述信息。
nas:AddTags	添加或者覆盖一个或者多个标签到一个文件系统实例。
nas:DescribeTags	查询已有标签。
nas:RemoveTags	从一个文件系统实例上解绑一个或多个标签。
nas:CreateFileSystem	创建新的文件系统。
nas:DeleteFileSystem	删除已有文件系统。
nas:DescribeFileSystems	返回文件系统描述信息。
nas:ModifyFileSystem	修改文件系统信息。
nas:CreateMountTarget	创建挂载点。
nas:DeleteMountTarget	删除已有挂载点。
nas:DescribeMountTargets	返回挂载点描述信息。
nas:ModifyMountTarget	修改挂载点信息。

AliyunCSManagedVKRole

ACK集群Virtual Kubelet组件使用AliyunCSManagedVKRole角色来访问您在其他云产品中的资源。

- VPC相关权限

权限名称（Action）	说明
vpc:DescribeVSwitches	查询已创建的交换机。
vpc:DescribeVpcs	查询已创建的VPC。
vpc:AssociateEipAddress	将弹性公网IP（EIP）绑定到同地域的云产品实例上。
vpc:DescribeEipAddresses	查询指定地域已创建的EIP。
vpc:AllocateEipAddress	申请弹性公网IP（EIP）。
vpc:ReleaseEipAddress	释放指定的弹性公网IP（EIP）。

- ECS相关权限

权限名称（Action）	说明
ecs:DescribeSecurityGroups	查询您创建的安全组的基本信息。
ecs:CreateNetworkInterface	创建一个弹性网卡（ENI）。
ecs:CreateNetworkInterfacePermission	创建弹性网卡（ENI）权限。
ecs:DescribeNetworkInterfaces	查看弹性网卡（ENI）列表。
ecs:AttachNetworkInterface	附加一个弹性网卡（ENI）到一台专有网络VPC类型ECS实例上。
ecs:DetachNetworkInterface	从一台ECS实例上分离一个弹性网卡（ENI）。
ecs:DeleteNetworkInterface	删除一个弹性网卡（ENI）。
ecs:DeleteNetworkInterfacePermission	删除弹性网卡（ENI）权限。

- VPTZ相关权限

权限名称（Action）	说明
pvtz:AddZone	创建私有区域。
pvtz:DeleteZone	删除私有区域。
pvtz:DescribeZones	查询用户的区域列表。
pvtz:DescribeZoneInfo	获取指定区域的详细信息。
pvtz:BindZoneVpc	绑定或者解绑区域与VPC列表两者之间的关系。
pvtz:AddZoneRecord	添加私有区域的解析记录。
pvtz:DeleteZoneRecord	删除解析记录。
pvtz:DeleteZoneRecordsByRR	删除解析记录列表。
pvtz:DescribeZoneRecordsByRR	查询解析记录列表

权限名称（Action）	说明
pvtz:DescribeZoneRecords	查询解析记录列表。

- ECI相关权限

权限名称（Action）	说明
eci:CreateContainerGroup	创建一个容器组。
eci:DeleteContainerGroup	删除一个容器组。
eci:DescribeContainerGroups	批量获取容器组信息。
eci:DescribeContainerLog	获取容器组日志信息。
eci:UpdateContainerGroup	更新实例。
eci:UpdateContainerGroupByTemplate	通过模板更新ECI实例。
eci:CreateContainerGroupFromTemplate	通过模板创建ECI实例。
eci:RestartContainerGroup	重启ECI实例。
eci:ExportContainerGroupTemplate	导出用户创建ECI的模板。
eci:DescribeContainerGroupMetric	查询一个ECI的监控信息。
eci:DescribeMultiContainerGroupMetric	同时查询多个容器组的监控信息。
eci:ExecContainerCommand	在容器内部执行命令。
eci:CreateImageCache	制作镜像缓存。
eci:DescribeImageCaches	查询镜像缓存信息。
eci:DeleteImageCache	删除镜像缓存。

AliyunCSClusterRole

ACK在应用运行期使用AliyunCSClusterRole角色来访问您在其他云产品中的资源。

- ECS相关权限

权限名称（Action）	说明
ecs:Describe*	查询ECS相关资源。

- OSS相关权限

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

- CMS相关权限

权限名称（Action）	说明
cms:List*	列举CMS相关资源权限。
cms:Get*	获取CMS相关资源权限。
cms:UpdateAlert	更新报警。
cms:CreateAlert	创建报警。
cms:DeleteAlert	删除报警。
cms:UpdateDimensions	更新监控规模配置。
cms:CreateDimensions	创建监控规模配置。
cms:DeleteDimensions	删除监控规模配置。
cms:SendAlarm	发送监控告警。
cms:CreateProject	创建监控项目。
cms:DeleteProject	删除监控项目。
cms:UpdateProject	更新监控项目。
cms:QueryAlarm	查询监控告警。
cms:ListAlarm	列举监控告警。
cms:CreateAlarm	创建监控告警
cms:DeleteAlarm	删除监控告警
cms:UpdateAlarm	更新监控告警

- SLB权限

权限名称（Action）	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。

权限名称（Action）	说明
slb:DeleteLoadBalancerListener	删除负载均衡实例。
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协议的监听。
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。

- 日志服务相关权限

权限名称（Action）	说明
log:Get*	获取日志服务相关资源权限。
log:List*	列举日志服务相关资源权限。
log:CreateProject	创建一个Project。
log:DeleteProject	删除一个指定的Project。
log:UpdateProject	更新一个Project。
log:CreateMachineGroup	根据需求创建一组机器，用以日志收集下发配置。
log:DeleteMachineGroup	删除机器组。
log:UpdateMachineGroup	更新机器组信息。
log:CreateLogStore	在Project下创建Logstore。
log:DeleteLogStore	删除Logstore。
log:UpdateLogStore	更新Logstore的属性。
log:CreateIndex	为指定Logstore创建索引。
log:DeleteIndex	删除指定Logstore的索引。
log:UpdateIndex	更新指定Logstore的索引。
log:CreateConfig	创建日志采集配置。
log:DeleteConfig	删除指定的日志采集配置。
log:UpdateConfig	更新配置内容。
log:ApplyConfigToGroup	将配置应用到机器组。

AliyunCSServerlessKubernetesRole

ASK默认使用AliyunCSServerlessKubernetesRole角色来访问您在其他云产品中的资源。

- VPC相关权限

权限名称（Action）	说明
DescribeVSwitches	查询已创建的交换机。
DescribeVpcs	查询已创建的VPC。
AssociateEipAddress	将弹性公网IP（EIP）绑定到同地域的云产品实例上。
DescribeEipAddresses	查询指定地域已创建的EIP。
AllocateEipAddress	申请弹性公网IP（EIP）。
ReleaseEipAddress	释放指定的弹性公网IP（EIP）。
AddCommonBandwidthPackageIp	添加EIP到共享带宽中。
RemoveCommonBandwidthPackageIp	移除共享带宽实例中的EIP。

- ECS相关权限

权限名称（Action）	说明
DescribeSecurityGroups	查询您创建的安全组的基本信息。
CreateNetworkInterface	创建一个弹性网卡（ENI）。
CreateNetworkInterfacePermission	创建弹性网卡（ENI）权限。
DescribeNetworkInterfaces	查看弹性网卡（ENI）列表。
AttachNetworkInterface	附加一个弹性网卡（ENI）到一台专有网络VPC类型ECS实例上。
DetachNetworkInterface	从一台ECS实例上分离一个弹性网卡（ENI）。
DeleteNetworkInterface	删除一个弹性网卡（ENI）。
DeleteNetworkInterfacePermission	删除弹性网卡（ENI）权限。

- SLB相关权限

权限名称（Action）	说明
slb:Describe*	查询负载均衡SLB相关资源。
slb:CreateLoadBalancer	创建负载均衡实例。
slb:DeleteLoadBalancer	删除后付费的负载均衡实例。
slb:RemoveBackendServers	移除后端服务器。
slb:StartLoadBalancerListener	启动监听。
slb:StopLoadBalancerListener	停止监听。

权限名称（Action）	说明
slb:DeleteLoadBalancerListener	删除负载均衡实例监听规则。
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:UploadServerCertificate	上传服务器证书。
slb:CreateLoadBalancerHTTPListener	为负载均衡实例创建基于HTTP协议的监听。
slb:CreateLoadBalancerHTTPSListener	为负载均衡实例创建基于HTTPS协议的监听。
slb:CreateLoadBalancerUDPListener	创建UDP监听。
slb:ModifyLoadBalancerInternetSpec	修改公网负载均衡实例的计费方式。
slb:CreateRules	为指定的HTTP或HTTPS监听添加转发规则。
slb:DeleteRules	删除转发规则。
slb:SetRule	修改目标虚拟服务器组的转发规则。
slb:CreateVServerGroup	向指定的后端服务器组中添加后端服务器。
slb:SetVServerGroupAttribute	修改虚拟服务器组的配置。
slb:AddVServerGroupBackendServers	向指定的后端服务器组中添加后端服务器。
slb:RemoveVServerGroupBackendServers	从指定的后端服务器组中移除后端服务器。
slb:ModifyVServerGroupBackendServers	替换服务器组中的后端服务器。
slb:DeleteVServerGroup	删除服务器组。
slb:SetLoadBalancerTCPListenerAttribute	修改TCP监听的配置。
slb:SetLoadBalancerHTTPListenerAttribute	修改HTTP监听的配置。
slb:SetLoadBalancerHTTPSListenerAttribute	修改HTTPS监听的配置。
slb:AddTags	为指定的负载均衡实例添加标签。

- PVTZ（云解析）相关权限

权限名称（Action）	说明
AddZone	创建私有区域。
DeleteZone	删除私有区域。
DescribeZones	查询用户的区域列表。
DescribeZoneInfo	获取指定区域的详细信息。
BindZoneVpc	绑定或者解绑区域与VPC列表两者之间的关系。

权限名称（Action）	说明
AddZoneRecord	添加私有区域的解析记录。
DeleteZoneRecord	删除解析记录。
DeleteZoneRecordsByRR	删除解析记录列表。
DescribeZoneRecordsByRR	查询解析记录列表
DescribeZoneRecords	查询解析记录列表。

- ACR相关权限

权限名称（Action）	说明
Get*	查询容器镜像服务相关资源。
List*	查看容器镜像仓库列表。
PullRepository	拉取镜像。

- ECI相关权限

权限名称（Action）	说明
CreateContainerGroup	创建一个容器组。
DeleteContainerGroup	删除一个容器组。
DescribeContainerGroups	批量获取容器组信息。
DescribeContainerLog	获取容器组日志信息。
UpdateContainerGroup	更新实例。
UpdateContainerGroupByTemplate	通过模板更新ECI实例。
CreateContainerGroupFromTemplate	通过模板创建ECI实例。
RestartContainerGroup	重启ECI实例。
ExportContainerGroupTemplate	导出用户创建ECI的模板。
DescribeContainerGroupMetric	查询一个ECI的监控信息。
DescribeMultiContainerGroupMetric	同时查询多个容器组的监控信息。
ExecContainerCommand	在容器内部执行命令。
CreateImageCache	制作镜像缓存。
DescribeImageCaches	查询镜像缓存信息。
DeleteImageCache	删除镜像缓存。

- RAM相关权限

权限名称（Action）	说明
ram:PassRole	访问CodePipeline控制台。

- OSS相关权限

权限名称（Action）	说明
oss:GetObject	获取文件或文件夹对象。
oss:GetObjectMeta	获取一个文件（Object）的元数据信息。

- FC相关权限

权限名称（Action）	说明
fc:CreateService	新建一个服务。
fc:ListServices	获取服务列表。
fc:GetService	获取指定服务。
fc:UpdateService	更新指定服务。
fc:DeleteService	删除指定服务。
fc:CreateFunction	新建一个新函数。
fc:ListFunctions	获取服务下的函数列表。
fc:GetFunction	获取指定函数的配置信息。
fc:GetFunctionCode	获取函数编码。
fc:UpdateFunction	更新函数，包括配置和代码。
fc:DeleteFunction	删除指定的函数。
fc:CreateTrigger	创建函数触发器。
fc:ListTriggers	获取函数下的触发器列表。
fc:GetTrigger	获取指定触发器。
fc:UpdateTrigger	更新指定函数触发器配置。
fc:DeleteTrigger	删除指定函数的触发器。
fc:PublishServiceVersion	发布函数计算服务版本。
fc:ListServiceVersions	列举函数计算服务版本。
fc:DeleteServiceVersion	删除函数计算服务版本。
fc:CreateAlias	创建一个别名，并且将别名与一个用户主密钥绑定。
fc:ListAliases	列出云账号在本地域的所有别名。

权限名称（Action）	说明
fc:GetAlias	获取别名信息。
fc:UpdateAlias	绑定指定别名到新的用户主密钥。
fc:DeleteAlias	删除指定别名。

AliyunCSKubernetesAuditRole

ACK审计功能使用AliyunCSKubernetesAuditRole角色来访问您在其他云产品中的资源。

权限名称（Action）	说明
log:CreateProject	创建一个Project。
log:GetProject	根据Project名称查询Project。
log:DeleteProject	删除一个指定的Project。
log:CreateLogStore	在Project下创建Logstore。
log:GetLogStore	查看Logstore属性。
log:UpdateLogStore	更新Logstore的属性。
log:DeleteLogStore	删除Logstore。
log:CreateConfig	创建日志采集配置。
log:UpdateConfig	更新配置内容。
log:GetConfig	获取采集配置的详细信息。
log:DeleteConfig	删除指定的日志采集配置。
log:CreateMachineGroup	根据需求创建一组机器，用以日志收集下发配置。
log:UpdateMachineGroup	更新机器组信息。
log:GetMachineGroup	查看具体的MachineGroup信息。
log:DeleteMachineGroup	删除机器组。
log:ApplyConfigToGroup	将配置应用到机器组。
log:GetAppliedMachineGroups	获得机器组上已经被应用的机器列表。
log:GetAppliedConfigs	获得机器组上已经被应用的配置名称。
log:RemoveConfigFromMachineGroup	从机器组中删除配置。
log:CreateIndex	为指定Logstore创建索引。
log:GetIndex	查询指定Logstore的索引。
log:UpdateIndex	更新指定Logstore的索引。

权限名称（Action）	说明
log:DeleteIndex	删除指定Logstore的索引。
log:CreateSavedSearch	创建快速查询。
log:GetSavedSearch	查看指定快速查询。
log:UpdateSavedSearch	更新快速查询。
log:DeleteSavedSearch	删除快速查询。
log:CreateDashboard	创建仪表盘。
log:GetDashboard	查看指定仪表盘。
log:UpdateDashboard	更新仪表盘。
log:DeleteDashboard	删除仪表盘。
log:CreateJob	创建任务。例如创建告警、订阅。
log:GetJob	查询任务。
log:DeleteJob	删除任务。
log:UpdateJob	更新任务。
log:PostLogStoreLogs	向指定的Logstore写入日志数据。

AliyunCSManagedNetworkRole

ACK集群网络组件使用AliyunCSManagedNetworkRole角色来访问您在其他云产品中的资源。

权限名称（Action）	说明
ecs:CreateNetworkInterface	创建一个弹性网卡（ENI）。
ecs:DescribeNetworkInterfaces	查看弹性网卡（ENI）列表。
ecs:AttachNetworkInterface	附加弹性网卡（ENI）到专有网络（VPC）类型实例上。
ecs:DetachNetworkInterface	从一台实例上分离一个弹性网卡（ENI）。
ecs:DeleteNetworkInterface	删除一个弹性网卡（ENI）。
ecs:DescribeInstanceAttribute	查询一台或多台ECS实例部分信息。
ecs:AssignPrivateIpAddresses	为一块弹性网卡分配一个或多个辅助私有IP地址。
ecs:UnassignPrivateIpAddresses	从一块弹性网卡删除一个或多个辅助私有IP地址。
ecs:DescribeInstances	查询一台或多台ECS实例的详细信息。
vpc:DescribeVSwitches	查询一台或多台交换机的详细信息。

AliyunCSDefaultRole

ACK在集群操作时默认使用AliyunCSDefaultRole角色来访问您在其他云产品中的资源。

- ECS相关权限

权限名称（Action）	说明
ecs:RunInstances	启动ECS实例。
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:WaitFor*	任务执行等待。
ecs:AddTags	添加标签。
ecs:ReplaceSystemDisk	更换ECS实例的系统盘。
ecs:ModifyInstanceAttribute	修改实例属性。
ecs:JoinSecurityGroup	将实例加入到指定的安全组。
ecs:LeaveSecurityGroup	将实例移出指定的安全组。
ecs:UnassociateEipAddress	解绑弹性公网IP。
ecs:ReleaseEipAddress	释放弹性公网IP。
ecs:CreateKeyPair	创建一对SSH密钥对。
ecs:ImportKeyPair	导入由其他工具产生的RSA密钥对的公钥部分。
ecs:AttachKeyPair	绑定一个SSH密钥对到一台或多台Linux实例。

权限名称（Action）	说明
ecs:DetachKeyPair	为一台或者多台Linux实例解绑SSH密钥对。
ecs:DeleteKeyPairs	删除一对或者多对SSH密钥对。
ecs:AttachInstanceRamRole	为一台或多台ECS实例授予实例RAM角色。
ecs:DetachInstanceRamRole	收回一台或多台ECS实例的实例RAM角色。
ecs:AllocateDedicatedHosts	创建一台或多台按量付费或者包年包月专有宿主机。
ecs:CreateOrder	创建ECS实例订单。
ecs:DeleteInstance	释放一台按量付费实例或者到期的包年包月实例。
ecs:CreateDisk	创建一块按量付费或包年包月数据盘。
ecs:Createvpc	创建ECS关联VPC。
ecs:Deletevpc	删除ECS关联VPC。
ecs:DeleteVSwitch	删除ECS关联交换机。
ecs:ResetDisk	使用磁盘的历史快照回滚至某一阶段的磁盘状态。
ecs:DeleteSnapshot	删除指定的快照。
ecs:AllocatePublicIpAddress	为一台ECS实例分配一个公网IP地址。
ecs:CreateVSwitch	创建ECS关联交换机。
ecs:DeleteSecurityGroup	删除一个安全组。
ecs:CreateImage	创建一份自定义镜像。
ecs:RemoveTags	删除ECS实例标签。
ecs:ReleaseDedicatedHost	释放一台按量付费专有宿主机。
ecs:CreateInstance	创建一台包年包月或者按量付费ECS实例。
ecs:RevokeSecurityGroupEgress	删除一条安全组出方向规则，撤销安全组出方向的访问权限。
ecs:DeleteDisk	释放一块按量付费数据盘。
ecs:StopInstance	停止运行一台实例。
ecs:CreateSecurityGroup	新建一个安全组。
ecs:RevokeSecurityGroup	删除一条安全组入方向规则，撤销安全组入方向的权限设置。
ecs:DeleteImage	删除一份自定义镜像。
ecs:ModifyInstanceSpec	调整一台按量付费ECS实例的实例规格和公网带宽大小。

权限名称（Action）	说明
ecs:CreateSnapshot	为一块云盘创建一份快照。
ecs:CreateCommand	新建一条云助手命令。
ecs:InvokeCommand	为一台或多台ECS实例触发一条云助手命令。
ecs:StopInvocation	停止一台或多台ECS实例中一条正在进行中（Running）的云助手命令进程。
ecs:DeleteCommand	删除一条云助手命令。
ecs:RunCommand	新建一份Shell、PowerShell或者Bat类型的云助手脚本，然后在一台或多台ECS实例中执行该脚本。
ecs:DescribeInvocationResults	查看云助手命令的执行结果，在指定ECS实例中的实际执行结果。
ecs:ModifyCommand	修改一条云助手命令。

- VPC相关权限

权限名称（Action）	说明
vpc:Describe*	查询VPC相关资源的信息。
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。
vpc:CreateVSwitch	创建交换机。
vpc:DeleteVSwitch	删除交换机。
vpc:CreateVpc	创建一个VPC。
vpc:DeleteVpc	删除一个VPC。
vpc:CreateNatGateway	创建一个NAT网关。
vpc:DeleteNatGateway	删除指定的NAT网关。
vpc:CreateSnatEntry	在SNAT列表中添加SNAT条目。
vpc:DeleteSnatEntry	删除指定的SNAT条目。
vpc:ModifyEipAddressAttribute	修改指定EIP的名称、描述信息和带宽峰值。

权限名称（Action）	说明
vpc:CreateForwardEntry	在DNAT列表中添加DNAT条目。
vpc:DeleteBandwidthPackage	创建指定的NAT带宽包。
vpc:CreateBandwidthPackage	删除指定的NAT带宽包。
vpc:DeleteForwardEntry	删除指定的DNAT条目。
vpc:TagResources	为指定的资源统一创建并绑定标签。
vpc:DeletionProtection	置实例删除保护功能。

- SLB相关权限

权限名称（Action）	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:CreateVServerGroup	创建虚拟服务器组，并添加后端服务器。
slb:CreateLoadBalancerHTTPSListener	为负载均衡实例创建基于HTTPS协议的监听。
slb:CreateLoadBalancerUDPListener	创建UDP监听。
slb:ModifyLoadBalancerInternetSpec	修改公网负载均衡实例的计费方式。
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:AddVServerGroupBackendServers	向指定的后端服务器组中添加后端服务器。
slb:DeleteVServerGroup	删除服务器组。
slb:ModifyVServerGroupBackendServers	替换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协议的监听。
slb:RemoveVServerGroupBackendServers	从指定的后端服务器组中移除后端服务器。
slb:DeleteLoadBalancerListener	删除负载均衡实例监听规则。

权限名称（Action）	说明
slb:AddTags	为指定的负载均衡实例添加标签。
slb:RemoveTags	解绑指定负载均衡实例下的标签。
slb:SetLoadBalancerDeleteProtection	为负载均衡实例设置删除保护。

- DNS相关权限

权限名称（Action）	说明
dns:Describe*	查询DNS相关资源。
dns:AddDomainRecord	添加域名解析记录。

- RDS相关权限

权限名称（Action）	说明
rds:Describe*	查询RDS相关资源。
rds:ModifySecurityIps	修改RDS实例IP白名单。

- ROS相关权限

权限名称（Action）	说明
ros:Describe*	获取ROS相关资源。
ros:WaitConditions	ROS脚本执行等待。
ros:AbandonStack	终止Stack。
ros:DeleteStack	删除Stack。
ros:CreateStack	创建Stack。
ros:UpdateStack	更新Stack。
ros:ValidateTemplate	校验ROS模板。
ros:DoActions	执行Actions。
ros:InquiryStack	查询Stack。
ros:SetDeletionProtection	设置删除保护。
ros:PreviewStack	预览Stack。

- ESS相关权限

权限名称（Action）	说明
ess:Describe*	查询ESS相关资源。

权限名称（Action）	说明
ess:CreateScalingConfiguration	创建一个伸缩配置。
ess:EnableScalingGroup	启用一个伸缩组。
ess:ExitStandby	使伸缩组内处于备用状态的ECS实例进入运行状态。
ess:DetachDBInstances	移除一个或多个RDS实例。
ess:DetachLoadBalancers	移除一个或多个负载均衡实例。
ess:AttachInstances	从一个伸缩组关联一台或多台ECS实例。
ess:DeleteScalingConfiguration	删除一个伸缩配置。
ess:AttachLoadBalancers	添加一个或多个负载均衡实例。
ess:DetachInstances	从一个伸缩组分离一台或多台ECS实例。
ess:ModifyScalingRule	修改伸缩组规则。
ess:RemoveInstances	从指定的伸缩组里移出ECS实例。
ess:ModifyScalingGroup	修改一个伸缩组。
ess:AttachDBInstances	添加一个或多个RDS实例。
ess:CreateScalingRule	创建一条伸缩规则。
ess:DeleteScalingRule	删除一条伸缩规则。
ess:ExecuteScalingRule	执行一条伸缩规则。
ess:SetInstancesProtection	保护或者停止保护伸缩组内的一台或者多台ECS实例。
ess:ModifyNotificationConfiguration	修改一条弹性伸缩事件及资源变化通知的信息。
ess:CreateNotificationConfiguration	创建一条弹性伸缩事件及资源变化通知的信息。
ess:EnterStandby	将伸缩组内的ECS实例设置为备用状态。
ess:DeleteScalingGroup	删除一个伸缩组。
ess:CreateScalingGroup	创建一个伸缩组。
ess:DeleteNotificationConfiguration	删除一条弹性伸缩事件及资源变化通知的信息。
ess:DisableScalingGroup	禁用一个伸缩组。
ModifyScalingConfiguration	修改一个伸缩配置。
SetGroupDeletionProtection	为伸缩组开启或关闭删除保护。

- RAM相关权限

权限名称（Action）	说明
ram:PassRole	RAM跨服务授权权限。
ram:Get*	获取RAM相关资源权限。
ram:List*	列举RAM相关资源权限。
ram:DetachPolicyFromRole	为角色撤销指定的权限。
ram:AttachPolicyToRole	为指定角色附加授权。
ram:DeletePolicy	删除指定权限策略。
ram:DeletePolicyVersion	删除指定版本的策略。
ram:DeleteRole	删除RAM角色。
ram:CreateRole	创建RAM角色。
ram:CreatePolicy	创建RAM策略。
ram:CreateServiceLinkedRole	创建SLR权限。

- CMS相关权限

权限名称（Action）	说明
cms:CreateMyGroups	创建个人应用分组。
cms:AddMyGroupInstances	添加个人分组内的资源实例。
cms:DeleteMyGroupInstances	删除个人分组内的资源实例。
cms:DeleteMyGroups	删除个人应用分组。
cms:GetMyGroups	获取个人应用分组。
cms:ListMyGroups	列举个人应用分组。
cms:UpdateMyGroupInstances	更新个人分组中的资源实例。
cms:UpdateMyGroups	更新个人应用分组。
cms:TaskConfigCreate	创建监控任务配置。
cms:TaskConfigList	列举监控任务配置。

- ESS相关权限

权限名称（Action）	说明
ess:CreateLifecycleHook	为伸缩组创建一个或多个生命周期挂钩。
ess:DescribeLifecycleHooks	查询生命周期挂钩。
ess:ModifyLifecycleHook	修改生命周期挂钩。

权限名称（Action）	说明
ess:DeleteLifecycleHook	删除生命周期挂钩。

- ENS相关权限

权限名称（Action）	说明
ens:Describe*	获取边缘节点服务相关资源权限。
ens:CreateInstance	创建边缘实例。
ens:StartInstance	启动边缘实例。
ens:StopInstance	停止边缘实例。
ens:ReleasePrePaidInstance	释放包年包月实例。

AliyunCSManagedKubernetesRole

ACK托管版集群默认使用AliyunCSManagedKubernetesRole角色来访问您在其他云产品中的资源。

- ECS相关权限

权限名称（Action）	说明
ecs:Describe*	查询ECS相关资源。
ecs:CreateRouteEntry	创建路由接口。
ecs:DeleteRouteEntry	删除路由接口。
ecs:CreateNetworkInterface	创建一个弹性网卡（ENI）。
ecs:DeleteNetworkInterface	删除一个弹性网卡（ENI）。
ecs:CreateNetworkInterfacePermission	创建弹性网卡（ENI）权限。
ecs:DeleteNetworkInterfacePermission	删除弹性网卡（ENI）权限。
ecs:ModifyInstanceAttribute	修改实例属性。
ecs:AttachKeyPair	绑定一个SSH密钥对到一台或多台Linux实例。
ecs:StopInstance	停止运行一台实例。
ecs:StartInstance	启动一台实例。
ecs:ReplaceSystemDisk	更换一台ECS实例的系统盘或者操作系统。

- SLB相关权限

权限名称（Action）	说明
slb:Describe*	查询负载均衡相关资源。
slb:CreateLoadBalancer	创建负载均衡实例。

权限名称（Action）	说明
slb:DeleteLoadBalancer	删除负载均衡实例。
slb:ModifyLoadBalancerInternetSpec	修改公网负载均衡实例的计费方式。
slb:RemoveBackendServers	移除后端服务器。
slb:AddBackendServers	添加后端服务器。
slb:RemoveTags	解绑指定负载均衡实例下的标签。
slb:AddTags	为指定的负载均衡实例添加标签。
slb:StopLoadBalancerListener	停止监听。
slb:StartLoadBalancerListener	启动监听。
slb:SetLoadBalancerHTTPListenerAttribute	修改HTTP监听的配置。
slb:SetLoadBalancerHTTPSListenerAttribute	修改HTTPS监听的配置。
slb:SetLoadBalancerTCPListenerAttribute	修改TCP监听的配置。
slb:SetLoadBalancerUDPListenerAttribute	修改UDP协议监听的配置。
slb:CreateLoadBalancerHTTPSListener	为负载均衡实例创建基于HTTPS协议的监听。
slb:CreateLoadBalancerHTTPListener	为负载均衡实例创建基于HTTP协议的监听。
slb:CreateLoadBalancerTCPListener	为负载均衡实例创建基于TCP协议的监听规则。
slb:CreateLoadBalancerUDPListener	创建UDP监听。
slb:DeleteLoadBalancerListener	删除负载均衡实例监听规则。
slb:CreateVServerGroup	向指定的后端服务器组中添加后端服务器。
slb:DescribeVServerGroups	查询服务器组列表。
slb:DeleteVServerGroup	删除服务器组。
slb:SetVServerGroupAttribute	修改虚拟服务器组的配置。
slb:DescribeVServerGroupAttribute	查询服务器组的详细信息。
slb:ModifyVServerGroupBackendServers	替换服务器组中的后端服务器。
slb:AddVServerGroupBackendServers	向指定的后端服务器组中添加后端服务器。
slb:ModifyLoadBalancerInstanceSpec	修改负载均衡的实例规格。
slb:ModifyLoadBalancerInternetSpec	修改公网负载均衡实例的计费方式。
slb:RemoveVServerGroupBackendServers	从指定的后端服务器组中移除后端服务器。

- VPC相关权限

权限名称（Action）	说明
vpc:Describe*	查询VPC相关资源的信息。
vpc:DeleteRouteEntry	删除自定义路由条目。
vpc:CreateRouteEntry	创建自定义路由条目。

- ACR相关权限

权限名称（Action）	说明
cr:Get*	查询容器镜像服务相关资源。
cr:List*	查看容器镜像仓库列表。
cr:PullRepository	拉取镜像。

AliyunCSManagedArmsRole

ACK集群应用实时监控插件使用AliyunCSManagedArmsRole角色来访问您在其他云产品中的资源。

权限名称（Action）	说明
arms:CreateApp	创建应用监控。
arms:DeleteApp	删除应用监控。
arms:ConfigAgentLabel	修改应用监控Agent的标签。
arms:GetAssumeRoleCredentials	获取应用监控角色扮演密钥凭证。
arms:CreateProm	创建阿里云Prometheus监控。
arms:SearchEvents	查询报警事件记录。
arms:SearchAlarmHistories	查询告警历史发送记录。
arms:SearchAlertRules	查询监控告警规则。
arms:GetAlertRules	获取监控告警规则。
arms:CreateAlertRules	创建监控告警规则。
arms:UpdateAlertRules	更新监控告警规则。
arms:StartAlertRule	启动监控告警规则。
arms:StopAlertRule	停止监控告警规则。
arms:CreateContact	创建告警联系人。
arms:SearchContact	查询告警联系人。
arms:UpdateContact	更新告警联系人。
arms:CreateContactGroup	创建告警联系人分组。

权限名称（Action）	说明
arms:SearchContactGroup	查询告警联系人分组。
arms:UpdateContactGroup	更新告警联系人分组。

相关文档

- 首次使用容器服务Kubernetes版

3.6. 授权管理FAQ

本文介绍授权管理过程的一些常见问题的解决方法。

- 授权是否可以精确到应用级别？
- 如何给RAM用户添加创建集群的权限？
- 如何手动打开角色收敛的RAM授权页面？
- 被授权了cs:admin权限的RAM用户在K8s无法创建CRD
- 页面提示没有权限，报错详情：APISERVER_403
- ECS的RAM角色被删后如何恢复？
- 已经授予了RAM用户对所有集群的只读权限，但是有些集群仍然看不到
- 如何在Kubernetes集群指定RAM中的自定义角色？
- RAM用户如何给其他RAM用户进行RBAC授权？
- 子账号使用Kubernetes集群出现“您没有权限进行当前操作。如果您是子账号用户，请联系主账号进行授权”报错
- 根据错误信息判断是RAM授权问题还是RBAC授权问题
- 已经有AdministratorAccess或AliyunCSFullAccess权限，仍旧无法访问集群

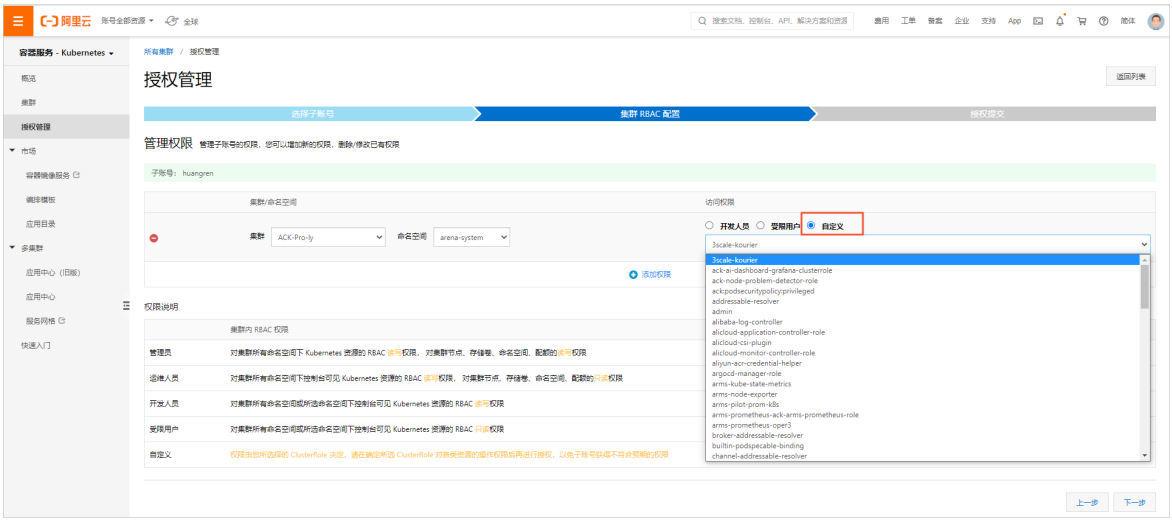
授权是否可以精确到应用级别？

可以。您可以通过自定义ClusterRole策略，在策略规则中的 `resourceNames` 字段指定相应应用实例名称实现应用级别的授权。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**授权管理**。
3. 在**授权管理**页面中的**选择子账号**配置向导中选择需要授权的RAM用户，单击**管理权限**。

   > **说明** 如果您使用RAM用户授权，请先确保该RAM用户已完成集群RBAC配置自定义RAM授权策略文档中的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 **添加权限**，指定集群和命名空间，选中**自定义**角色，并在**自定义**的下拉列表中选择对应的ClusterRole名称，完成后单击**下一步**。



说明 当前针对RAM用户的授权，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：
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Kubernetes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Kubernetes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Kubernetes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RAM用户获得不符合预期的权限。详情请参见容器服务自定义 RAM 授权策略。

相关后续操作步骤，请参见配置RAM用户RBAC权限。

如何给RAM用户添加创建集群的权限？

1. 用阿里云账号进行容器服务系统角色的批量授权。
   - 服务角色的授权是一次性操作，如果您不确定是否已经完成，可以使用阿里云账号登录后打开以下链接做容器服务系统角色一次性的批量授权：ur.alipay.com/1paTcxSWdAEW70GVH5TZiO
   - 关于容器服务默认系统角色说明，请参见容器服务默认角色。
2. 主账号对目标子账号进行自定义RAM授权策略操作。

   请确保目标RAM账号至少有 `cs:CreateCluster` 权限。具体操作，请参见自定义RAM授权策略。
   YAML文件模板样例如下：

```
{
 "Statement": [{
   "Action": [
     "cs:*"
   ],
   "Effect": "Allow",
   "Resource": [
     "CreateCluster"
   ]
 }],
 "Version": "1"
}
```

> **说明**
>
> - 由于集群创建过程中需要选择关联的VPC等云资源，请根据需要进行对应的RAM授权操作。
> - 请确保目标子账号至少有VPC资源的List权限，该权限的授权可以通过授权系统策略AliyunVPCReadOnlyAccess完成。
> - 如果有其他的资源配置需要，请参考对应服务的系统策略或授权文档进行授权操作。更多信息，请参见RAM授权。

如何手动打开角色收敛的RAM授权页面？

因为某种原因把ACK的RAM授权给删了，需要重新授权。

具体操作，请参见步骤二：一键授权默认角色。

> **说明** 重新授权需要用阿里云账号操作。

被授权了 cs:admin 权限的RAM用户在K8s无法创建CRD

部分容器服务集群（2019年05月以前创建的集群）的默认管理员权限并不包含全部K8s资源访问权限，您可以绑定一个自定义的 cluster-admin ，或者删除集群已有 cs:admin 的ClusterRole，然后重新创建ClusterRole。

YAML文件模板样例如下：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: cs:admin
rules:
- apiGroups:
 - '*'
 resources:
 - '*'
 verbs:
 - '*'
- nonResourceURLs:
 - '*'
 verbs:
 - '*'
```

页面提示没有权限，报错详情：APISERVER_403

当前RAM用户没有被授予K8s集群相应的RBAC权限，需要在**授权管理**页面对该账号进行授权。具体操作，请参见配置RAM用户RBAC权限。

ECS的RAM角色被删后如何恢复？

ECS节点上运行的应用中请求 metadata api 100 地址时提示404或Message:Node condition RAMRoleError is now: True, reason: NodeHasNoRAMRole。您可以选择以下方案尝试恢复ECS的RAM角色：

- ECS的RAM角色被删后，您需要为节点授予相应的RAM角色。具体操作，请参见更换实例RAM角色。
  - Master节点需要授予**集群信息 > 集群资源 > Master RAM角色**的RAM角色。
  - Worker节点需要授予**集群信息 > 集群资源 > Worker RAM 角色**的RAM角色。
- 如果您修改了RAM角色的策略内容，需要确认修改的内容是否有问题。
- 如果您是在上报提示前修改了RAM角色，请尝试恢复为上报提示前RAM角色策略内容授权版本。

已经授予了RAM用户对所有集群的只读权限，但是有些集群仍然看不到

问题现象

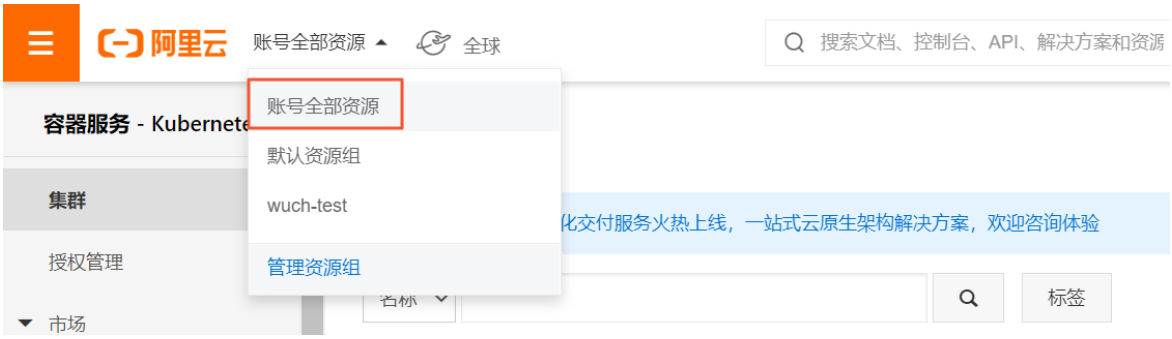
通过RAM控制台给RAM用户授权了全部集群的只读权限，通过RBAC授予了其中两个集群指定命名空间的访问权限。之前RAM用户在控制台可以看到所有集群，然而现在只能看到部分集群。用户的RAM权限正常，且近期没有做权限变更。

问题原因

您没有使用对应的RAM用户登录RAM控制台或者使用了特定的资源组，切换为使用相应的RAM用户登录RAM控制台并且资源组切换为**账号全部资源**。

解决方案

1. 登录容器服务管理控制台。
2. 在顶部菜单栏，选择**账号全部资源 > 账号全部资源**。



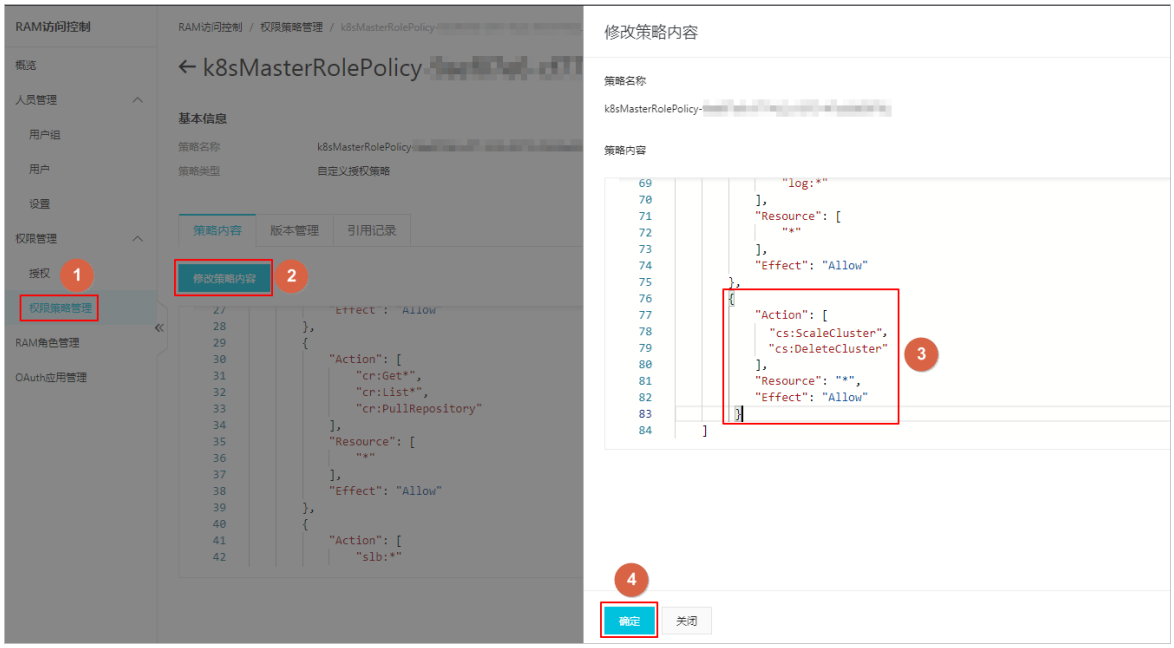
3. 将鼠标移至控制台右上角头像位置，确保是子账号登录。

如何在Kubernetes集群指定RAM中的自定义角色？

Kubernetes集群暂不支持指定RAM中的自定义角色，但是在Kubernetes集群中创建Worker时会自动生成一个角色，您可以通过给该角色添加Policy的方式，来实现角色授权。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群信息管理页单击**集群资源**页签。
5. 在**集群资源**页面，单击**Worker RAM角色**右侧的链接。
6. 在**RAM 角色管理**页面，单击权限策略名称。
7. 进入**策略权限管理**页面，单击**修改策略内容**，将以下内容填入后，单击**确定**。本示例中授予该角色伸缩和删除集群的权限。如果您想了解更多的权限，请参见自定义RAM授权策略。

```
{
      "Action": [
       "cs:ScaleCluster",
       "cs:DeleteCluster"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
```



RAM用户如何给其他RAM用户进行RBAC授权？

默认情况下，RAM用户（该例用RAM用户A描述）不具备对其他RAM用户授权RBAC的能力，需要首先确保RAM用户A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，同时给RAM用户A授予RAM相应权限，策略内容包括：

- 查看其他RAM用户
- 授予RAM权限策略
- 查看RBAC权限配置
- RBAC授权能力

1. 向RAM用户A授予RAM权限。

   登录RAM控制台，向RAM用户A授予RAM权限，请参见自定义RAM授权策略。
   授权策略内容示例参考：

```
{
  "Statement": [{
      "Action": [
        "ram:Get*",
        "ram:List*",
        "cs:GetUserPermissions",
        "cs:GetSubUsers",
        "cs:GrantPermission"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ram:AttachPolicyToUser",
        "ram:AttachPolicy"
      ],
      "Effect": "Allow",
      "Resource": [
        "acs:ram:*:*:policy/xxxxxx",
        "acs:*:*:*:user/*"
      ]
    }
  ],
  "Version": "1"
}
```

说明 *xxxxxx*需要替换成您想要绑定的RAM策略名称。例如，您替换成*，表示RAM用户A拥有所有RAM策略的授权绑定能力。

2. RAM用户A给其他RAM用户授权。

   当RAM用户A完成了上述策略的绑定后，即拥有对其他RAM用户在指定策略范围内的RAM授权能力，以及在集群内的RBAC授权能力。您可以参见配置RAM用户RBAC权限，对其他RAM用户授权。

根据错误信息判断是RAM授权问题还是RBAC授权问题

可以通过API或控制台的错误信息快速判断是RAM权限缺失还是RBAC权限缺失导致的问题。

- RAM权限缺失问题

  问题现象

  API或控制台上报类似下方的错误信息：

  ```
  RAM policy Forbidden for action cs:DescribeEvents
  STSToken policy Forbidden for action cs:DescribeClusterNodes
  ```

  问题原因

  从上述错误信息可以得出，RAM用户未授权RAM Action `cs:DescribeEvents` 的权限。

  解决方案

  如果API或控制台的错误信息中包含 `RAM policy Forbidden` 或 `STSToken policy Forbidden` ，说明是RAM权限缺失导致的问题。请对相应的RAM权限进行授权，具体操作，请参见自定义RAM授权策略。

- RBAC权限缺失问题

  问题现象

  API或控制台上报类似下方的错误信息：

```
events is forbidden: User "<uid>" cannot list resource "events" in API group "" at the cluster scope
ForbiddenQueryClusterNamespace, Forbidden query namespaces
```

问题原因

从上述错误信息可以得出，用户 `<uid>` 未被授权列出资源事件的权限。

解决方案

如果API或控制台的错误信息中包含 `APISERVER_403` 、 `User "xxx" cannot xx resource "xx" in API group` 或 `ForbiddenQueryClusterNamespace` ，说明是RBAC权限缺失导致的问题。请对相应角色添加对应集群内RBAC权限，具体操作，请参见配置RAM用户RBAC权限。

已经有AdministratorAccess或AliyunCSFullAccess权限，仍旧无法访问集群

容器服务ACK的授权体系包含RAM授权和RBAC授权两部分。更多信息，请参见授权概述。在RAM管理控制台完成AdministratorAccess或AliyunCSFullAccess的RAM授权后，还需要在**授权管理**页面完成RBAC授权后才能访问集群。关于如何进行RBAC授权，请参见配置RAM用户RBAC权限。

4.集群

4.1. 迁移至Pro集群

4.1.1. 热迁移ACK标准托管集群至Pro托管集群

为了能够让存量的ACK标准托管集群用户享受到ACK Pro托管集群的功能和特性，容器服务平台推出了标准版到Pro版的集群热迁移功能，使您可以动态地迁移ACK标准托管集群至Pro托管集群。本文介绍如何迁移ACK标准托管集群至Pro托管集群中。

背景信息

ACK Pro托管集群是在ACK标准托管集群基础上针对企业大规模生产环境，进一步增强了可靠性、安全性，并且提供可赔付的SLA的Kubernetes集群。关于ACK Pro托管集群的详细信息，请参见ACK Pro版集群概述。

注意事项

- 热迁移ACK标准托管版集群至Pro托管版集群功能，只支持集群Kubernetes 1.16及以上版本。
- ACK不支持将Pro托管版集群迁移到标准托管版集群。

迁移流程

ACK标准托管集群到Pro版的集群迁移流程涉及两个部分：集群迁移前置检查和集群迁移。

只有通过了集群迁移前置检查，ACK标准托管集群才会被开始迁移至ACK Pro托管集群中。

- 如果迁移前置检查未通过，您可以根据提示跳转到对应的检查页面，查看检查未通过的具体原因。
- 如果迁移前置检查顺利通过，则正式开始集群迁移。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，找到需要迁移的标准托管版集群，然后在**操作**列下单击**迁移至Pro版**。
4. 在**迁移至Pro版**对话框中，阅读**注意事项**然后单击**确定**。



单击**确定**后，容器服务会先对您的集群进行迁移前置检查。如果检查未通过，则可以通过提示跳转到检查结果详情页面，进行处理。如果检查顺利完成，则进入到集群迁移流程中。

5. 迁移流程结束后，在**迁移至Pro版**对话框中，单击**确定**完成集群迁移。
   成功迁移的ACK标准托管集群的**集群类型**由**标准托管版**自动变更为**Pro托管版**。

相关文档

您还可以迁移ACK专有版集群至Pro托管版集群。具体操作，请参见热迁移ACK专有版集群至Pro托管集群。

4.1.2. 热迁移ACK专有版集群至Pro托管集群

为了能够让存量的ACK专有版集群用户享受到ACK Pro托管版集群的功能和特性，容器服务平台推出了专有版到Pro托管版的集群热迁移功能，使您可以动态地迁移ACK专有版集群至Pro托管版集群。本文介绍如何迁移ACK专有版集群至Pro托管集群中。

前提条件

- 准备一个ACK专有版集群。具体步骤，请参见创建Kubernetes专有版集群。
- 集群的Master节点上已安装云助手。更多信息，请参见云助手概述和安装云助手客户端。
- 集群的内网SLB实例规格的配置为slb.s1.small及以上。
- 已将管控面组件（API Server、Kube Controller Manager、Cloud Controller Manager、Scheduler）和其他kube-system中的DaemonSet之外的Pod驱逐到Worker节点上。管控面组件在迁移之后会被托管的组件所替代。
- 创建一个OSS的存储空间（Bucket）。具体步骤，请参见创建存储空间。

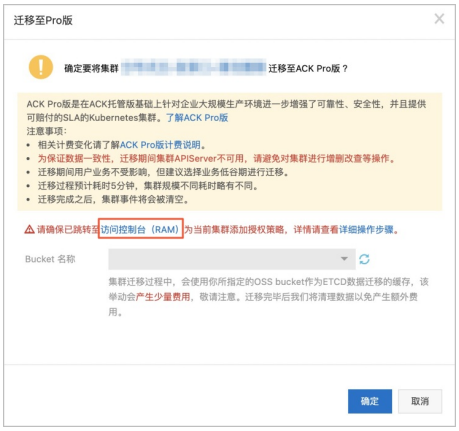
注意事项

- 迁移ACK专有版集群至Pro托管版集群为白名单功能，请提交工单申请使用该功能。
- 请确保您的ACK专有版集群的Kubernetes版本在1.16及以上。如不满足该条件，请升级您的集群版本。关于如何升级集群的Kubernetes版本的具体步骤，请参见升级ACK集群K8s版本。
- 如果集群的内网SLB的配置是共享版，则需要至少升级其到slb.s1.small才能进行集群迁移。关于如果升级SLB实例规格，请参见按量付费实例升降配。
- 部分ACK专有版老集群仍使用公网SLB访问API Server，此类集群迁移到Pro托管版后无法继续通过公网SLB访问，需要手动切换为EIP模式（将EIP绑定到API Server内网SLB），以支持公网访问API Server。关于手动切换EIP模式的具体操作，请参见控制集群API Server的公网访问能力。
- 迁移完成后，需注意：
  - 原有专有版集群的Master节点将脱离集群（状态变为**未知**），终止使用。
  - 不支持将Pro托管版集群回退至专有版集群。
  - 原有专有版集群的Master节点的ECS实例不会被自动删除，请注意及时删除不需要的ECS实例。

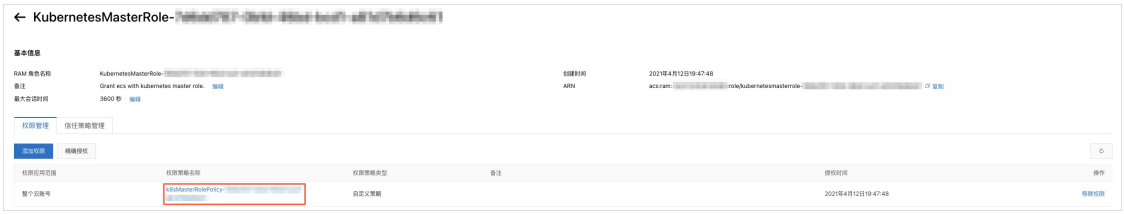
热迁移ACK专有版集群至Pro托管集群

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在集群列表页面，找到需要迁移的专有版集群，然后在**操作**列下单击**迁移至Pro版**。
4. 在**迁移至Pro版**对话框中完成专有版集群迁移至Pro托管版集群的授权操作，然后单击**确定**。
   迁移专有版集群至Pro托管版集群，您需要添加以下迁移所需的权限。

i. 在**迁移至Pro版**对话框中，单击**访问控制台（RAM）**。



ii. 在Master RAM角色的详情页，单击以**k8sMasterRolePolicy**开头的授权策略名称。



iii. 在目标授权策略的详情页的**策略内容**页签，单击**修改策略内容**。

iv. 在**修改策略内容**面板中的**Statement**字段中补充以下策略内容，然后单击**确定**。

```
,
    {
      "Action": [
        "oss:PutObject",
        "oss:GetObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "acs:oss:*:*:<YOUR_BUCKET_NAME>/*" # 需要将策略中的<YOUR_BUCKET_NAME>替换为您在迁移对话框中所指定的Bucket名称。
      ]
    }
```

添加策略后的示例如下图。



完成迁移之后，原有的Master节点状态会变成**未知**。查看节点状态步骤如下：

i. 在集群列表页面，单击目标集群右侧**操作**列下的**详情**。

ii. 在集群管理左侧导航栏中，选择**节点管理 > 节点**。

iii. 在节点列表的**角色/状态**列下，查看Master节点的状态。

删除专有版集群热迁移后的Master节点

在完成迁移专有版集群至Pro托管版集群后，您可以自行将Master节点从集群中删除。目前ACK暂时不支持从控制台直接删除Master节点，您可以通过**kubectl**命令方式删除Master节点。

在执行命令前，请确保您可以使用**kubectl**命令连接集群。关于如何使用kubectl命令连接集群的具体操作，请参见通过kubectl工具连接集群。

1. 执行以下命令查看需要删除的Master节点名称。

```
kubectl get node | grep master
```

2. 执行以下命令删除目标Master节点。

```
kubectl delete node <MASTER_NAME>
```

> **说明** 将<MASTER_NAME>替换为上步中查询到的Master节点名称。

后续步骤

若ACK专有版集群已安装共享GPU基础版，在迁移至ACK Pro托管版集群后，需要将共享GPU基础版升级为共享GPU专业版。具体操作，请参见在ACK Pro版集群中将共享GPU基础版升级为共享GPU专业版。

4.2. 操作系统

4.2.1. ACK等保加固使用说明

本文介绍ACK基于Alibaba Cloud Linux 2实现等保2.0三级版以及如何配置等保合规的基线检查。

背景信息

阿里云根据国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》中对操作系统提出的一些等级保护要求，ACK基于云原生操作系统Alibaba Cloud Linux实现等保2.0三级版。您可以使用ACK的等保加固配置满足以下等保合规要求：

- 身份鉴别
- 访问控制
- 安全审计
- 入侵防范
- 恶意代码防范

ACK使用Alibaba Cloud Linux等保2.0三级版

在创建ACK集群时，您可以选择配置启用**等保加固**，这样集群在创建时会自动配置对应的等保加固项，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》中对操作系统的等级保护要求。

等保加固配置图



> 注意
>
> - 为了满足等保2.0三级版的标准要求，ACK会在等保加固的Alibaba Cloud Linux 2操作系统中默认创建ack_admin、ack_audit、ack_security三个普通用户。
> - 为了满足等保2.0三级版的标准要求，等保加固的Alibaba Cloud Linux 2禁止使用Root用户通过SSH登录。您可通过ECS控制台使用VNC方式，登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。

4.2.2. ACK CIS加固使用说明

您可以在ACK集群中启用CIS加固功能，提升ACK集群所有节点的操作系统的安全性。本文介绍ACK基于Alibaba Cloud Linux 2实现CIS安全加固以及如何配置CIS安全加固后的检查。

背景信息

CIS（Center for Internet Security）是一个第三方安全组织，致力于采用线上社区的模式与大公司、政府机构、学术机构一起打造优秀的安全实践解决方案。当前各大个公司发布的Linux操作系统大多都已经提供CIS Benchmark，包括Alibaba Cloud Linux 2、CentOS、Ubuntu等。当前发布CIS Benchmark已经成为很多阿里云客户对于操作系统安全的重要评判依据之一。更多信息，请参见CIS WorkBench。

Alibaba Cloud Linux 2不仅是阿里云官方操作系统镜像，也是ACK的首选默认系统镜像。Alibaba Cloud Linux 2在2019年8月16日正式通过了CIS组织的全部认证流程并对外发布CIS Aliyun Linux 2 Benchmark version 1.0.0。更多信息，请参见CIS Aliyun Linux 2 Benchmark version 1.0.0。

CIS Aliyun Linux 2 Benchmark version 1.0.0加固项分类说明

CIS Aliyun Linux 2 Benchmark version 1.0.0是最新的CIS Aliyun Linux 2 Benchmark。您可以对其进行下载，具体操作，请参见下载CIS Aliyun Linux 2 Benchmark version 1.0.0。CIS Aliyun Linux 2 Benchmark version 1.0.0一共包含204项，按照每一项所处的安全等级分为Level 1（168项）和Level 2（36项）。其中Level 1与Level 2的主要区别如下：

- Level 1是说明此加固条目是基础项加固且不会带来较大的性能影响。
- Level 2说明此条目适用于安全性更高的场景，且可能会对系统带来性能开销。

此外CIS Aliyun Linux 2 Benchmark从计分（Scoring Information）的维度分为了Scored和Not Scored两类：

- Scored：意味着此条目会纳入计分项，如果验证系统是安全的，则加分，如果不安全，则减分。
- Not Scored：意味着无论是否安全，都不纳入计分项，也就是说不增减分。

综上，CIS Aliyun Linux 2一共204项，可以分为四类：

- Level 1 Scored（共145项）
- Level 1 Not Scored（共21项）
- Level 2 Scored（共33项）
- Level 2 Not Scored（共3项）

因为Level 2可能会影响性能且Not Scored项不纳入计分，因此对节点的Alibaba Cloud Linux 2操作系统，**ACK集群只针对Level 1 Sorced项进行安全加固**。

使用Alibaba Cloud Linux 2 CIS加固版

在创建ACK集群时，您可以选择启用CIS加固，这样集群在创建时会自动配置对应的CIS加固项，使得ACK集群所有节点的Alibaba Cloud Linux 2镜像满足CIS Aliyun Linux 2 Benchmark version 1.0.0中大部分Level 1 Scored项的加固要求。关于具体满足的加固项，请参见ACK CIS加固集群满足的CIS Level 1 Scored加固项清单。



> **说明** 为了满足CIS Level 1的加固要求，ACK会在CIS加固的Alibaba Cloud Linux 2操作系统中默认创建ack_cis这个普通用户。

ACK CIS加固集群满足的CIS Level 1 Scored加固项清单

经过对CIS Aliyun Linux 2 Benchmark version 1.0.0所有Level 1 Scored（共145项）进行分析和测试，容器服务ACK团队对除下表所列的17项以外的128项（88%以上）进行了安全加固。

ACK集群中未加固的CIS Level 1 Scored项清单

加固项	未加固的原因
1.1.2 Ensure /tmp is configured (Scored)	涉及到分区的修改。
1.1.18 Ensure sticky bit is set on all world-writable directories (Scored)	影响ACK管控逻辑。
1.7.1.1 Ensure message of the day is configured properly (Scored)	需要删除Alibaba Cloud Linux 2系统MOTD中的使用指南的链接。
3.1.1 Ensure IP forwarding is disabled (Scored)	对ACK的网络组件有影响。
3.5.1.1 Ensure default deny firewall policy (Scored)	需要您设置自己的防火墙策略。
3.5.1.2 Ensure loopback traffic is configured (Scored)	需要您设置自己的Loopback规则。
3.5.1.4 Ensure firewall rules exist for all open ports (Scored)	需要您对开放的端口设置防火墙策略。
3.5.2.1 Ensure IPv6 default deny firewall policy (Scored)	需要您设置自己的IPv6防火墙策略。
3.5.2.2 Ensure IPv6 loopback traffic is configured (Scored)	需要您设置自己的IPv6 Loopback规则。
4.2.1.4 Ensure rsyslog is configured to send logs to a remote log host (Scored)	需要您配置Rsyslog来设置自己的远程Log host。
4.2.3 Ensure permissions on all logfiles are configured (Scored)	改动文件太多，存在潜在风险。
5.2.10 Ensure SSH root login is disabled (Scored)	需要您自行创建其它登录账号或者选择VNC等非SSH登录方式，然后才能禁用Root登录。
5.2.18 Ensure SSH access is limited (Scored)	需要您配置允许登录的用户和组。
5.2.3 Ensure permissions on SSH private host key files are configured (Scored)	扫描脚本有误，`ssh_keys`的GID被硬编码为998，但实际系统不一定是998（可能是996等）。
5.3.2 Ensure lockout for failed password attempts is configured (Scored)	Benchmark给出的修复建议跟Alibaba Cloud Linux 2系统对应的配置文件差异较大，建议谨慎修改。
6.1.11 Ensure no unowned files or directories exist (Scored)	影响ACK管控逻辑。
6.1.12 Ensure no ungrouped files or directories exist (Scored)	影响ACK管控逻辑。

您可以参考CIS Aliyun Linux 2 Benchmark version 1.0.0手册中的以下条目，修复ACK集群中没修复的CIS Level 1 Scored项。根据以下内容中的**Remediation**部分进行修复，然后按照**Audit**检查您的修复是否生效。

条目	描述
Profile Applicability	该项属于Level 1还是Level 2。
Description	加固条目的简单介绍。
Rationale	用于描述条目的细节和背景，告诉读者这么加固的意义和原因。
Audit	关键项，用于判断检测系统是否达标的脚本。根据此脚本的运行返回值来判断是否需要加固。
Remediation	关键项，如果Audit环节判断系统需要进行加固，那此环节就是执行脚本进行安全处理。
Impact	影响，主要来描述如果不进行正确配置可能会导致的影响。
References	参考文献。
CIS Controls	此条目对应的CIS control文档的讲解，需要注册后才能下载。

下载CIS Aliyun Linux 2 Benchmark version 1.0.0

1. 登录并访问CIS Benchmark官方主页。
2. 同时选中**Operating Systems**和**Linux**。



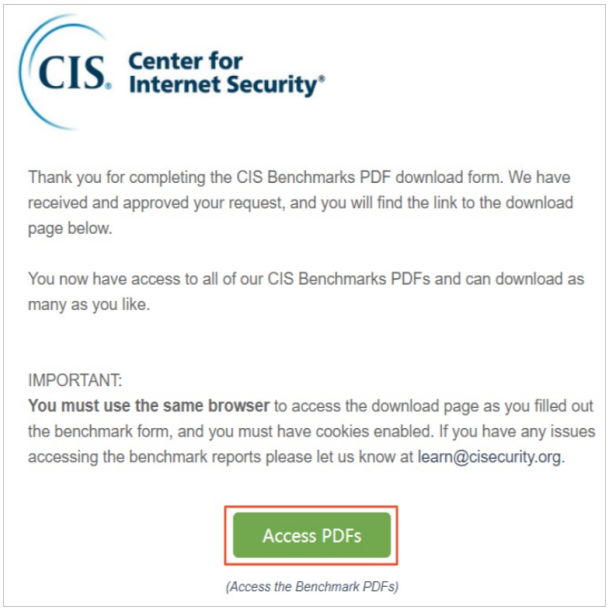
3. 找到Aliyun Linux CIS Benchmark，并单击**Download CIS Benchmark**。



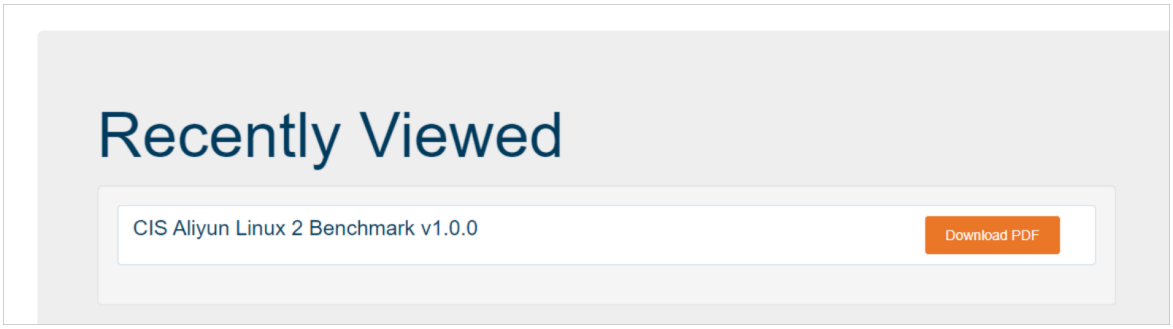
4. 在CIS下载界面完成填写个人的基本信息，然后单击**Get Free Benchmarks Now**。



5. 等待几分钟后，您填写的邮箱会收到CIS的邮件，单击邮件中的**Access PDFs**。



6. 在下载页面找到最近访问的CIS Aliyun Linux 2 Benchmark v1.0.0并单击**Download PDF**。



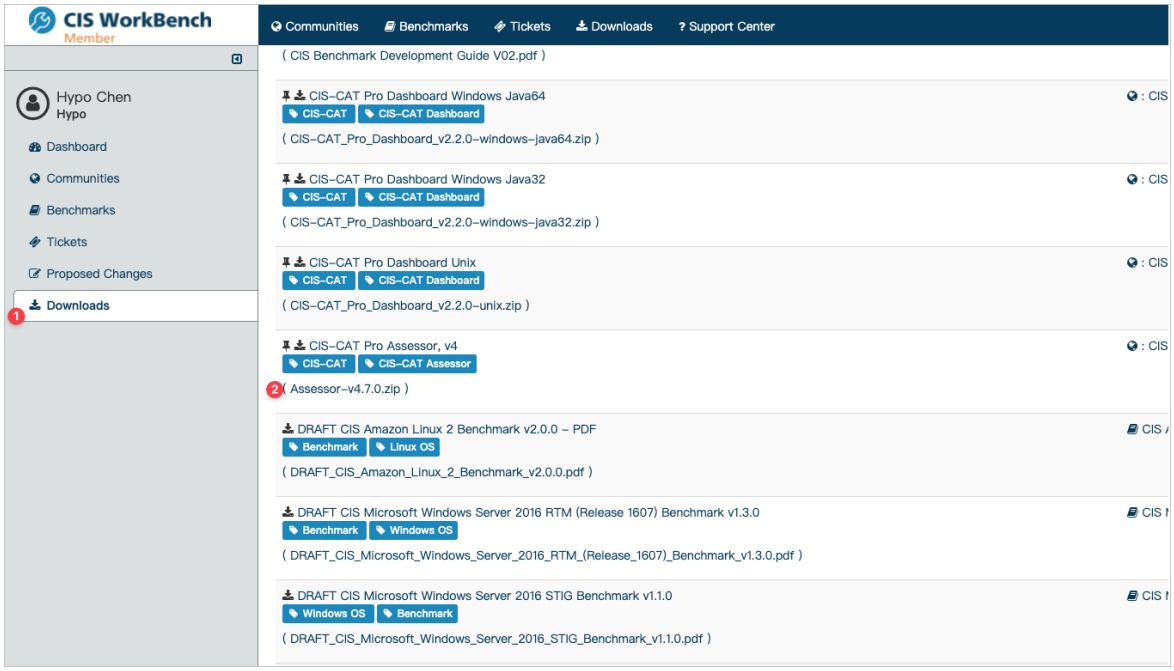
使用CIS-CAT扫描工具验证ACK集群的CIS加固效果

如果您想验证ACK集群的CIS加固效果，可以使用CIS官网提供的CIS-CAT工具进行扫描验证。CIS-CAT是一个安全配置评估软件工具，即扫描工具，它能够详细地提供目标系统的评估结果。通过运行该工具，可以得到目标系统在指定CIS Benchmark profile的合规分数，同时还会针对不合规的配置给出修复步骤。更多信息，请参见CIS-CAT工具。

CIS-CAT分为Lite和Pro版，Lite提供的功能有限，且仅支持的系统包括Windows 10, Ubuntu 18.04和Google Chrome等，不支持Alibaba Cloud Linux 2操作系统，因此无法扫描ACK集群的CIS加固效果。

CIS-CAT Pro具有v4和v3两个版本系列。以v4为例，本文将阐述如何使用CIS-CAT Pro进行扫描验证ACK集群的CIS加固效果。

1. 登录CIS SecureSuite，注册成为CIS SecureSuite会员，并下载CIS-CAT Pro工具包（Assessor-CLI-v4.0.23.zip）。



2. 登录到ACK集群的任一Alibaba Cloud Linux 2操作系统的节点。

   关于连接ACK集群ECS节点的具体方式，请参见查看节点列表和连接方式概述。

3. 分别执行以下命令安装CIS-CAT需要的Java环境。

```
yum -y install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

```
cat > /etc/profile.d/java8.sh <<EOF
export JAVA_HOME=$(dirname $(dirname $(readlink $(readlink $(which javac)))))
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:$JAVA_HOME/jre/lib:$JAVA_HOME/lib:$JAVA_HOME/lib/tools.jar
EOF
```

```
source /etc/profile.d/java8.sh
```

4. 分别执行以下命令使用CIS-CAT Pro工具（Assessor-CLI-v4.0.23.zip）进行扫描。

```
unzip Assessor-CLI-v4.0.23.zip
```

```
cd Assessor-CLI
```

```
chmod +x ./Assessor-CLI.sh
```

```
./Assessor-CLI.sh -b ./benchmarks/CIS_Aliyun_Linux_2_Benchmark_v1.0.0-xccdf.xml -p "Level 1" -html
```

> **说明**
>
> - `-b`：用来选择Benchmark的类型（操作系统和对应的版本类型）。
> - `-p`：用来选择扫描的Level，因ACK只对CIS Level 1 Scored进行扫描，因此扫描时选择Level 1。

5. 查看扫描结果。

```
------------------------------------------------------------------------------
***** Assessment Results Summary *****
------------------------------------------------------------------------------
  Total # of Results: 204
Total Scored Results: 145
         Total Pass: 128
         Total Fail: 17
        Total Error: 0
      Total Unknown: 0
Total Not Applicable: 0
   Total Not Checked: 11
  Total Not Selected: 36
 Total Informational: 12
------------------------------------------------------------------------------
***** Assessment Scoring *****
------------------------------------------------------------------------------
        Score Earned: 128.0
   Maximum Available: 145.0
               Total: 88.28%
------------------------------------------------------------------------------
```

扫描结果解释如下。更多信息，请参见CIS-CAT Pro Assessor v4 Report。

参数	解释
Total # of Results	所选择的Benchmark的加固项总数，CIS_Aliyun_Linux_2_Benchmark_v1.0.0总共204项。
Total Scored Results	所选Level的Scored项总数，Level 1一共145项。
Total Pass	所选Level的Scored项中当前系统通过的总数，CIS加固的ACK集群每个节点通过的Level 1 Scored项为128项。
Total Fail	所选Level的Scored项中当前系统未通过的总数，CIS加固的ACK集群每个节点未通过的Level 1 Scored项为17项。
Total Error	所选Level的Scored项中扫描脚本执行出错的项，这里CIS-CAT Pro都正确返回，因此结果为0。
Total Unknown	所选Level的Scored项中，CIS-CAT无法判断是否满足安全加固标准的项，这里结果为0。
Total Not Applicable	所选的Benchmark中与当前操作系统不匹配的项。在Alibaba Cloud Linux 2上执行CIS_Aliyun_Linux_2_Benchmark_v1.0.0的CIS-CAT Pro扫描不存在不匹配的项。
Total Not Checked	与Total Informational一样同属于Not Scored。
Total Not Selected	所选Benchmark中未检查的加固项。CIS-CAT Pro对Level 1进行检查，因此这里未检查的加固项为所有的Level 2（共36项）。
Total Informational	需要手动评估是否满足，在所选Level中属于Not Scored。

相关文档

- Alibaba Cloud Linux 2概述
- CIS Aliyun Linux 2 Benchmark version 1.0.0

- CIS-CAT工具
- CIS-CAT Pro Assessor v4 Report

4.2.3. 使用操作系统Alibaba Cloud Linux 2

ACK目前已经全面支持Alibaba Cloud Linux 2的节点创建，并结合Alibaba Cloud Linux 2的高内核特性提供了多场景的优化。本文介绍ACK中使用Alibaba Cloud Linux 2操作系统的优势，及ACK基于Alibaba Cloud Linux 2提供的多场景优化。

背景信息

Alibaba Cloud Linux 2是新一代阿里云原生Linux操作系统，为云上应用程序提供安全、稳定、高性能的定制化运行环境，并针对云基础设施进行了深度优化，为您打造更好的运行时体验。您可以免费使用Alibaba Cloud Linux 2公共镜像，并免费获得阿里云针对该操作系统的长期支持。

使用Alibaba Cloud Linux 2操作系统镜像的优势

Alibaba Cloud Linux 2的操作系统是针对阿里云飞天虚拟化平台开发的，针对阿里云的环境提供了很多优化和新特性，主要包括：

- 是阿里云上启动最快的Linux发行版。
- 针对ECS大规格虚拟机、裸金属服务器深度优化，特别是大规格实例的多任务的场景。
- 默认安装阿里云常用软件包，如阿里云CLI、cloud-init等，减少云资源的管理成本。
- 精简系统，最小的安全攻击面，最少的系统资源占用。
- 更完善的支持体系，在阿里云上提供多种渠道的技术支持体系。
- 更及时的软件安全漏洞（CVE）修复。
- 支持内核热补丁升级的能力，大大保障漏洞修复时的业务连续性。

应用实例在Alibaba Cloud Linux 2上运行的性能优势包括：

- 针对ECS实例环境大幅优化启动速度，在系统压力来临时快速的扩容启动计算资源，启动速度相对CentOS 7提升29%。
- 针对ECS实例环境优化了多任务的场景，提升大规格实例多任务的性能，同样规格下能有16%的性能提升。
- 更高效的系统调用，系统调用的性能提升11%。
- Linux网络栈的优化，综合网络性能相对CentOS 7提升7.8%。
- Alibaba Cloud Linux 2默认是编译好BBR的拥塞控制算法的，在公网访问较多的场景中允许您修改拥塞控制算法为BBR（Bottleneck Bandwidth and RTT），提升公网访问的带宽稳定性。
- 针对TLS协议，进行了加密优化。
- 支持新的Budget Fair Queueing的IO调度器，降低云盘的延迟。

ACK结合Alibaba Cloud Linux 2的场景优化

阿里云容器化业务通过内核级的优化，让容器任务混布的密度提升的同时不影响在线业务的响应。Alibaba Cloud Linux 2的操作系统和Kernel中包含了这些优化。ACK针对这些优化特性，提供了多场景的优化，促进容器化业务更快更平稳的运行。

- **IPVS优化**
  - 场景一：大规格机器（CPU数大于64）且有大量IPVS虚拟IP的场景。
    解决问题：IPVS的统计定时器（estimation timer）会定期计算各个连接当前的速率，在连接数多的情况下，该操作会长时间占用CPU，导致网络收包不及时，引起**Ping**命令高达200 ms的波动。
    优化方案：将IPVS的统计定时器放到节点中执行；同时添加**sysctl**命令关掉IPVS的统计。
    效果：不再有由于统计定时器引起的波动。

- 场景二：容器滚动升级的场景。
  解决问题：在容器发生了滚动升级的时候，如果五元组没有发生变化，新的TCP SYN包命中了旧的IPVS五元组连接记录，并且需要被调度到新的目的地址的时候，IPVS默认会丢掉SYN包，导致SYN包重传，从而引发1秒的延时问题。
  优化方案：在新Conntrack项已经存在的情况下，通过释放Conntrack中的TIME_WAIT状态的连接，将其调度并替换为新连接。
  效果：可以几乎没有时延的切换到新的真实节点（real server）。

- **CoreDNS优化**
  场景一：容器内大量DNS查询导致Conntrack表满的场景。
  解决问题：容器内的应用查询固定的地址（addr）或端口（port）的DNS会让相应Conntrack entry变成stream mode状态。由于DNS请求的类型是UDP无状态、问答式、时间短，因此导致Conntrack entry维护了很多无用的UDP Conntrack entries，不能及时清理，进而可能导致Conntrack表膨胀引起NAT的性能下降。
  优化方案：
  - 只有UDP连接持续2秒以上才会被设置成Stream mode状态，从而避免了Conntrack entry的快速膨胀。
  - 缩短默认UDP Conntrack过期时间，从180s缩短到120s，让其更快过期，减少对Conntrack entry的影响。

  效果：相同测试场景下UDP的Conntrack表项下降到原来的一半。

- **容器网络性能优化**
  在Alibaba Cloud Linux 2的节点上，容器服务Terway网络插件支持IPVlan的容器网络模式，在小包的场景下相对传统Bridge和策略路由网络性能提升40%。Alibaba Cloud Linux 2默认编译了BBR的拥塞控制算法，在公网访问较多的场景中允许您修改容器的拥塞控制算法为BBR，提升公网访问的带宽稳定性，在容器的公网连接和跨公网的镜像拉取上性能大幅提升。

- **安全容器支持和优化**
  阿里云与Kata Containers和Clear Linux社区合作。在弹性裸金属实例上，您可以无缝地部署整套Kata Containers的解决方案。ACK还优化了安全沙箱（RunV）镜像的启动时间，使得Kata Containers的整体解决方案可以正常运行。ACK在此之上，提供了和普通集群体验几乎一致的安全沙箱容器集群，让应用运行在一个轻量虚拟机沙箱环境中，适合于多用户间的负载隔离，对不可信应用的隔离，在提升安全性的同时，对性能影响也做到非常小。

- **AutoScaler优化**
  Alibaba Cloud Linux 2针对ECS实例环境优化启动速度，节点的启动时间相对Centos 7的系统降低60%，结合容器服务ACK灵活高效的自动弹性伸缩，在应用压力来临时，容器服务ACK集群会自动根据负载状况创建和启动ECS的节点加入集群，并调度和启动应用实例，Alibaba Cloud Linux 2的快速的扩容启动的能力让计算资源可以及时满足流量峰值的需求。

- **资源监控和控制能力优化**
  Alibaba Cloud Linux 2的内核提供了针对容器场景的PSI压力模型、per-cgroup kswapd、Memory Priority等的细粒度可视化和控制能力。在Alibaba Cloud Linux 2的ACK集群中我们可以通过CGroup Controller来利用这些能力，可以实现细粒度BufferIO Control、TCP、CPUSet、Mem、NUMA等细粒度资源的配置和动态更新，在逐步提升资源利用率的同时也能保障应用间的互相干扰降到最低。

- **AI和数据加速优化**
  Alibaba Cloud Linux 2对大规格机型和多任务的优化可以提升高性能计算任务的速度，对存储的流式读写的优化也可以提高模型大文件的读写性能，综合起来大大加速AI和高性能计算任务的效率。实际测试场景如下：
  - 通过Alluxio利用64个线程加载OSS数据1152个文件144 GB，在CentOS需要3分25秒，Alibaba Cloud Linux 2只需2分19.037秒，速度是CentOS上的1.6倍。
  - 运行ResNet50 Batch 128模型训练，数据缓存到Alluxio中，在CentOS下V100仅仅为5212.00 images/s。在Alibaba Cloud Linux 2下V100可以达到8746.59 images/s，速度是CentOS上的1.7倍。

- **容器资源增强展示优化**
  由于当前多容器共宿主机的形态，导致容器中直观看到的资源都是宿主机资源，对很多应用非常不友好。
  Alibaba Cloud Linux 2对内核的CGroup资源展示进行了优化，可以在容器内部正确显示容器所占用的资源，例如TOP命令、CPUInfo和MemInfo接口的显示信息，大大方便了您的观测需求。

- **其他优化**
  - 使用Linux 4.19的内核，ACK将阿里巴巴内核和容器化实践融入其中。

- 降低Overlayfs的性能损耗，减少容器化对存储性能的损失。
- 大量Sysctl Namespace化，在4.19的内核中大部分Sysctl配置都支持在容器中单独设置，例如TCP的超时和重传时间，不同应用有不同的超时和重传的需求，但是在CentOS 7的内核中是修改不了的，在Alibaba Cloud Linux 2中支持对其做Pod级别的配置。

使用Alibaba Cloud Linux 2作为集群节点系统镜像

您可以在创建集群的配置过程中，将**操作系统**选择为**Alibaba Cloud Linux 2.1903**来使用Alibaba Cloud Linux 2作为集群节点系统镜像。具体步骤，请参见创建Kubernetes专有版集群。



> **说明**
> 如果您选用Alibaba Cloud Linux 2，在创建集群以及后期扩容节点、添加节点、自动伸缩节点时，ACK会自动检测Alibaba Cloud Linux 2的安全补丁更新并自动安装补丁。

相关文档

- Alibaba Cloud Linux 2产品详情页
-
- 阿里云操作系统及Kernel团队博客
- 阿里云内核开源官网
- 体验Alibaba Cloud Linux 2不多花一分钱提升15%服务器性能
- Kubernetes网络的IPVlan方案

4.2.4. ContainerOS概述

ContainerOS是为容器场景垂直优化的阿里云官方操作系统，完全兼容现有的Kubernetes生态。ContainerOS在Alibaba Cloud Linux3的基础上，提供了更高的安全性、更快的启动速度和极为精简的系统服务与软件包，默认集成云原生组件，开箱即用。本文介绍ContainerOS的背景信息、特性以及优势。

适用范围

ContainerOS目前仅适用于阿里云ACK托管节点池，且容器运行时为Containerd。关于托管节点池的更多信息，请参见托管节点池概述。

背景信息

在云原生快速发展的大背景下，应用被容器化部署，云原生组件（容器运行时、Kubernetes）的出现使得您可以聚焦于应用的开发，而不必过多地关注底层基础设施的细节。传统的操作系统OS发行版为了适应更多的使用场景，包含了大量的用户态工具、软件包、系统服务等，使得OS变得极为臃肿、启动慢、各种软件包种类繁多、版本零散，给运维人员带来了极大的挑战。为了解决传统OS在云原生场景下的各种适配问题，给容器服务的用户带来更好的使用体验，一个专为容器场景而生的操作系统ContainerOS应运而生。

您可以在ACK托管节点池中免费使用ContainerOS镜像，并免费获得阿里云针对该操作系统的长期支持。

产品特性

- **镜像精简**
  ContainerOS只包含Kubernetes Pods运行所依赖的软件包和系统服务，通过全系统整合优化，极大地缩短了系统启动时间，精简的操作系统使得整个系统的攻击面也大大减小。ContainerOS不提供Python支持，不直接提供SSH登录功能，更多时候ContainerOS对于您是一个透明的存在。您只需要专注于业务的开发和运行，而不必关心操作系统层面的问题。
- **安全加固**
  ContainerOS根文件系统为只读权限，只有*/etc*和*/var*目录可写，以满足基础的系统配置需求。这种设计既符合云原生场景下的基础设施不可变原则，又能防止逃逸容器篡改主机文件系统。同时，ContainerOS禁止用户直接登录到系统中进行一系列可能无法追溯的操作，但仍然提供一个专用的运维容器满足非日常的运维需求。关于运维容器使用方法，请参见ContainerOS运维容器使用指导。
- **原子升级**
  ContainerOS遵循云原生不可变基础设施原则，不支持单个RPM包粒度的安装、卸载和升级，不提供YUM等软件包管理工具，只支持以整个操作系统镜像为粒度的更新和回滚，以此来保证集群内各个节点的软件包版本与系统配置的一致性。每个镜像经过内部严格的测试之后才会上线，相较于传统操作系统基于单个RPM包的升级带来的不确定性，以镜像为粒度的测试发布更能保证升级后系统的稳定性。

优势

- ContainerOS针对容器场景垂直优化，具有启动快、安全加固、根文件系统不可变等特征，既提升了性能，又方便集群范围的运维与管理，最大程度上保持集群内各个节点的一致性。
- 提升OS的可运维能力，镜像按需发布。ContainerOS与容器服务ACK管控配合，可持续更新Kubernetes和其它系统软件版本、修复CVE漏洞。
- ContainerOS与容器服务ACK联合优化，大大缩短了Kubernetes节点因为运维导致的不可用时间，有利于业务平稳运行。
- ContainerOS内核版本以及绝大部分软件包与Alibaba Cloud Linux3完全保持一致，采用最新的kernel 5.10 LTS，为云上应用程序环境提供Linux社区的最新操作系统功能。关于Alibaba Cloud Linux3的更多信息，请参见Alibaba Cloud Linux 3概述。

计费

ContainerOS是免费镜像，但当您使用ContainerOS镜像时，需要支付其他资源产生的费用，例如vCPU、内存、存储、公网带宽和快照等。关于其他资源的计费详情，请参见计费概述。

更新记录

关于ContainerOS镜像发布的更多信息，请参见操作系统镜像发布记录。

相关文档

- Alibaba Cloud Linux 3概述
- 操作系统镜像发布记录
- 托管节点池概述
- Containerd运行时发布记录

4.2.5. ContainerOS运维容器使用指导

为了达到更高的安全性，ContainerOS原则上不支持任何账户直接登录实例进行操作，也不提供SSH登录功能。若您仍然有登录实例进行运维的需求，可以启动并登录运维容器。本文介绍ContainerOS运维容器的常见操作。

前提条件

- 已安装ECS云助手客户端。管理运维容器需要通过ECS云助手调用专用的API工具来实现，关于ECS云助手的安装和使用方法，请参见安装云助手客户端和云助手概述。
- 实例或集群的登录类型为密钥对。运维容器仅支持密钥登录，这需要您在创建实例或集群时，选择登录凭证为密钥对。若创建时未选择密钥对登录，可以在实例创建完成后，为实例绑定一对密钥。关于绑定密钥的具体操作，请参见绑定SSH密钥对。

- 安全组已开放22端口，否则无法通过Workbench正常登录。关于开放安全组端口的具体操作，请参见安全组概述。

背景信息

运维容器相比于主机，拥有更多的软件包，同时也允许您通过包管理软件YUM安装需要的软件包。在运维容器中，您可以查看系统进程信息、网络信息、系统配置等。此外，运维容器里还提供了专用的命令，用于从容器中进入主机，效果等同于直接通过Workbench来登录实例。

进入主机环境

1. 登录ECS管理控制台，进入云助手**执行命令**面板。具体操作，请参见立即执行命令。
2. 在ECS云助手，执行以下命令，启动运维容器。

   ```
   lifseacli container start
   ```

   通过云助手成功启动运维容器后，预期输出：

   ```
   Downloading docker image...
   Successfully downloaded docker image
   Successfully start admin container!
   ```

3. 执行以下命令，通过指定的私钥登录到容器中。

   ```
   ssh -i <ssh-private-key.pem> admin@<instance-ip>
   ```

   `<ssh-private-key.pem>` 为您绑定到实例的密钥对中的私钥，`<instance-ip>` 为实例的IP。

   > **注意** 您也可以通过Workbench直接登录，用户名为admin，密钥为您绑定到实例的密钥对中的私钥。请务必保证您的实例开放了22号端口，否则无法通过Workbench正常登录。

   成功登录运维容器的界面如下所示。

   ```
   Welcome to LifseaOS maintenance container!

   This container is based on Alibaba Cloud Linux 3, therefore most things
   should behave the same as a Alibaba Cloud Linux 3 VM or container.
   Some common tools are pre-installed for debug purpose and you can install
   any tools you want through yum.

   Also you can access the LifseaOS host root filesystem (see /.lifsea/rootfs)
   within the container.

   If you want to place yourself in a real LifseaOS host environment, a tool
   named "superman" (`sudo superman`) is provided. Once run, you will go
   inside the LifseaOS host's rootfs with a root shell.

   Enjoy!

   [admin@iZbp1in          tziZ ~]$
   ```

   主机的根文件系统被挂载到运维容器的*/.lifsea/rootfs*目录（只读挂载），您可以直接在这个目录中检索您需要的系统信息、配置等。

4. 执行以下命令，进一步进入系统中。

   ```
   sudo superman
   ```

   进入主机环境，预期输出：

   ```
   [admin@iZbp1i          ~]$ sudo superman
   bash-4.4#
   ```

5. 执行 `ls` 命令，查询可使用的系统命令。

   预期输出：

```
[admin@iZbp1i                ~]$ sudo superman
bash-4.4#
bash-4.4# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  ostree  proc  root  run  sbin  srv  sys  sysroot  tmp  usr  var
bash-4.4#
```

可以看出在主机环境中，可用的系统命令有限。

其他操作

进入主机环境后，执行 exit 命令可退出主机环境，再次执行 exit 命令可退出运维容器。此时，运维容器仍然存在并处于运行状态，您可再次通过SSH登录进去，或者通过命令停止、重启、销毁运维容器。

- **停止运维容器：**

  ```
  lifseacli container stop
  ```

- **重启运维容器：**

  ```
  lifseacli container restart
  ```

  > **说明** 若您在启动运维容器之后，重新绑定或解绑了密钥对，则需要重启运维容器，否则绑定或解绑密钥对不会生效。

- **销毁运维容器：**

  ```
  lifseacli container rm
  ```

  > **注意** 销毁运维容器之后，您此前在容器中安装的软件、保存的文件也会随之一并销毁。若您重新启动一个运维容器，便是一个全新的环境，请尽量不要在运维容器中保存关键数据。

- **查询运维容器状态：**

  ```
  lifseacli container status
  ```

常见问题

登录出错，提示UNPROTECTED PRIVATE KEY FILE!错误怎么办？

问题现象

报错如下图所示。

```
$ssh -i manxi.pem admin@47
@@@
```

4.3.1. 创建Kubernetes专有版集群

Kubernetes专有版集群需要创建至少3个Master节点以保证高可用性，以及若干Worker节点，可对集群基础设施进行更细粒度的控制，但需要自行规划、维护、升级集群。本文介绍如何通过容器服务管理控制台创建Kubernetes专有版集群。

前提条件

登录RAM管理控制台和弹性伸缩控制台开通相应的服务。

> **说明**
> 您在使用集群过程中，请注意以下限制：
>
> - 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。
> - ACK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。如果您需要提高配额，请提交工单申请。
>   - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。
>   - 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的ACK集群的网络模式是Flannel时，集群的路由条目最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多路由条目数，您需要对目标VPC提交工单申请提高配额 。
>   - 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：
>   支持创建按量付费和包年包月的ECS实例。
>   实例创建后，您可以通过ECS管理控制台将按量付费转预付费，请参见按量付费转包年包月。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**创建集群**。
4. 单击**ACK专有版**页签，然后完成集群配置。
   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
地域	选择集群所在的地域。

配置项	描述
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。
账号全部资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
Kubernetes版本	显示当前ACK支持的Kubernetes版本。
容器运行时	支持**Containerd**、**Docker**和**安全沙箱**。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
IPv6双栈	开启IPv6双栈将创建双栈Kubernetes集群。双栈功能正在公测中，如需使用请提交工单申请。 **注意** ■ 仅1.20.4-aliyun.1及以上版本的Kubernetes集群支持IPv6双栈功能。 ■ 集群内部Worker和Master节点之间通信使用IPv4地址。 ■ 容器网络插件需选择Terway。 ■ 集群使用的VPC及ECS需要支持IPv6双栈。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。

配置项	描述
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体请参见Terway与Flannel对比。 ■ 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，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。 **说明** ■ 在Terway模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。 ■ 网络插件选择**Terway**时，会使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 **说明** ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。

配置项	描述
Terway模式	设置**网络插件**为**Terway**时，需要配置**Terway模式**： ■ 设置是否选中**Pod独占弹性网卡以获得最佳性能** ■ 如果选中，Pod将独占一个专有的弹性网卡。 ■ 如果不选中，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。 说明 当前只有白名单用户可使用上述**Pod独占弹性网卡以获得最佳性能**功能。提交工单申请使用。 ■ 设置是否选中**IPvlan** ■ 只在弹性网卡共享模式支持选中。 ■ 如果选中，采用IPvlan eBPF作为网卡共享模式虚拟化技术，并且只能使用Alibaba Cloud Linux 2系统，性能优于默认模式。 ■ 如果不选中，则使用默认模式，采用策略路由作为网卡共享模式虚拟化技术，同时兼容Centos 7和Alibaba Cloud Linux 2的系统。 关于Terway模式的IPvlan模式的详细信息，请参见Terway IPvlan模式。 ■ 设置是否选中**NetworkPolicy支持** ■ 只在弹性网卡共享模式下支持选中，默认不选中。 ■ 如果选中，集群支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制。 ■ 如果不选中，集群将不会支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制，这样将不存在网络策略对Kubernetes的API Server产生过大的负载。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。
Service IPv6 CIDR	开启IPv6双栈后，您需要为Service CIDR配置IPv6地址段。配置网段时，请注意： ■ 必须使用ULA地址，地址段范围在fc00::/7内，且地址前缀长度在112~120之间。 ■ 推荐和Service CIDR保持相同的可用地址数量。 关于Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
配置SNAT	创建集群时，默认不开通公网。如果您选择的VPC不具备公网访问能力，选中**为专有网络配置SNAT**后，ACK将为您创建NAT网关并自动配置SNAT规则。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 注意 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。

配置项	描述
SSH登录	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，具体操作请参见通过SSH连接ACK专有版集群的Master节点。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 **说明** 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 **说明** ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。

配置项	描述
标签	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。
自定义镜像	允许您选择一个自定义ECS镜像。选择自定义镜像后，集群所有节点将基于此镜像进行部署 。有关创建自定义镜像操作，请参见使用自定义镜像创建Kubernetes集群。 **说明** ■ 目前自定义镜像的操作系统仅支持CentOS 7.x和Alibaba Cloud Linux 2.x。 ■ 当前只有白名单用户可使用该功能。提交工单申请使用。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影
集群CA	如果选中自定义集群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。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。
资源组	创建的集群将归属于选择的资源组。一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念。更多信息，请参见资源组。

5. 单击**下一步：Master配置**，完成Master节点配置。

配置项	描述
Master实例数量	设置您所需的Master节点数量。目前支持创建3个或者5个Master节点。
实例规格	选择Master节点的实例规格。详情参见实例规格族。
系统盘	默认为Master节点挂载系统盘，支持SSD云盘、ESSD云盘和高效云盘。 **说明** ○ 支持选中**开启云盘备份**以备份云盘数据。 ○ **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。

6. 单击**下一步：节点池配置**，完成节点池配置。
   i. 选择实例。
      - 如果您选择**新增实例**，需要进行以下配置。

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	支持选择多个实例规格。详情请参见实例规格族。
已选规格	呈现选中的规格。
数量	新增Worker实例（ECS实例）的数量。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。

配置项	描述
操作系统	ACK支持的节点操作系统包括： ■ Alibaba Cloud Linux 2.x（ACK默认的操作系统） 选中Alibaba Cloud Linux 2.x操作系统时，您可以进行以下安全加固的配置： ■ 不开启：不对Alibaba Cloud Linux 2.x操作系统进行安全加固。 ■ 等保加固：对Alibaba Cloud Linux 2操作系统进行等保加固。关于等保加固的详细信息，请参见ACK等保加固使用说明。 **注意** 等保加固在保障原生镜像兼容性和性能的基础上进行了等保合规适配，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》。 等保加固会禁止Root用户通过SSH远程登录。您可通过ECS控制台使用VNC方式登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。 ■ CIS加固：对Alibaba Cloud Linux 2.x操作系统进行CIS加固。关于CIS加固的使用说明，请参见ACK CIS加固使用说明。 ■ CentOS 7.x **说明** 暂不支持CentOS 8.x及以上的操作系统。
登录方式	■ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

■ 如果您选择**添加已有实例**，则需要预先在此地域下创建ECS云服务器，然后配置**操作系统**、**登录方式**和**密钥对**，配置描述参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
节点服务端口范围	设置节点服务端口范围。默认端口范围为30000~32767。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

7. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认**不安装**。 ○ 若选择**Nginx Igress**，则默认选中**安装Ingress组件**。关于Nginx Igress的更多使用说明，请参见Ingress高级用法。 ○ 若选择**ALB Ingress（公测）**，则默认安装ALB Ingress Controller组件。关于如何在ACK集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。 NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume和存储管理-CSI。
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**安装node-problem-detector并创建事件中心**，您可以选择是否在日志服务控制台中添加事件中心。更多信息，请参见创建并使用Kubernetes事件中心。

配置项	描述
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

8. 单击**下一步：确认配置**。
9. 选中服务协议并单击**创建集群**。

说明 一个包含多节点的Kubernetes集群的创建时间约为十分钟。

执行结果

- 集群创建成功后，您可以在容器服务管理控制台的**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**详情**，单击**基本信息**和**连接信息**页签，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，可以通过此服务在客户端使用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。
  - **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` 。

    说明 单击**重新绑定域名**，您可以重新绑定访问域名。
- 您可以通过kubectl工具连接集群，执行 `kubectl get node` 查看集群的节点信息。

```
shell@Alicloud:~$ use-k8s-cluster [redacted]

Type "kubectl" to manage your kubenetes cluster c50f6[redacted]

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

4.3.2. 创建Kubernetes托管版集群

Kubernetes托管版集群只需创建Worker节点，Master节点由阿里云容器服务创建并托管，从而您可以更多关注业务本身。本文介绍如何通过容器服务控制台创建Kubernetes托管版集群。

前提条件

登录RAM管理控制台和弹性伸缩控制台开通相应的服务。

> 说明
> 您在使用集群过程中，请注意以下限制：
>
> - 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。
> - ACK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。如果您需要提高配额，请提交工单申请。
>   - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。
>   - 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的ACK集群的网络模式是Flannel时，集群的路由条目最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多路由条目数，您需要对目标VPC提交工单申请提高配额 。
>   - 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：
>   支持创建按量付费和包年包月的ECS实例。
>   实例创建后，您可以通过ECS管理控制台将按量付费转预付费，请参见按量付费转包年包月。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**创建集群**。
4. 在**ACK托管版**页面，完成集群配置。
   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 说明 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
集群规格	选择集群规格，支持**标准版**和**Pro版**。 选中**标准版**创建标准托管集群。
地域	选择集群所在的地域。

配置项	描述
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。
账号全部资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
Kubernetes版本	显示当前ACK支持的Kubernetes版本。
容器运行时	支持**Containerd**、**Docker**和**安全沙箱**。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
IPv6双栈	开启IPv6双栈将创建双栈Kubernetes集群。双栈功能正在公测中，如需使用请提交工单申请。 **注意** ■ 仅1.20.4-aliyun.1及以上版本的Kubernetes集群支持IPv6双栈功能。 ■ Worker节点与控制平面之间的通信使用IPv4地址。 ■ 容器网络插件需选择Terway。 ■ 集群使用的VPC及ECS需要支持IPv6双栈。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。

配置项	描述
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体请参见Terway与Flannel对比。 ■ 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，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。 说明 ■ 在Terway模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。 ■ 网络插件选择Terway时，会使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。
Terway模式	设置**网络插件**为Terway时，需要配置**Terway模式**： ■ 设置是否选中**Pod独占弹性网卡以获得最佳性能** ■ 如果选中，Pod将独占一个专有的弹性网卡。 ■ 如果不选中，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。 说明 当前只有白名单用户可使用上述**Pod独占弹性网卡以获得最佳性能**功能。提交工单申请使用。 ■ 设置是否选中**IPvlan** ■ 只在弹性网卡共享模式支持选中。 ■ 如果选中，采用IPvlan eBPF作为网卡共享模式虚拟化技术，并且只能使用Alibaba Cloud Linux 2系统，性能优于默认模式。 ■ 如果不选中，则使用默认模式，采用策略路由作为网卡共享模式虚拟化技术，同时兼容Centos 7和Alibaba Cloud Linux 2的系统。 关于Terway模式的IPvlan模式的详细信息，请参见Terway IPvlan模式。 ■ 设置是否选中**NetworkPolicy支持** ■ 只在弹性网卡共享模式下支持选中，默认不选中。 ■ 如果选中，集群支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制。 ■ 如果不选中，集群将不会支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制，这样将不存在网络策略对Kubernetes的API Server产生过大的负载。
Pod虚拟交换机	网络插件选择Terway时，您需要为Pod分配IP的虚拟交换机。每个Pod虚拟交换机分别对应一个Worker实例的虚拟交换机，Pod虚拟交换机和Worker实例的虚拟交换机的可用区需一致。

配置项	描述
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 说明 ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。
Service IPv6 CIDR	开启IPv6双栈后，您需要为Service CIDR配置IPv6地址段。配置网段时，请注意： ■ 必须使用ULA地址，地址段范围在fc00::/7内，且地址前缀长度在112~120之间。 ■ 推荐和Service CIDR保持相同的可用地址数量。 关于Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
配置SNAT	创建集群时，默认不开通公网。如果您选择的VPC不具备公网访问能力，选中**为专有网络配置SNAT**后，ACK将为您创建NAT网关并自动配置SNAT规则。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 注意 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 说明 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。

配置项	描述
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 说明 ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。
资源组	创建的集群将归属于选择的资源组。一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念。更多信息，请参见资源组。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。
标签	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。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。 具体操作，请参见自定义集群API Server证书SAN。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影

5. 单击**下一步：节点池配置**，完成节点池配置。

i. 选择实例。

■ 如果您选择**新增实例**，需要进行以下配置。

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。

配置项	描述
实例规格	支持选择多个实例规格。详情请参见实例规格族。
已选规格	呈现选中的规格。
数量	新增Worker实例（ECS实例）的数量。 **说明** 建议创建时最少选择两台，创建好之后，可以根据您的实际情况加减节点。节点太少，或者规格太低，会影响集群组件的运行。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。 ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。

配置项	描述
操作系统	ACK支持的节点操作系统包括： ■ Alibaba Cloud Linux 2.x（ACK默认的操作系统） 选中Alibaba Cloud Linux 2.x操作系统时，您可以进行以下安全加固的配置： ■ 不开启：不对Alibaba Cloud Linux 2.x操作系统进行安全加固。 ■ 等保加固：对Alibaba Cloud Linux 2操作系统进行等保加固。关于等保加固的详细信息，请参见ACK等保加固使用说明。 **注意** 等保加固在保障原生镜像兼容性和性能的基础上进行了等保合规适配，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》。 等保加固会禁止Root用户通过SSH远程登录。您可通过ECS控制台使用VNC方式登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。 ■ CIS加固：对Alibaba Cloud Linux 2.x操作系统进行CIS加固。关于CIS加固的使用说明，请参见ACK CIS加固使用说明。 ■ CentOS 7.x **说明** 暂不支持CentOS 8.x及以上的操作系统。
登录方式	■ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

■ 如果您选择**添加已有实例**，则需要预先在此地域下创建ECS云服务器，然后配置**操作系统**、**登录方式**和**密钥对**，配置描述请参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义镜像	允许您选择一个自定义ECS镜像。选择自定义镜像后，集群所有节点将基于此镜像进行部署 。有关创建自定义镜像操作，请参见使用自定义镜像创建Kubernetes集群。 **说明** ■ 目前自定义镜像的操作系统仅支持CentOS 7.x和Alibaba Cloud Linux 2.x。 ■ 当前只有白名单用户可使用该功能。提交工单申请使用。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

6. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认**不安装**。 ○ 若选择**Nginx Igress**，则默认选中**安装Ingress组件**。关于Nginx Igress的更多使用说明，请参见Ingress高级用法。 ○ 若选择**ALB Ingress（公测）**，则默认安装ALB Ingress Controller组件。关于如何在ACK集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。 NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume和存储管理-CSI。

配置项	描述
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。
报警配置	选中**使用默认报警模板配置报警**，开启默认报警规则。详细介绍，请参见容器服务报警管理。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**创建Ingress Dashboard**，您可以选择是否在日志服务控制台中**创建Ingress Dashboard**。更多信息，请参见Ingress访问日志分析与监控。 默认选中**安装node-problem-detector并创建事件中心**，您可以选择是否在日志服务控制台中添加事件中心。更多信息，请参见创建并使用Kubernetes事件中心。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

7. 单击**下一步：确认配置**。
8. 选中**服务协议**并单击**创建集群**。

说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

执行结果

- 集群创建成功后，您可以在容器服务管理控制台的**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**详情**，单击**基本信息**和**连接信息**页签，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。
    绑定EIP和**解绑EIP**功能仅支持托管版Kubernetes集群。
    - 绑定EIP：您可以选择在已有EIP列表中绑定EIP或者新建EIP。
      绑定EIP操作为导致API Server短暂重启，请避免在此期间操作集群。
    - 解绑EIP：解绑EIP后您将无法通过公网访问API Server。
      解绑EIP操作会导致API Server短暂重启，请避免在此期间操作集群。
  - **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。
  - **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` 。

    说明 单击**重新绑定域名**，您可以重新绑定访问域名。

- 您可以通过kubectl工具连接集群，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    <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:~$
```

4.3.3. 使用自定义镜像创建Kubernetes集群

随着云原生时代的到来，用户应用、业务上云的需求也越来越多，不同的业务场景对容器平台的需求也不尽相同，其中一个非常重要的需求就是使用自定义镜像创建Kubernetes集群。本文主要为您介绍如何使用自定义镜像创建Kubernetes集群。

前提条件

确定要使用自定义镜像前，您需要了解容器服务对于自定义镜像的一些要求。

- 基础镜像推荐使用容器服务使用的最新的基础镜像，因为该镜像不仅可以满足部署Kubernetes集群的需求，同时也是经过容器服务团队严格测试的镜像。自定义镜像需要满足如下几点：
  - 需要满足阿里云cloud-init的要求。更多信息，请参见安装cloud-init。
  - 需要开启sshd server，且使用默认端口22。
  - 使用阿里云NTP Server进行NTP时间同步。
- 若需使用自定义镜像请提交工单申请。

背景信息

容器服务支持您使用自定义镜像创建Kubernetes集群，但在制作打包自定义镜像时，往往会遇到以下情况：

- 人工操作步骤，效率低。
- 镜像变更历史记录缺失，不便于故障定位。
- 无法对自定义镜像进行校验并判断是否符合Kubernetes集群节点要求。

基于以上痛点，容器服务团队开源了ack-image-builder项目帮助您快速制作符合Kubernetes集群节点要求的自定义镜像。
ACK-image-builder项目基于开源工具HashiCorp Packer，提供默认配置模板和校验脚本。

使用ACK-image-builder项目创建Kubernetes集群自定义节点镜像的步骤如下。

操作步骤

1. 安装Packer。

   从官方下载页面选择操作系统对应的软件版本，并按照安装说明文档安装和验证Packer。
   执行如下命令，显示结果如下时，说明Packer已安装成功。

   ```
   packer version
   ```

   ```
   Packer v1.4.1
   ```

2. 定义Packer模板。

   使用Packer创建自定义镜像时，需要创建一个JSON格式的模板文件。在该模板文件中，您需要指定创建自定义镜像的Alicloud Image Builder（生成器）和Provisioners（配置器）。

```
{
 "variables": {
  "region": "cn-hangzhou",
  "image_name": "test_image{{timestamp}}",
  "source_image": "centos_7_06_64_20G_alibase_20190711.vhd",
  "instance_type": "ecs.n1.large",
  "access_key": "{{env `ALICLOUD_ACCESS_KEY`}}",
  "secret_key": "{{env `ALICLOUD_SECRET_KEY`}}"
 },
 "builders": [
  {
   "type": "alicloud-ecs",
   "access_key": "{{user `access_key`}}",
   "secret_key": "{{user `secret_key`}}",
   "region": "{{user `region`}}",
   "image_name": "{{user `image_name`}}",
   "source_image": "{{user `source_image`}}",
   "ssh_username": "root",
   "instance_type": "{{user `instance_type`}}",
   "io_optimized": "true"
  }
 ],
 "provisioners": [
  {
   "type": "shell",
   "scripts": [
    "scripts/updateKernel.sh",
    "scripts/reboot.sh",
    "scripts/cleanUpKerneles.sh",
    "config/default.sh",
    "scripts/updateDNS.sh",
    "scripts/verify.sh"
   ],
   "expect_disconnect": true
  }
 ]
}
```

参数	描述
access_key	您的AccessKey ID。
secret_key	您的AccessKey Secret。
region	创建自定义镜像时使用临时资源的地域。
image_name	自定义镜像的名称。
source_image	基础镜像的名称，可以从阿里云公共镜像列表获得。
instance_type	创建自定义镜像时生成的临时实例的类型。
provisioners	创建自定义镜像时使用的Packer配置器类型。

3. 创建子账号并生成AccessKey。

   制作自定义镜像的权限要求较大，一般建议用户创建子账户并授权Packer需要的对应RAM

Policy并AccessKey。

4. 导入AccessKey信息并制作自定义镜像。
   i. 执行如下命令，导入AccessKey信息。

```
export ALICLOUD_ACCESS_KEY=XXXXXX
export ALICLOUD_SECRET_KEY=XXXXXX
```

ii. 执行如下命令，制作自定义镜像。

```
packer build alicloud.json
```

```
alicloud-ecs output will be in this color.
==> alicloud-ecs: Prevalidating source region and copied regions...
==> alicloud-ecs: Prevalidating image name...
  alicloud-ecs: Found image ID: centos_7_06_64_20G_alibase_20190711.vhd
==> alicloud-ecs: Creating temporary keypair: xxxxxx
==> alicloud-ecs: Creating vpc...
  alicloud-ecs: Created vpc: xxxxxx
==> alicloud-ecs: Creating vswitch...
  alicloud-ecs: Created vswitch: xxxxxx
==> alicloud-ecs: Creating security group...
  alicloud-ecs: Created security group: xxxxxx
==> alicloud-ecs: Creating instance...
  alicloud-ecs: Created instance: xxxxxx
==> alicloud-ecs: Allocating eip...
  alicloud-ecs: Allocated eip: xxxxxx
  alicloud-ecs: Attach keypair xxxxxx to instance: xxxxxx
==> alicloud-ecs: Starting instance: xxxxxx
==> alicloud-ecs: Using ssh communicator to connect: 47.111.127.54
==> alicloud-ecs: Waiting for SSH to become available...
==> alicloud-ecs: Connected to SSH!
==> alicloud-ecs: Provisioning with shell script: scripts/verify.sh
  alicloud-ecs: [20190726 11:04:10]: Check if kernel version >= 3.10. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if systemd version >= 219. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if sshd is running and listen on port 22. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if cloud-init is installed. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if wget is installed. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if curl is installed. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if kubeadm is cleaned up. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if kubelet is cleaned up. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if kubectl is cleaned up. Verify Passed!
  alicloud-ecs: [20190726 11:04:10]: Check if kubernetes-cni is cleaned up. Verify Passed!
==> alicloud-ecs: Stopping instance: xxxxxx
==> alicloud-ecs: Waiting instance stopped: xxxxxx
==> alicloud-ecs: Creating image: test_image1564110199
  alicloud-ecs: Detach keypair xxxxxx from instance: xxxxxxx
==> alicloud-ecs: Cleaning up 'EIP'
==> alicloud-ecs: Cleaning up 'instance'
==> alicloud-ecs: Cleaning up 'security group'
==> alicloud-ecs: Cleaning up 'vSwitch'
==> alicloud-ecs: Cleaning up 'VPC'
==> alicloud-ecs: Deleting temporary keypair...
Build 'alicloud-ecs' finished.
==> Builds finished. The artifacts of successful builds are:
--> alicloud-ecs: Alicloud images were created:
cn-hangzhou: m-bp1aifbnupnaktj00q7s
```

其中scripts/verify.sh为对检查项的校验部分。

5. 使用自定义镜像创建Kubernetes集群。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击页面右上角的**创建集群**。

iv. 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**。请参见创建Kubernetes专有版集群。

v. 在**ACK托管版**页签，配置使用自定义镜像创建集群的参数。关于创建集群的参数说明，请参见创建ACK Pro版集群。

a. 创建需要的相关基础信息后，单击下方的**显示高级选项**，在**自定义镜像**中单击**选择**。

b. 在**选择自定义镜像**页面，在目标镜像右侧单击**使用**。

vi. 配置完成后，单击**创建集群**。

待集群创建完成后，该集群就是使用了自定义镜像，后续的扩容等均会使用该镜像。

4.3.4. 通过生成OpenAPI参数创建集群

创建集群时，如果由于请求参数组合不正确导致集群创建失败，您可以通过生成OpenAPI参数，为您创建集群时所请求的参数生成正确的参数组合。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在弹出的**选择集群模板**中，选择需要创建的集群类型，并单击**创建**。

   默认进入集群配置页面。配置信息，请参见创建Kubernetes专有版集群。
5. 配置集群参数。

   您必须配置了以下参数，才能生成OpenAPI参数。

   - 如果是创建专有版Kubernetes集群，必选参数，请参见创建Kubernetes专有版集群。
   - 如果是创建托管版Kubernetes集群，必选参数，请参见创建Kubernetes托管版集群。
6. 配置完成后，单击**生成OpenAPI请求参数**。



OpenAPI请求参数对话框中展示您创建集群时所请求的参数的组合。



7. 单击**复制**，您可以根据参数组合创建集群。

   创建集群时，您可以使用OpenAPI创建集群，请参见创建Kubernetes专有版集群和创建Kubernetes托管版集群。
   您也可以通过CLI创建集群，请参见创建集群。

4.4. 连接集群

4.4.1. 通过kubectl工具连接集群

除了通过容器服务控制台来管理集群之外，您还可以通过Kubernetes命令行工具kubectl来管理集群以及应用。本文介绍如何通过kubectl客户端连接ACK集群。

背景信息

Kubectl是标准的Kubernetes命令行管理工具，通过Kubectl您可以连接和管理阿里云容器服ACK的所有集群类型，包括ACK专有版、ACK托管版（包括ACK标准版和ACK Pro版）以及ASK。关于kubectl命令行的更多信息，请参见kubectl。

步骤一：安装kubectl工具

1. 根据需要，确定安装kubectl的客户端机器。

   您可以通过公网或私网来连接集群，客户端机器的选择建议如下所示：

- 通过公网连接
  当集群开启了API Server公网访问能力（即集群已通过阿里云弹性公网IP EIP暴露了API Server）时，您可以通过公网中的任一机器作为客户端来连接集群。关于如何开启API Server的公网访问能力，请参见控制集群API Server的公网访问能力。
- 通过私网连接
  当您只需要通过私网来连接集群时，kubectl客户端机器必须与集群位于同一VPC。

2. 下载kubectl工具，并安装至对应的客户端机器。

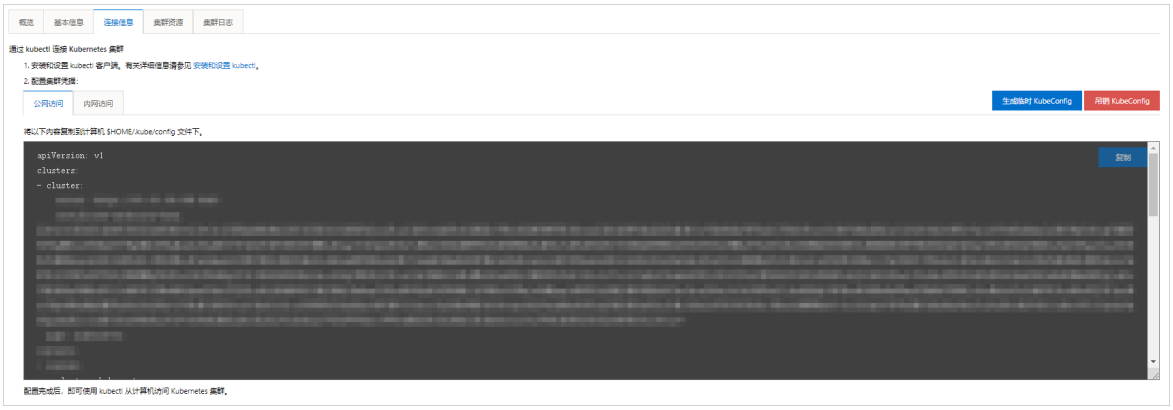
   详细信息，请参见安装和设置kubectl。

步骤二：选择集群凭证类型

ACK集群提供了两种集群凭证（即KubeConfig），分别用于公网访问和私网访问。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面，单击**连接信息**页签，根据需要选择公网或私网访问凭证。

> **说明** 为了实现更灵活的集群连接策略，您还可以生成临时KubeConfig，以控制凭证的有效期。详情请参见生成临时的KubeConfig。



步骤三：配置集群凭证

kubectl工具默认会从客户端机器的*$HOME/.kube*目录下查找名为*config*的文件，该文件用于存储所要管理集群的访问凭证，kubectl会根据该配置文件连接至集群。

1. 选择**公网访问**或**私网访问**页签，单击**复制**。
2. 将复制的集群凭证内容粘贴至*$HOME/.kube*目录下的*config*文件中，保存并退出。

> **说明** 如果*$HOME/*目录下没有*.kube*目录和*config*文件，请自行创建。

如果您使用的是专有版集群并且集群开启了公网SSH登录时，在**连接信息**页签下会显示**SSH**页签。除了参照以上方式连接集群外，您还可以通过SSH登录Master节点，配置集群访问凭证后，即可使用kubectl管理集群。关于如何通过SSH方式连接专有版集群，请参见通过SSH连接ACK专有版集群的Master节点。



步骤四：验证集群连通性

集群凭证配置之后，您可以执行kubectl命令以验证集群的联通性。以查询命名空间为例，执行以下命令。

```
kubectl get namespace
```

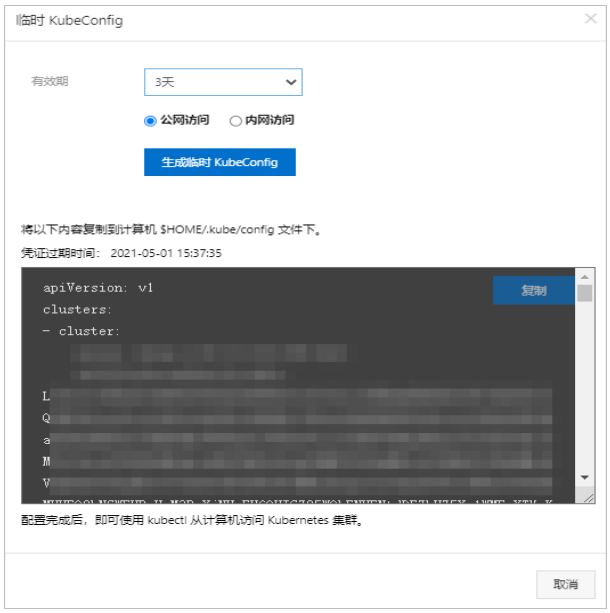
预期输出：

```
NAME              STATUS   AGE
default           Active   4h39m
kube-node-lease   Active   4h39m
kube-public       Active   4h39m
kube-system       Active   4h39m
```

生成临时的KubeConfig

为了实现更灵活的集群连接策略，您可以生成临时集群凭证（KubeConfig），并设置凭证使用的有效期。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面，单击**连接信息**页签，然后单击**生成临时KubeConfig**。
5. 在**临时KubeConfig**对话框中，设置临时凭证的有效期及访问集群的方式（包括公网访问和内网访问）。



   - 如果您使用公网访问，请选择**公网访问**，单击**生成临时KubeConfig**，然后单击**复制**，将内容粘贴到本地计算机的 `$HOME/.kube/config` 文件下。
   - 如果您使用内网访问，请选择**内网访问**，单击**生成临时KubeConfig**，然后单击**复制**，将内容粘贴到本地计算机的 `$HOME/.kube/config` 文件下。

   之后便可通过临时凭证连接至集群。当临时凭证过期后，将无法连接集群。

吊销KubeConfig

您可以吊销集群的KubeConfig，以保障集群的安全。具体操作，请参见吊销集群的KubeConfig凭证。

相关文档

- 获取集群kubeconfig接口

4.4.2. 在CloudShell上通过kubectl管理Kubernetes集群

CloudShell是阿里云推出的云命令行工具，您可以使用CloudShell工具在任意浏览器上运行CloudShell命令管理阿里云资源。本文为您介绍如何在容器服务ACK控制台上利用CloudShell通过kubectl管理集群。

前提条件

CloudShell仅支持通过公网连接集群，因此待连接集群的API Server必须已开启公网访问能力。具体操作，请参见控制集群API Server的公网访问能力。

背景信息

如果您希望通过kubectl工具管理容器服务ACK中的Kubernetes集群，您可下载kubectl到客户端（具体操作，请参见通过kubectl工具连接集群）。您也可以在容器服ACK控制台上打开CloudShell，系统会自动加载集群的KubeConfig文件，您无需任何安装或配置操作，可以直接在CloudShell中通过kubectl管理集群。

Cloud Shell是网页版命令行工具。您可以在任意浏览器上运行Cloud Shell管理阿里云资源。在Cloud Shell启动时会自动为您分配一台Linux虚拟机供您免费使用。在该虚拟机上预装Python、Java、Node.js环境和Aliyun CLI、cURL、SSH、kubectl、Fun、Terraform、Ansible、Vim等多种云管理工具和系统工具，配合Cloud Shell提供的Web IDE，辅助您完成日常的云资源管理。更多信息，请参见什么是云命令行？。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群右侧**操作**列下的**更多 > 通过CloudShell管理集群**。
4. （可选）单击  ，然后单击**挂载存储空间**。

   关联并挂载一个NAS文件系统实例可以持久化存储您的常用脚本和文件，否则您的文件会随实例释放而销毁。可根据您的实际情况单击**创建并绑定**或**暂不创建**。

执行结果

在CloudShell中可通过kubectl工具管理ACK集群。

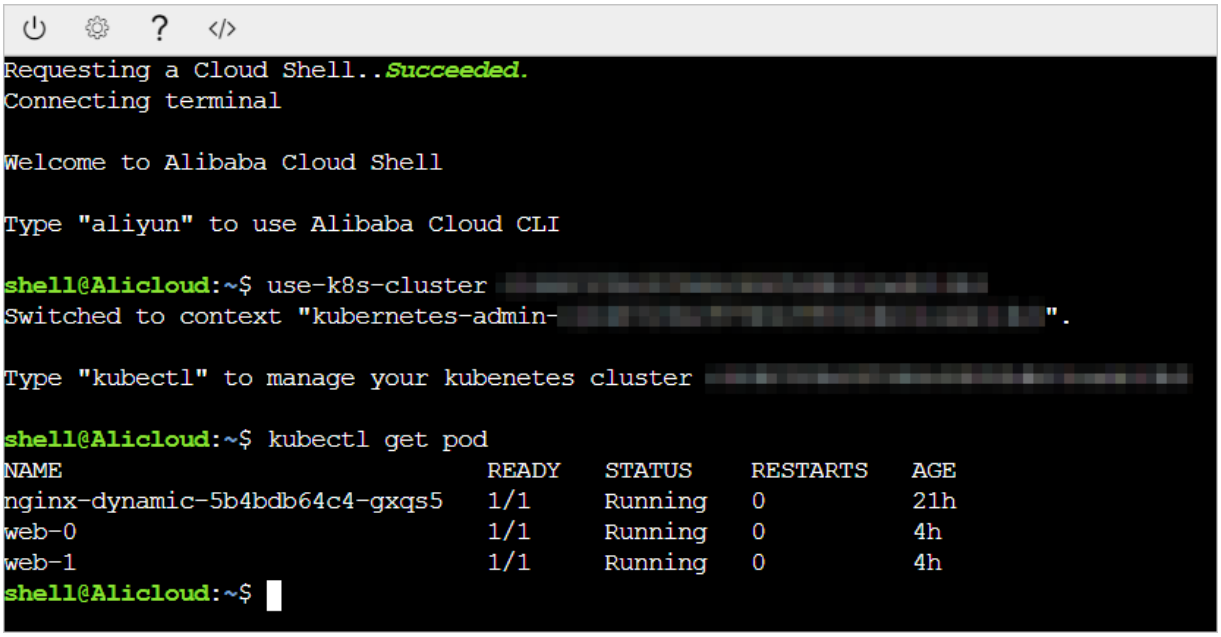
> **说明** 在打开集群关联的CloudShell时，系统会自动加载集群的*kubeconfig*文件。您可以通过kubectl直接管理您的集群。

例如，执行以下命令，查看当前集群的Pod。

```
kubectl get pod
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE
nginx-dynamic-5b4bdb64c4-gxqs5  1/1    Running  0     178m
web-0                               1/1    Running  0     34d
web-1                               1/1    Running  0     34d
```



相关文档

- 使用云命令行
- 云命令行支持工具列表
- 上传和下载文件

4.4.3. 集群访问控制

4.4.3.1. 配置集群API Server的访问控制策略

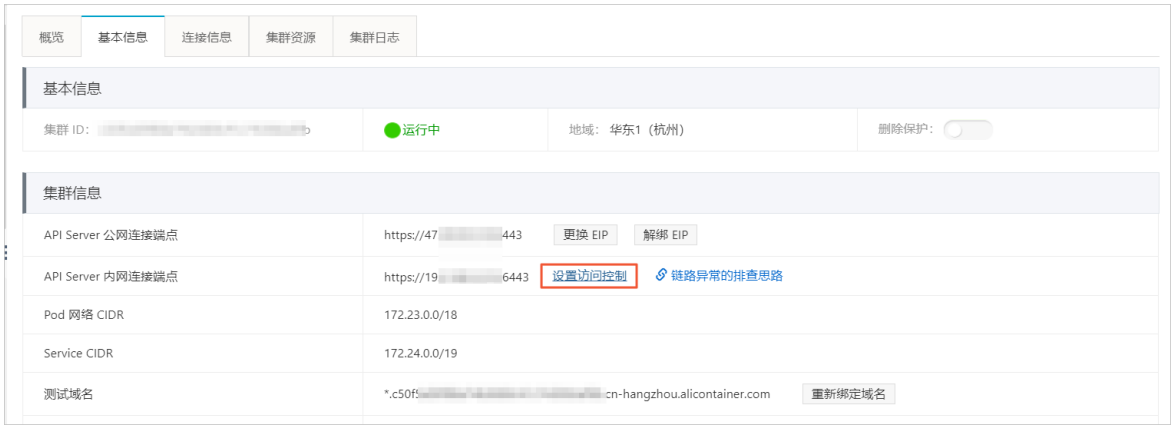
ACK集群创建时会为API Server自动创建一个私网SLB实例，作为集群API Server的内网连接端点。如果您需要精准地访问并控制API Server，您可以对该私网SLB实例的6443端口监听（Listener）配置访问控制，即设置访问白名单或者黑名单。本文介绍如何通过配置私网SLB的监听实现API Server的访问控制。

背景信息

负载均衡提供监听级别的访问控制。您可以在创建监听时配置访问控制，也可以在监听创建后修改或重新配置访问控制。有关更多详情，请参见访问控制概述。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面中，单击**基本信息**页签。
5. 在**基本信息**页签中的**集群信息**区域，找到并单击**API Server内网连接端点**右侧的**设置访问控制**。



6. 在负载均衡控制台的**访问控制设置**面板中，打开**启用访问控制**开关，然后设置访问控制白名单或黑名单，选择访问控制策略组。

   在开启访问控制前，您需要创建访问控制策略。关于如何创建访问控制策略组，请参见创建访问控制策略组。关于如何启用访问控制，请参见开启访问控制。

   > **注意**
   > - 当配置白名单时，需将容器服务ACK管控的IP段100.104.0.0/16和节点所在的VSwitch网段加入白名单中。
   > - 当配置黑名单时，勿将容器服务ACK管控的IP段100.104.0.0/16和节点所在的VSwitch网段加入黑名单中。

7. 单击**确定**。

4.4.3.2. 控制集群API Server的公网访问能力

您可以使用弹性公网IP EIP（Elastic IP Address）暴露Kubernetes集群的API Server。使用EIP暴露集群API Server后，集群API Server将具有公网访问能力。您可以在创建集群时和创建集群后绑定EIP。本文介绍如何通过绑定和解绑EIP来控制集群API Server的公网访问能力。

创建集群时绑定EIP

您可以在创建集群的时候通过选中**使用EIP暴露API Server**获得从公网访问集群API Server的能力。创建集群的详细步骤请参见创建Kubernetes托管版集群。

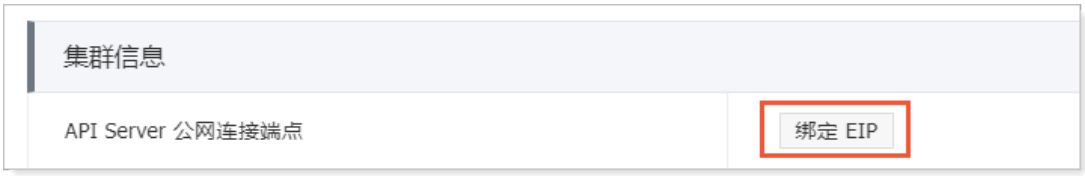


创建集群后绑定EIP

如果您创建集群时未选择为API Server开放公网，您可以通过以下步骤为集群绑定EIP。

> **注意** 目前只有ACK标准版、ACK Pro版和ASK集群支持在创建集群后绑定EIP。

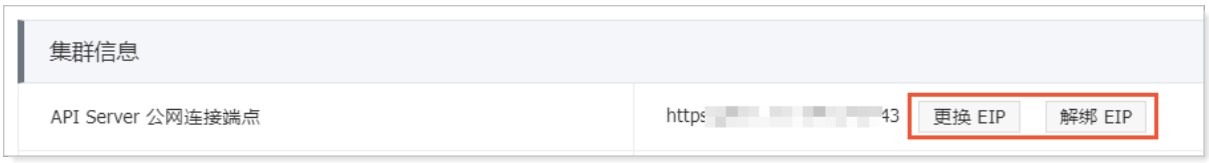
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群信息页面，单击**基本信息**页签，然后在**集群信息**区域，单击**绑定EIP**。



5. 选择已有EIP，单击**确定**。
   绑定完成后，API Server公网连接端点将会显示公网IP地址。

解绑或更换EIP

绑定EIP后，您还可以更换EIP或者解绑EIP。解绑EIP后，集群的API Server将失去公网访问能力。



> **注意** 目前只有ACK标准版、ACK Pro版和ASK集群支持**更换EIP**和**解绑EIP**功能。

4.4.3.3. 吊销集群的KubeConfig凭证

在常见的多租场景下，容器服务会为不同角色的用户签发带有其身份信息的KubeConfig凭证用于连接集群。当企业内部员工离职或是某签发KubeConfig遭到疑似泄露等情况发生时，吊销该集群的KubeConfig可有效保障集群的安全。本文主要介绍在容器服务中如何吊销已经分发的用户KubeConfig凭证。

前提条件

- 如果您需要吊销的是Serverless Kubernetes集群的KubeConfig凭证，请选择2019年9月6号以后创建的集群。
- 如果您需要吊销的是专有版或者托管版Kubernetes集群的KubeConfig凭证：
  - 当前用户是主账号，则2019年10月15日之前创建的Kubernetes集群无法吊销主账号KubeConfig凭证。
  - 当前用户是子账号，则可以对管理范围内的Kuberneres集群吊销KubeConfig凭证。

吊销KubeConfig凭证包含以下两种场景：

- 当前用户可以吊销自身的KubeConfig凭证。
- 主账号可以吊销其子账号的KubeConfig凭证。

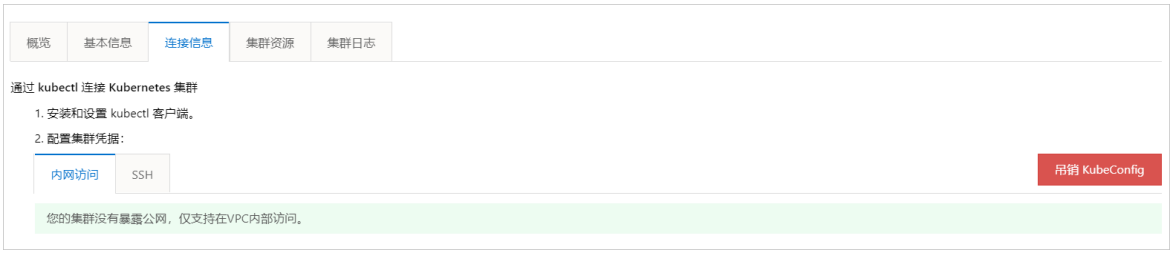
吊销自身KubeConfig凭证

> **注意** 吊销KubeConfig后，用户就无法使用原KuberConfig连接集群，请谨慎操作。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

   > **说明**
   > 如果您使用主账号登录，请选择2019年10月15日及以后创建的集群。
   > 如果您需要吊销的是Serverless Kubernetes集群的KubeConfig凭证，请选择2019年9月6号以后创建的集群。

4. 在**集群信息**页面，单击**连接信息**页签，单击**吊销 KubeConfig**。



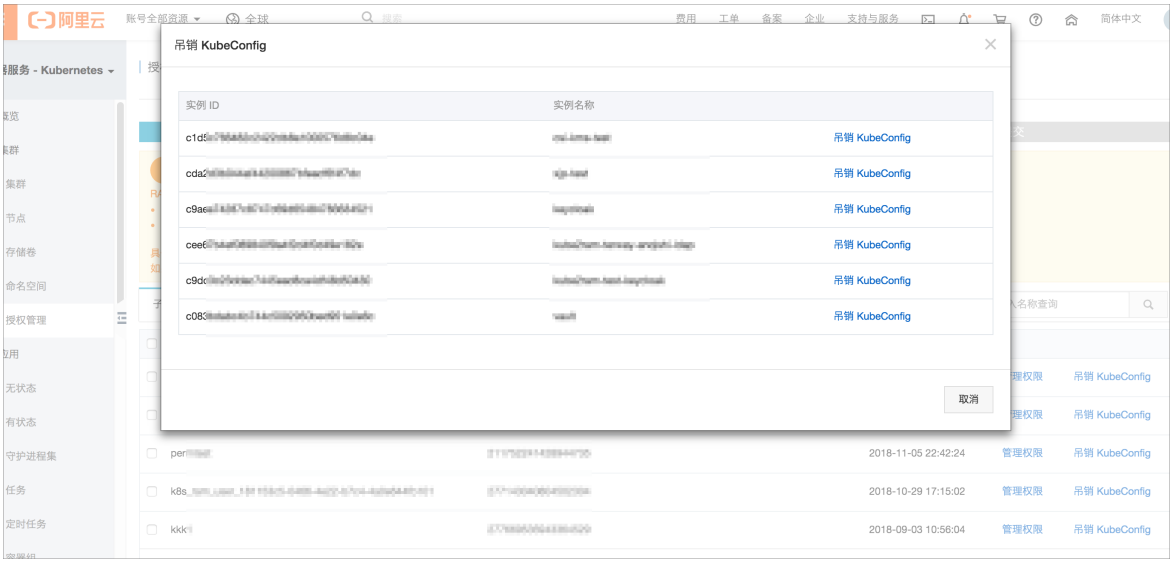
5. 在弹出的对话框中单击**确定**。

   吊销当前账号下该集群的KubeConfig凭证。此时，系统会自动为您分配新的KubeConfig凭证。

主账号吊销其子账号的KubeConfig凭证

1. 使用主账号登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**授权管理**。
3. 在**选择子账号**配置项页面的**子账号**页签，在需要吊销的子账号右侧单击**吊销KubeConfig**。

   此时会弹出该子账号名下所有的集群ID。



4. 找到需要吊销的集群所对应的集群ID，单击**吊销KubeConfig**。
5. 在弹出的对话框中单击**确定**。

4.4.4. 更多连接方式

4.4.4.1. 通过SSH连接ACK专有版集群的Master节点

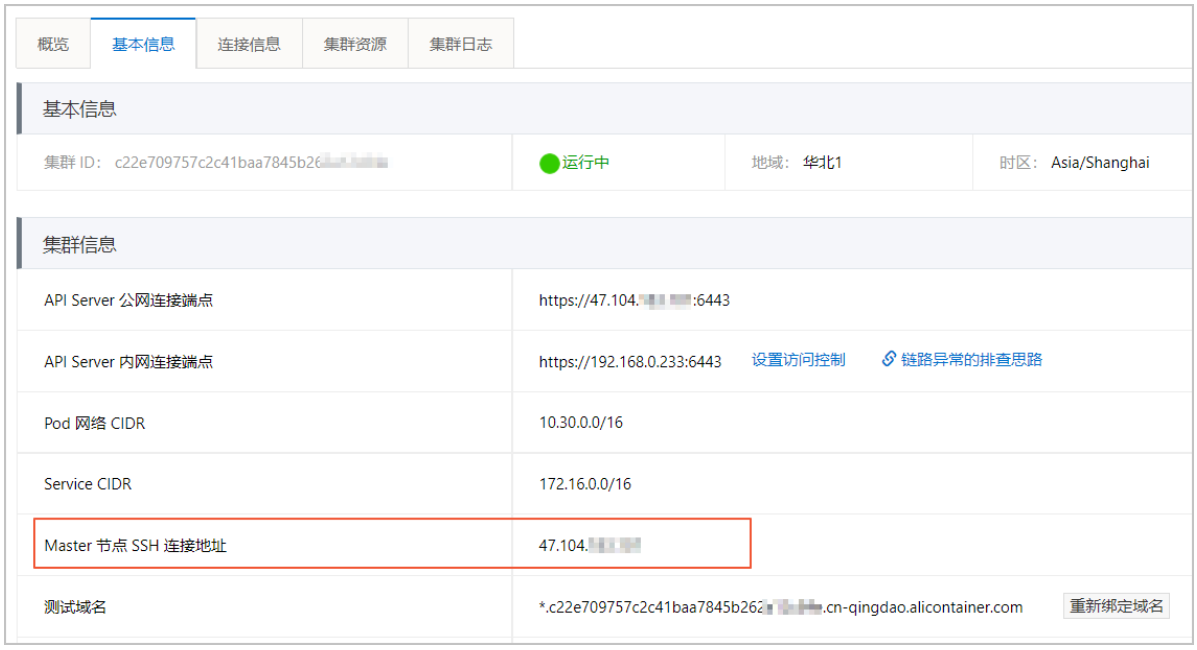
由于ACK专有版集群的Master节点由您自行创建并维护，因此为了方便您的维护管理，容器服务ACK支持通过SSH方式连接专有版集群的Master节点。本文介绍如何为专有版集群开通SSH公网登录，以及如何通过SSH方式连接专有版集群的Master节点。

前提条件

ACK专有版集群的API Server已绑定EIP。如未绑定，请参见控制集群API Server的公网访问能力。

背景信息

- 创建专有版集群时，如果您选中了**开放公网SSH登录**，ACK会自动为您生成Master节点SSH 连接地址，在集群的**基本信息**页签下可以查看。此时您可以直接使用该地址登录专有版集群的Master节点。



- 如果创建时没有开放公网SSH登录，您需要为API Server的SLB手动添加SSH（22）端口监听，之后才可以通过SSH方式登录Master节点。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面，单击**集群资源**页签，找到您的**APIServer 负载均衡（SLB）**，并单击右侧实例ID链接。
5. 在负载均衡SLB左侧导航栏，选择**传统型负载均衡CLB（原SLB） > 实例管理**，单击**监听**页签，然后单击**添加监听**。

   以下为简要配置步骤，更详细的操作，请参见添加TCP监听。

   i. 选择负载均衡协议为**TCP**，设置**监听端口**为22（即SSH所用端口），单击**下一步**。

   ii. 选择**默认服务器组**，在**已添加服务器**区域为所有的Master节点配置端口为22，根据负载均衡的需求设置不同节点的权重，然后单击**下一步**。

   iii. 在**健康检查**页面，单击**下一步**。

   iv. 在**配置审核**页面，单击**提交**。

   添加成功后，会在下面的**监听**页签中显示新创建的名为tcp_22的监听。



6. 在SLB实例的**实例详情**页签，查看SLB的弹性（即公网）地址。
   该地址便成为专有版集群对外的Master节点SSH地址。



7. 通过SSH登录Master节点SSH地址。

   在创建集群时，有两种登录方式可供选择：设置密钥和设置密码（详情请参见创建Kubernetes专有版集群）。请根据您的实际设置选择以下登录方式：

   - 密码登录：用户名默认为root，密码为您创建集群时设置的密码。
   - 密钥登录：请参见通过密钥认证登录Linux实例。

4.4.4.2. 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。

前提条件

- 已创建Kubernetes托管版集群。具体操作，请参见创建Kubernetes托管版集群。
- 已通过Kubectl连接到Kubernetes托管版集群。具体操作，请参见通过Kubectl连接Kubernetes集群。

操作步骤

1. 执行以下命令获取API Server内网连接端点。

   ```
   kubectl get endpoints kubernetes
   ```

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

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

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

3. 执行以下命令，部署公网Endpoints。

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

4. 执行以下命令，获取公网SLB地址，即EXTERNAL-IP。

```
kubectl get service <name>
```

> **说明** 此处的<name>与步骤2*kubernetes-public-service.yaml*文件中的name的值保持一致，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 --namespace=<namespace> <name> -o jsonpath={.data.token} | base64 -d
```

> **说明** 此处的<namespace>与步骤5中的<namespace>保持一致。<name>是步骤5中找到ServiceAccount对应的Secret名称。

7. 执行以下命令，访问托管版Kubernetes集群。

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

> **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。

执行结果

执行命令后，出现以下信息，表明连接成功。



4.5. 管理集群

4.5.1. 查看集群信息

容器服务Kubernetes版集群提供集群概览展示页，包括应用状态、组件状态和资源监控等功能板块，方便您快速了解集群的健康状态信息。同时您还可以查看集群的基本信息、连接信息、集群资源、集群日志以及集群时区。

查看集群概览

您可以查看应用状态、节点状态、资源监控等信息。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 单击**概览**页签，进入Kubernetes集群概览页面。
5. 选择所需的命名空间，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、有状态副本集等应用状态的示意图，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。
   - **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。

查看基本信息

单击**基本信息**页签，查看集群ID、地域、API Server连接端点以及其他网络信息。

查看连接信息

单击**连接信息**页签，您可以获取公网和内网环境下KubeConfig文件的配置内容，用于配置通过Kubectl客户端访问集群。

查看集群资源

单击**集群资源**页签，您可以查看集群所使用的相关云资源。通过单击相应的资源ID可以跳转至对应的控制台。

> **注意** 这些资源由阿里云容器服务进行管理，请不要删除或自行修改，以避免导致集群异常，影响集群内应用的正常运行。

查看集群日志

您有两种方式查看集群日志：

- 方式一：**集群信息**入口
  单击**集群日志**页签，您可以查看集群日志。
- 方式二：**运维管理**入口
  在集群详情左侧导航栏，选择**运维管理 > 日志中心**，在**集群日志**页面查看集群日志。

查看集群时区

您可以分别查看集群的托管侧时区和Worker节点时区。

- 通过kubectl连接集群，执行以下命令查看集群的托管侧时区。

  ```
  kubectl get configmap -n kube-system ack-cluster-profile -o yaml | grep timezone
  ```

  输出示例：`timezone: Asia/Shanghai`
- 通过SSH工具登录Worker节点，执行以下命令查看节点时区。

  ```
  ls -l /etc/localtime
  ```

  输出示例：`lrwxrwxrwx 1 root root 30 Sep 30 18:44 /etc/localtime -> /usr/share/zoneinfo/Asia/Shanghai`

4.5.2. 集群证书

4.5.2.1. 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5年。

注意事项

- 在证书更新过程中集群kube-apiserver，kube-controller-manager，kube-scheduler等系统组件将有短暂的重启过程，如果您的服务逻辑强依赖API Server等系统组件，请您在证书更新前确认服务在更新过程中的可用性。
- 建议您在非业务高峰期执行更新操作。

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/pk*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

Master节点

证书或conf名称	路径	证书有效期
• *apiserver.crt* • *apiserver.key*	*/etc/kubernetes/pki*	初始证书有效期10年，更新证书后有效期延长5年。
• *apiserver-kubelet-client.crt* • *apiserver-kubelet-client.key*	*/etc/kubernetes/pki*	初始证书有效期10年，更新证书后有效期延长5年。
• *front-proxy-client.crt* • *front-proxy-client.key*	*/etc/kubernetes/pki*	初始证书有效期10年，更新证书后有效期延长5年。
• *dashboard.crt* • *dashboard.key*	*/etc/kubernetes/pki/dashboard*	初始证书有效期10年，更新证书后有效期延长5年。

证书或conf名称	路径	证书有效期
• *kubelet.crt* • *kubelet.key* **说明** • 如果不存在*kubelet.key*，则无需更新。 • *kubelet.crt*和*kubelet.key*过期不影响集群使用。	*/var/lib/kubelet/pki* **说明** 如果不存在数据，则无需更新。	初始证书有效期10年，更新证书后有效期延长5年。
admin.conf	*/etc/kubernetes*	初始证书有效期10年，更新证书后有效期延长5年。
kube.conf	*/etc/kubernetes*	初始证书有效期10年，更新证书后有效期延长5年。
controller-manager.conf	*/etc/kubernetes*	初始证书有效期10年，更新证书后有效期延长5年。
scheduler.conf	*/etc/kubernetes*	初始证书有效期10年，更新证书后有效期延长5年。
kubelet.conf	*/etc/kubernetes*	初始证书有效期10年，更新证书后有效期延长5年。
config	*~/.kube/*	初始证书有效期10年，更新证书后有效期延长5年。
• *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。	初始证书有效期1年，即将过期时自动更新证书，有效期延长1年。

Worker节点

证书或conf名称	路径	证书有效期
• *kubelet.crt* • *kubelet.key* **说明** • 如果不存在*kubelet.key*，则无需更新。 • *kubelet.crt*和*kubelet.key*过期不影响集群使用。	*/var/lib/kubelet/pki* **说明** 如果不存在数据，则无需更新。	初始证书有效期10年，更新证书后有效期延长5年。
• *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。	初始证书有效期1年，即将过期时自动更新证书，有效期延长1年。
kubelet.conf	*/etc/kubernetes*	初始证书有效期10年，更新证书后有效期延长5年。

4.5.2.2. 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

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes专有版集群或创建Kubernetes托管版集群。

  说明 您无需担心托管版集群中Master节点证书过期情况，容器服务ACK会自动更新托管版集群中Master节点的证书。

- 已通过kubectl连接集群。具体操作，请参见通过kubectl连接Kubernetes集群。

控制台更新所有节点证书

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
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. 在**更新证书**页面单击**更新证书**。

5. 在**提示**页面单击**确定**。
   成功更新集群证书后，您可以看到以下内容：
   - 在**更新证书**页面，显示**更新成功**。
   - 在**集群列表**页面，目标集群无**更新证书**提示。

命令行自动更新所有节点证书

在集群任意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
```

结果验证：

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COMPLETIONS均为*1*，Worker节点对应的COMPLETIONS为集群Worker节点数时，所有证书完成更新。

   ```
   kubectl -n kube-system get job
   ```

   ```
   [root@ ~]# kubectl get job -nkube-system
   NAME                                COMPLETIONS   DURATION   AGE
   aliyun-cert-renew-master-1          1/1           46s        4m49s
   aliyun-cert-renew-master-2          1/1           28s        4m19s
   aliyun-cert-renew-master-3          1/1           26s        3m48s
   aliyun-cert-renew-worker            6/6           46s        3m18s
   ingress-nginx-admission-create      1/1           29s        3d
   ingress-nginx-admission-patch       1/1           43s        3d
   kube-eventer-init-1.5-5e0e7c1-aliyun   1/1           31s        3d
   ```

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：通过命令行
     执行以下命令：

     ```
     kubectl get nodes
     ```

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

- 方法二：通过控制台
  a. 登录容器服务管理控制台。
  b. 在控制台左侧导航栏中，单击**集群**。
  c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  d. 在集群管理页的左侧导航栏中，选择**节点管理 > 节点**获取Master个数和和对应的名称、IP地址、实例ID。

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COMPLETIONS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@[redacted] ~]# kubectl get job -nkube-system
NAME          COMPLETIONS  DURATION  AGE
cert-job-2    1/1          46s       22m
cert-job-3    1/1          28s       2m
cert-job-4    1/1          26s       1m
[root@[redacted] ~]#
```

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 tolerations ，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：
>
> i. 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 $taint }}
>       {{- .key }}={{ .value }}:{{ .effect }}{{ "\t" }}
>     {{- end }}
>     {{- "\n" }}
>   {{- end}}
> {{- end}}
> ```
>
> ii. 执行以下命令，查询带有Taint的Worker节点的${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：

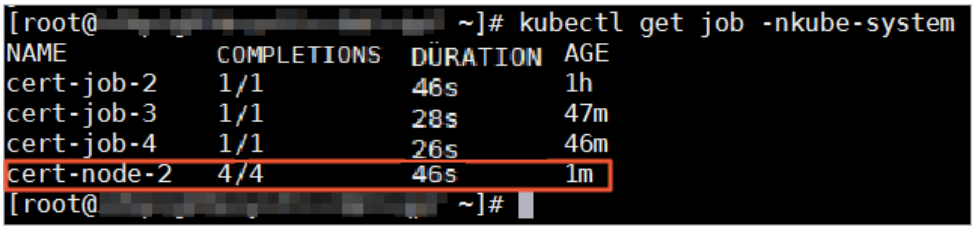
   - ${jobname}为Job的名称，此处设置为*cert-node-2*。
   - ${nodesize}为Worker节点个数，获取方法可参见手动更新Worker节点证书的步骤1。此处请将*nodesize*替换为集群的Worker个数。
   - ${key}为集群的CAKey，此处请将*key*替换为手动更新Worker节点证书步骤2获取到的CAKey。

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COMPLETIONS为集群Worker节点数时，证书完成更新。

   ```
   kubectl get job –nkube-system
   ```



4.5.2.3. 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API Server的通讯都将被禁止，因此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。

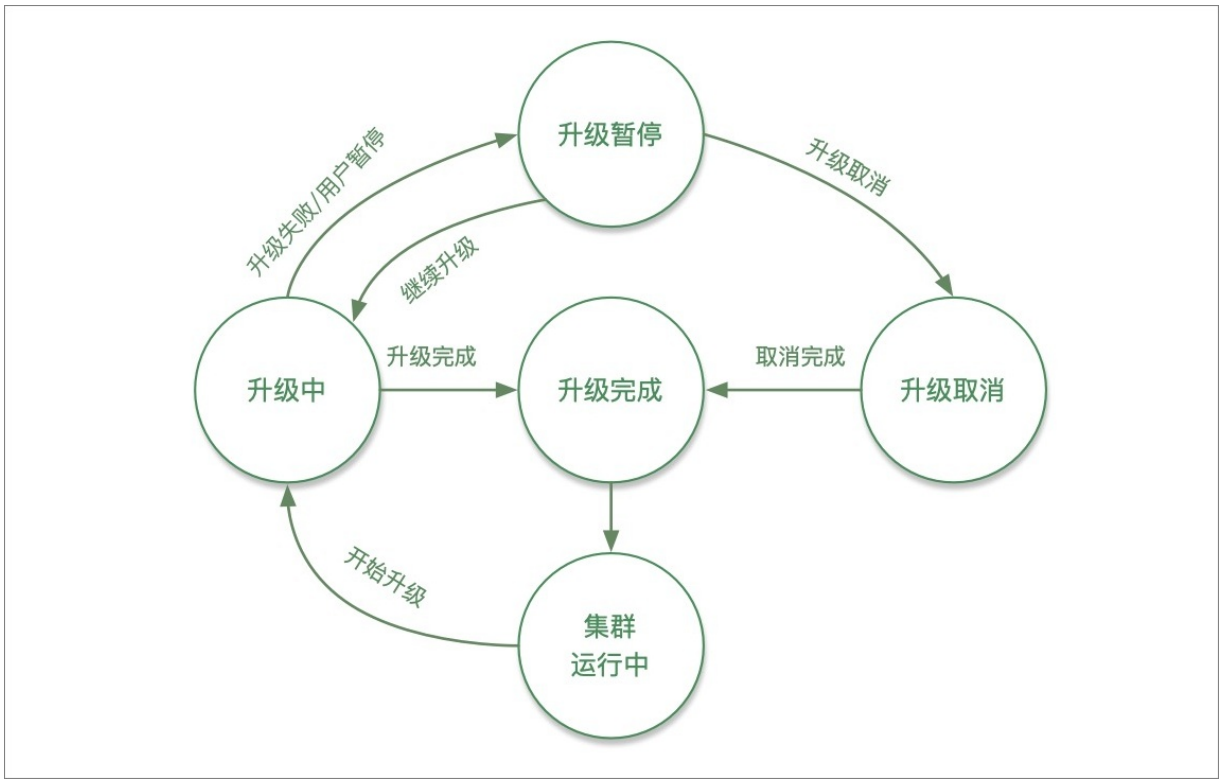
6. 在集群的每个Worker节点，重复上述步骤，完成所有Worker节点已过期的证书更新。

4.5.3. 升级ACK集群K8s版本

您可以通过容器服务管理控制台，可视化升级集群的Kubernetes（K8s）版本。您可以在Kubernetes集群列表页面查看集群的Kubernetes版本，以及当前是否有新的版本可供升级。升级集群的过程包含升级前置检查、升级Master（专有版会展示当前正在升级的Master编号）、升级Node（会展示已经升级的节点数和总节点数）。

功能原理

下面主要为您介绍集群升级过程中的相关功能及实现原理。



- 集群升级策略
  集群升级策略定义了您将使用怎样的策略对集群进行升级。目前默认策略为分批升级。分批升级会在**升级Node**阶段对集群内的节点进行分批升级。其具体策略为：
  - 第一批升级的节点数为1，后续的批次以2的幂数进行增长。暂停后重新恢复升级的第一批次为1，后续也是以2的幂数进行增长。
  - 每一批节点的最大数量不会超过节点总数的10%。

- 集群升级前置检查
  在您开始集群升级之后，ACK会为您自动启动集群升级前置检查。该检查会对集群进行多项健康检查，以确保您的集群可以顺利的完成此次升级。
  如果您的集群存在不合理配置或者潜在风险，则无法通过前置检查，如下图所示。



单击**查看详情**按钮。跳转到集群运维页面，查看具体的失败原因。



> **说明**
> - 如果遇到前置检查失败请自行修复，或者提交工单。
> - 升级检查仅针对集群升级前进行的前置检查，即使不通过也不会影响当前集群的运行及集群状态。
> - 如果您的集群顺利通过升级检查。则可以进入集群升级环节。

- 集群升级暂停
  通过集群升级暂停功能，您可以在集群升级的任意阶段对其升级进程进行暂停。

  > **说明**
  > - 暂停升级之后，当前批次已经开始升级的节点会完成升级。还未开始升级的节点不会升级。
  > - 集群暂停状态为集群升级的中间状态，建议您不要在此时对集群进行操作，并尽快完成升级过程。

  您可以在集群成功暂停之后，单击**继续**，恢复集群的升级进程。
  如果集群升级过程中发生错误，集群升级进程会被系统所暂停。具体失败原因会展示在页面下方详情中。您可根据报错进行排查或者提交工单。
- 集群升级取消
  您可以在暂停升级后，单击**取消**，对本次升级进行取消操作。

  > **说明**
  > - 取消升级之后，当前批次已经开始升级的节点会完成升级。还未开始升级的节点不会升级。
  > - 已经完成升级的节点不受影响。

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，请参见创建一个云盘快照。
- 集群升级Kubernetes过程中，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。
- 由于老版本的FlexVolume（v1.11.2.5及以前）挂载的OSS卷在升级的时候会重新挂载，使用OSS卷的Pod在集群升级后需要重建。
- 如果您对Kubernetes集群有过任何的配置更改（例如打开了swap分区），则升级过程有可能失败。
- 集群升级过程中您可以在一批节点升级完成后中断进程，此时集群处于升级的中间状态，建议您不要对集群进行

操作，并尽快完成升级过程。处于中间状态的集群会在15日之后关闭升级过程，同时清理一切升级相关的事件和日志信息。

- 集群升级过程中，如非发生错误，请勿修改kube-upgrade命名空间下面的相关资源。
- 如果集群升级失败，升级过程会暂停，您需要分析失败原因并清理kube-upgrade命名空间下失败的Pod，确认修复成功后重启升级过程。如需帮助，请联系在线客服。
- 在完成Kubernetes集群升级后，请您同步升级您本地的kubectl版本。如果未及时升级，在使用本地kubectl的过程中可能会因为与集群API Server版本不同，发生类似**invalid object doesn't have additional properties**的报错。
- 在升级到Kubernetes v1.14之前，检查是否做了对应配置，排除访问LoadBalancer暴露出去的SLB地址不通的风险，请参见Kubernetes集群中访问LoadBalancer暴露的SLB地址不通。
- 在升级ACK集群的Kubernetes 1.16到1.18版本时，如果您集群中的应用使用了云盘数据卷，且数据卷类型为Block Volume，请在升级前阅读CSI Block Volume升级说明。
- ACK集群的Kubernetes版本升级只能一级一级的升级。例如低版本ACK集群的Kubernetes版本为1.12，如果要升级到1.18 Kubernetes版本，需进行三次手动升级，先升级到1.14，然后1.16，最后1.18版本。
- 由于Kubernetes 1.20版本的集群不支持selfLink字段，若您的集群使用FlexVolume存储插件且安装了alicloud-nas-controller组件，在将低版本集群升级至Kubernetes 1.20版本之前需要将alicloud-nas-controller的镜像升级到v1.14.8.17-7b898e5-aliyun或更高版本。

准备工作

> **说明** 如果您在非生产环境中有待升级的集群，强烈建议您先对该集群进行升级验证，再在生产环境中启动集群升级。

请在集群升级前检查集群的健康状况，确保集群已具备升级条件。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**集群 > 集群**。
3. 在控制台左侧导航栏中，单击**集群**。
4. 在**集群列表**页面中，选择目标集群，并在目标集群右侧的**操作**列下，选择**更多 > 集群检查**。
5. 在**容器智能运维**左侧导航栏中，选择**检查 > 升级检查**。
6. 在**升级检查**页面，单击**执行升级检查**。
7. 在弹出的**升级检查**页面，选中**注意事项**后，单击**执行检查**。
   检查完成后，单击**查看详情**。
   当检查报告中**检查结果**为**正常**时，表示升级检查成功，您可以进行集群升级操作。

如果检查结果异常可以自行修复，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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 集群升级**。
4. 单击**升级**。
5. 弹出升级提示页面，单击**确定**。
   此时，您可以可视化的看到升级的全过程。

> **说明**
> - 如果在升级到某个阶段您需要暂停升级时，您可以通过单击**暂停**，暂停集群的升级进程。
> - 如果您需要取消集群升级，可以在暂停升级后，单击**取消**，取消对该集群的升级。

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。

4.5.4. 扩容ACK集群的节点

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

使用限制

目前不支持扩容集群中的Master节点。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 集群扩容**。
4. 在**节点池**页面单击**扩容**。
5. 在目标节点池扩容对话框中的**扩容数量**右侧输入需要扩容的节点数。

> **注意** 请确保您的集群可添加足够的节点数。如果您需要添加更多节点，请提交工单申请扩大配额。关于ACK集群的配额限制，请参见使用限制。

6. 在**扩容数量**下侧单击**修改节点池配置**。

   有关配置项的详细说明，请参见创建Kubernetes托管版集群。部分配置项说明如下：

配置项	描述
地域	默认显示集群所在的地域。
容器运行时	默认选择Docker 19.03.5。
专有网络	设置节点的网络。您可以在已有VPC列表中选择所需的VPC。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型，付费详情请参见计费方式概述。选择**包年包月**时： ○ 需设置**购买时长**。目前支持选择1、2、3、6个月和1~5年。 ○ 需设置是否**自动续费**。
实例规格	支持选择多个实例规格。更多信息，请参见实例规格族。
已选规格	选中的规格呈现在这里。

配置项	描述
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ○ 支持选中**开启云盘备份**以备份云盘数据。 ○ **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。
登录方式	○ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ○ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。
ECS标签	为ECS实例添加标签。
节点标签	为Kubernetes集群节点添加标签。
污点（Taints）	为集群内所有Worker节点添加污点。
自定义镜像	允许您选择一个自定义ECS镜像。选择自定义镜像后，扩容的节点将基于此镜像进行部署 。有关创建自定义镜像操作，请参见使用自定义镜像创建Kubernetes集群 **说明** ○ 目前自定义镜像的操作系统仅支持CentOS 7.x和Alibaba Cloud Linux 2.x。在使用自定义镜像扩容节点前，请确保使用该自定义镜像的节点已被通过**手动添加**方式成功加入集群中。有关如何手动添加节点至集群，请参见手动添加节点。 ○ 当前只有白名单用户可使用该功能。提交工单申请使用。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 **说明** 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。

配置项	描述
实例自定义数据	请参见ECS实例自定义数据概述。

7. 单击**提交**。
   在**节点池**页面，如果节点池**状态**显示**更新中**，则说明节点池正在扩容中。扩容完成后，**状态**显示为**已激活**。

后续步骤

单击目标节点池中的**详情**，在**节点管理**页签中，查看节点池中的扩容节点。

4.5.5. 升级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

- 您已成功创建一个专有版Kubernetes集群。有关如何创建Kubernetes集群，参见创建Kubernetes专有版集群。
- 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

1. 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: apps/v1
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=##CLUSTER_ID##
&public=true'
```

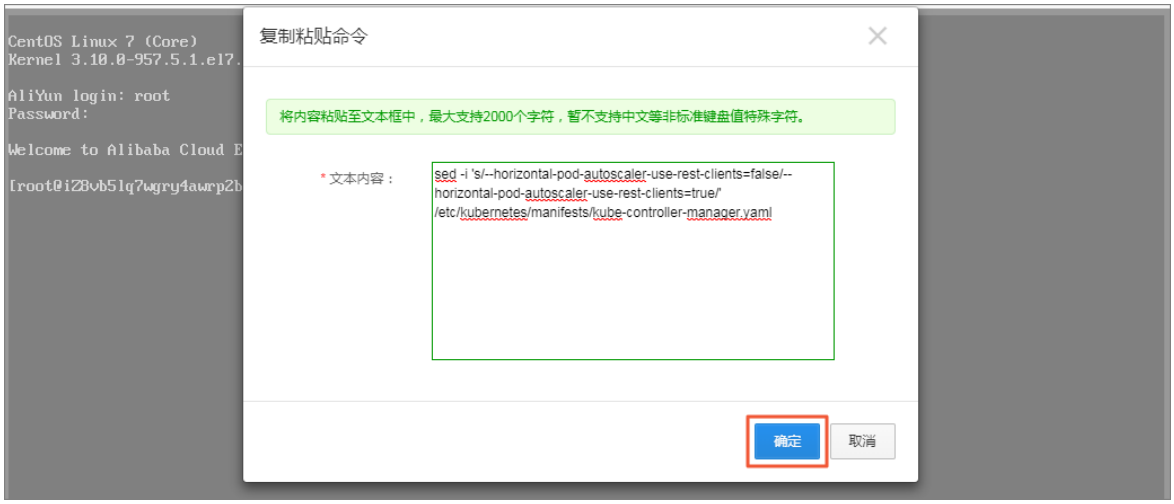
说明 您需要将 ##REGION## 与 ##CLUSTER_ID## 替换为所选集群的地域名称（例如，华东1：cn-hangzhou）与集群ID。

切换监控数据链路

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4.
5. 找到集群的三个Master节点。单击Master节点的实例ID。
6. 在**实例详情**页面，单击**远程连接**。

   在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
   ```

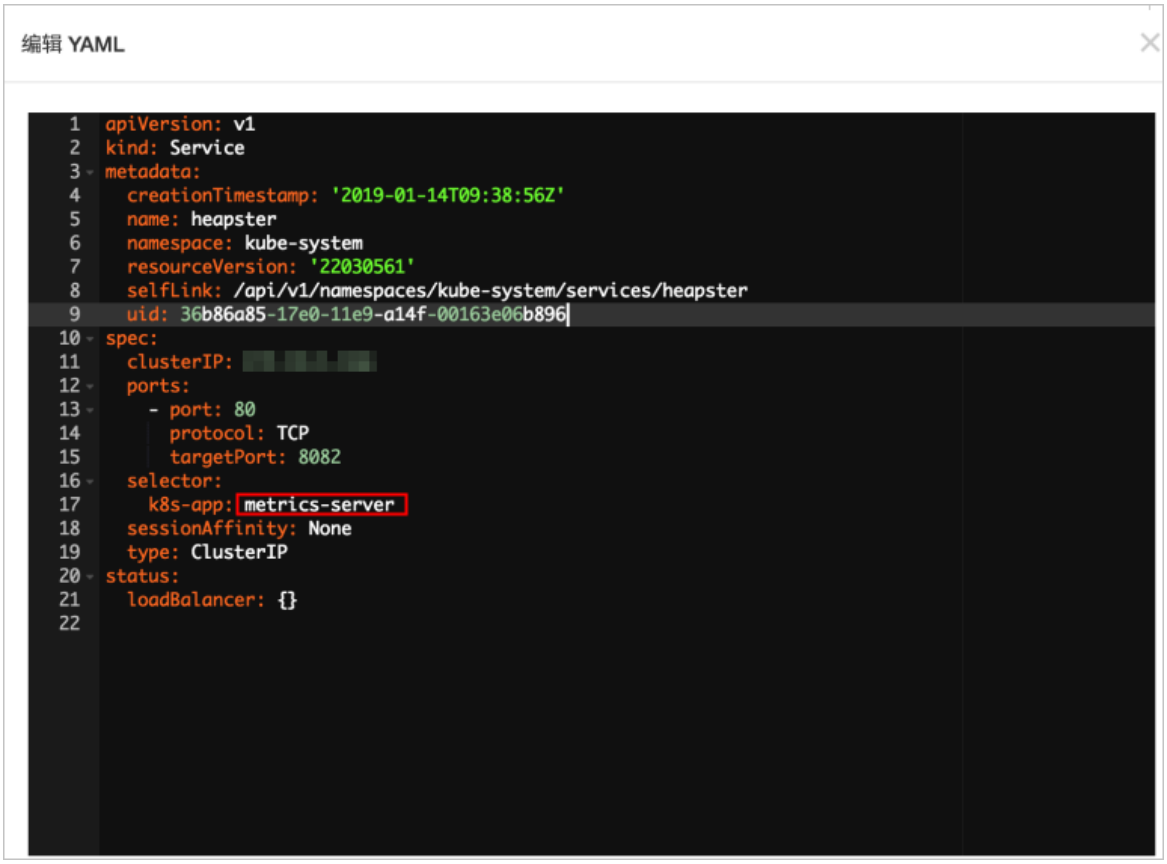


7. 重复步骤5-6，在另外两个Master节点上执行该命令。

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. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 选择目标命名空间**kube-system**，并单击服务**heapster**右侧**操作**列下的**查看YAML**。
6. 在弹出的对话框中，修改selector中k8s-app的值为metrics-server。单击**更新**。



7. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
8. 在**无状态**页签，选择目标**命名空间**kube-system。
9. 选择Heapster相关组件（heapster和monitoring-influxdb），并单击右侧**操作**列下的**更多 > 删除**。
10. 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**。

11. 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。
    在集群管理页，单击**工作负载 > 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。

名称	状态	重启次数	Pod IP	节点	创建时间	CPU（核）	内存（字节）
alicloud-application-controller-6859945f97-4fx95	运行中	0			2019-01-28 11:54:55	0.001	9.035 Mi
alicloud-disk-controller-5df487b4dc-hbtxf	运行中	0			2019-01-28 11:54:55	0	10.164 Mi
alicloud-monitor-controller-d9bbf8485-4k5lw	运行中	0			2019-01-28 11:54:55	0.002	31.762 Mi
aliyun-acr-credential-helper-7d7fc6f5bb-g78gs	运行中	0			2019-01-28 11:54:56	0	34.074 Mi
cloud-controller-manager-59qsh	运行中	0			2019-01-28 16:31:47	0	10.313 Mi
cloud-controller-manager-8j2xm	运行中	0			2019-01-28 16:31:30	0	10.406 Mi
cloud-controller-manager-xshcq	运行中	0			2019-01-28 16:31:33	0.016	34.637 Mi
coredns-84cc785c87-2hp97	运行中	0			2019-01-28 11:54:56	0.002	9.07 Mi
coredns-84cc785c87-56bdp	运行中	0			2019-01-28 11:54:56	0.001	8.902 Mi
flexvolume-227ls	运行中	3			2019-02-14 14:35:13	0	0

4.5.6. 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。

> 注意 删除集群后，手动添加的节点不会被自动释放。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择所需的集群并单击右侧的**更多 > 删除**。
4. 在**删除集群**对话框中，确认内容无误后，选中**我已知晓以上信息并确认删除集群**，单击**确定**。

常见问题

1、删除集群失败怎么办？

如果您在阿里云资源编排服务ROS（Resource Orchestration Service）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。例如在ROS创建的VPC下手动添加了一个VSwitch，这样就会导致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。



1. 单击删除失败的集群对应的**更多 > 删除**。
2. 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，选中**保留资源**，则不会释放这些资源，您需要手动释放。有关如何排查不能释放的资源，请参见删除Kubernetes集群失败。

2、如何关闭集群删除保护状态？

如果集群开启了集群删除保护功能，在删除集群时，页面会提示需先关闭删除保护。您可以执行以下步骤关闭集群删除保护功能：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在目标集群右侧的**操作**列下，选择**更多 > 修改集群删除保护状态**。
4. 在**修改集群删除保护状态**对话框中，关闭**集群删除保护**开关。

4.6. 通过集群检查功能定位集群问题

本文介绍如何通过容器服务提供的集群检查功能快速定位Kubernetes集群和网络的常见问题。

前提条件

- 已成功创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- Kubernetes集群处于正常运行中状态（在**集群列表**中确认集群状态为**运行中**）。

适用场景

您只需要单击**一键检查**，无需输入任何参数，就能对您的集群资源、集群组件及集群配置进行全面检查并提供相应建议。建议您定期对集群进行检查和维护。

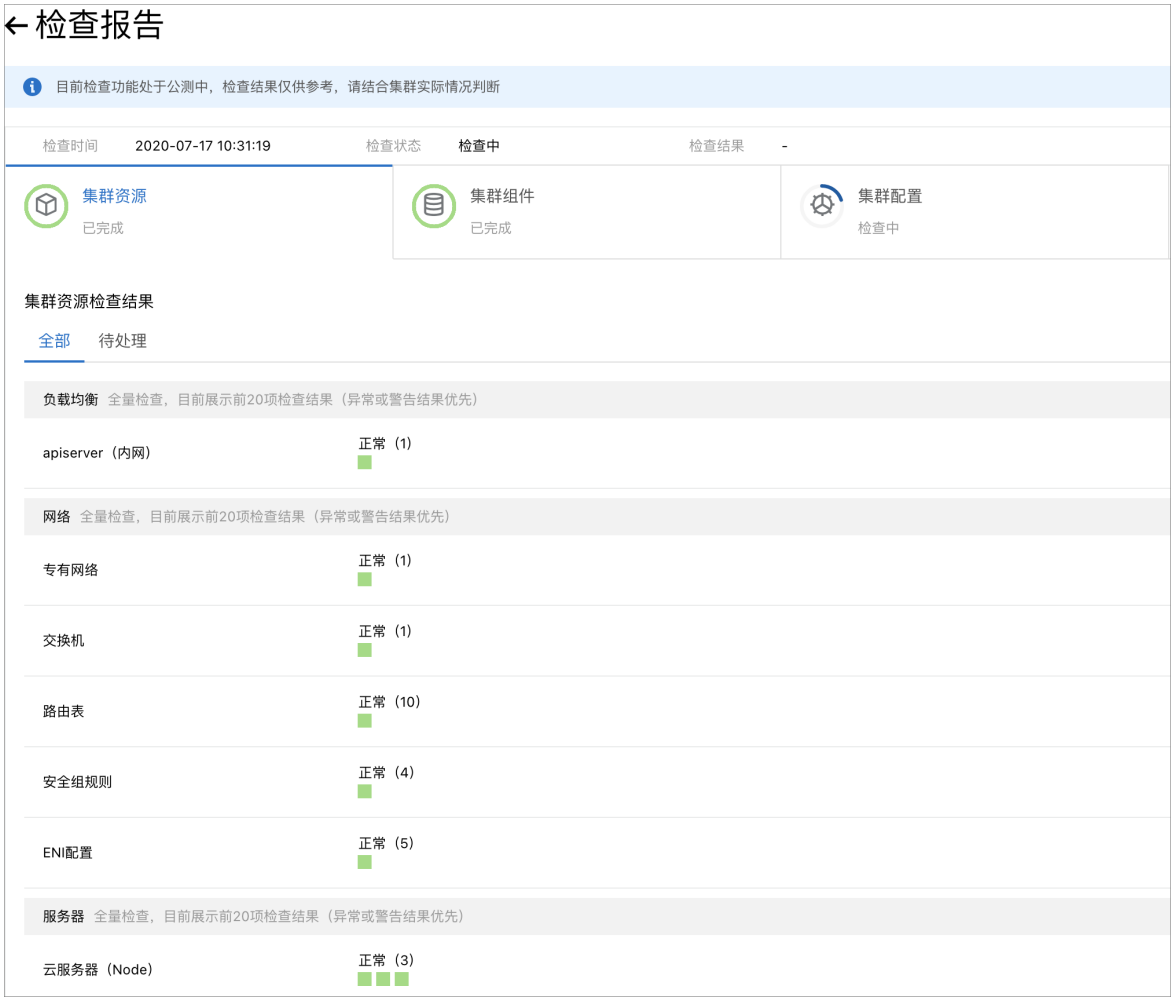
操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，选择目标集群，并在目标集群右侧的**操作**列下，选择**更多 > 集群检查**。
4. 选择**评测 > 一键检查**，或单击工具箱区域的**一键检查**，进入**一键检查**页面。



- 集群信息：呈现集群的名称、类型、运行状态、以及最近一次检查的结果。
- 一键检查：按照时间由近及远的顺序，呈现前5次检查时间及检查结果（如果集群首次执行检查任务，该区域呈现的是**执行一键检查**，您也可以单击该按钮进入**一键检查**页面）。
- 工具箱：提供的集群检查功能，当前一键检查功能已上线，后续将为您提供节点检查等更多集群检查功能。

5. 单击**执行一键检查**，在弹出的**一键检查**页面，确认并选中注意事项，单击**执行检查**，启动集群检查任务。
6. 您可以看到集群检查的进度，单击**查看详情**，在**检查报告**页面，查看集群资源和集群配置的状态。



7. 已经完成的检查任务，可以根据检查报告提示进行相应处理。

- 正常：无需处理。
- 警告：需要确认，引起集群异常的情况需要处理。
- 异常：尽快处理，避免集群无法正常工作。

针对单项检查结果，可以单击状态颜色框，查看被检查对象信息及具体检查结果，检查结果包含检查内容、可能影响以及推荐的修复方案，您可以参考影响及修复方案酌情处理。

后续步骤

若通过集群检查的检查报告无法定位节点、Pod及网络的具体故障原因，建议您通过集群故障诊断功能进行故障的诊断修复。具体操作，请参见通过集群故障诊断功能定位集群问题。

4.7. 通过集群故障诊断功能定位集群问题

阿里云容器服务ACK提供一键故障诊断能力，包括节点诊断、Pod诊断、网络诊断，可以辅助您定位集群中出现的问题。本文主要介绍如何在ACK集群中使用故障诊断功能。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- Kubernetes集群处于正常运行中状态。
  在容器服务管理控制台，**集群列表**中目标集群的**集群状态**为**运行中**。

适用场景

您已明确当前故障所在Pod、节点或网络，单击**节点诊断**、**Pod诊断**或**网络诊断**后需要输入对应诊断项参数，发起诊断后可获取具体的诊断结果及修复建议。

使用限制

目前仅支持在华北5（呼和浩特）、西南1（成都）、华北3（张家口）及华南2（河源）地域使用集群故障诊断功能。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 集群检查**。
5. 在**容器服务运维中心**的左侧导航栏中选择**评测 > 故障诊断**。
6. 单击**节点诊断**、**Pod诊断**或**网络诊断**，在对应诊断配置面板配置参数。

   关于诊断项的参数介绍如下表所示：

诊断项	参数	说明
节点诊断	**节点名称**：目标节点名称。例如，`cn-hangzhou.10.160.XX.XX`。	如果某个节点的状态为NotReady，输入对应节点名称执行节点诊断，检测集群配置及机器进程状态等，给出诊断项和诊断结果。
Pod诊断	○ **Pod Namespace**：目标Pod所在命名空间。 ○ **Pod Name**：目标Pod名称。	输入对应Pod的命名空间和名称执行Pod诊断，会检测Pod状态、配置、所属节点状态等检查项，给出诊断项和诊断结果。

诊断项	参数	说明
网络诊断	○ **源地址**：源地址IP。 ○ **目的地址**：目的地址IP。 ○ **端口**：目的端口。 ○ **报文跟踪**：选择是否需要报文跟踪。主机会实际发送TCP或UDP报文，在对应的网络节点上尝试捕获这些报文，根据报文到达的位置，判断哪些链路上会丢包。报文追踪需要运行1~5分钟。	执行网络诊断，输入源地址、目的地址和目的端口，可以模拟网络访问，给出诊断结果。

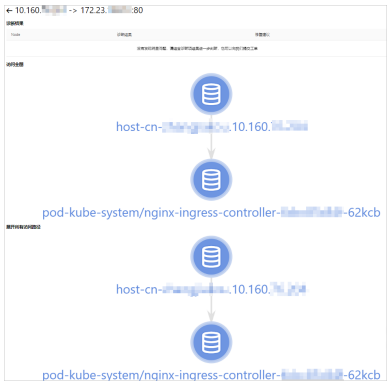
> **注意** 诊断操作需要在您的集群节点上执行数据采集程序并收集检查结果，采集的信息包括系统版本、负载、Docker、kubelet等组件运行状态及系统日志中关键错误信息，不会采集任何和客户相关的业务信息以及敏感数据。

配置相关参数后，选中**我已知晓并同意**，单击**发起诊断**。

7. 在故障诊断列表，单击目标诊断项右侧**操作**列的**诊断详情**。

   关于诊断详情的说明如下表所示：

诊断项	诊断界面	检查项状态	说明
节点诊断		○ 正常：无需处理。 ○ 警告：需要确认，引起集群异常的情况需要处理。 ○ 异常：尽快处理，避免集群无法正常工作。 ○ 未知：未正常完成检查或结果未知。	诊断详情页中，会给出诊断结果、修复建议及具体的诊断项列表。 ○ 如果有异常或警告类的检查项，会显示在**待处理**页签中。 ○ 检查项状态为异常时，可以在目标检查项右侧**状态**中的**查看详情**的Tips中查看异常。 ○ 节点诊断包括**Node**、**Network**、**NodeComponent**、**ClusterComponent**四类检查项，通过ECS状态、Node状态、机器信息采集及集群组件状态综合判定节点异常原因，具体检测项以诊断详情为准。
Pod诊断			Pod诊断包括**Pod**、**Node**、**Network**、**NodeComponent**、**ClusterComponent**五类检查项。通过Pod状态、节点状态、集群组件状态综合判定Pod异常原因，具体检测项以诊断详情为准。

诊断项	诊断界面	检查项状态	说明
网络诊断		不涉及	根据**诊断结果**、**访问全图**及**展开所有访问路径**查看网络诊断详情。

4.8. 集群管理FAQ

本文主要为您介绍集群管理的常见问题。

- 已有集群是否可以添加支持Intel SGX的节点？
- Alibaba Cloud Linux2操作系统的集群兼容CentOS的容器镜像吗？
- 为何创建Kubernetes集群失败？
- Kubernetes集群扩容常见问题的排查及解决方法
- 为何删除Kubernetes集群失败？
- 通过CloudShell管理集群出现超时问题
- 创建集群后，是否可以更改容器运行时？
- Docker运行时和安全沙箱运行时有什么区别？
- 容器服务ACK通过等保三级认证了吗？
- 误删了专有版集群的一个Master节点后，还能升级集群吗？

已有集群是否可以添加支持Intel SGX的节点？

集群满足以下三个条件后可以添加支持Intel SGX的节点：

- Kubernetes版本大于等于1.14.0。
- 集群的网络插件为**Flannel**。
- 创建集群时操作系统为AliyunLinux 2.xxxx，并且在节点接入时不要选择自定义镜像。

您可以容器服务管理控制台，找到符合以上三个条件的集群，并在该集群右侧的**操作**列选择**更多 > 系统组件升级**，安装aesm和sgx-device-plugin。

Alibaba Cloud Linux2操作系统的集群兼容CentOS的容器镜像吗？

完全兼容。更多信息，请参见使用操作系统Alibaba Cloud Linux 2。

创建集群后，是否可以更改容器运行时？

问题现象：创建集群的时候，选择的容器运行时为Containerd，如何改回Docker运行时？

解决办法：创建集群后，不支持切换容器运行时，但是您可以创建不同类型运行时的节点池。节点池与节点池的运行时可以不同。更多信息，请参见节点池概述。

Docker运行时和安全沙箱运行时有什么区别？

相比于原有Docker运行时，安全沙箱提供的一种新的容器运行时选项，可以把应用运行在一个轻量虚拟机沙箱环境中，拥有独立的内核，具备更好的安全隔离能力。安全沙箱特别适合于不可信应用隔离、故障隔离、性能隔离、多用户间负载隔离等场景。在提升安全性的同时，对性能影响非常小，并且具备与Docker容器一样的用户体验，例如日志、监控、弹性等， 但目前安全沙箱仅支持ECS神龙裸金属型号，暂不支持ECS虚拟机型号。

更多关于Docker和沙箱的信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。

容器服务ACK通过等保三级认证了吗？

容器服务ACK已同阿里云整体一起通过了等保三级认证，但是您如何使用云服务我们并不做限制，这部分的行为还需要您自行负责。以下为阿里云厂商和您各自负责的云服务安全项：

- 阿里云厂商负责的云服务安全项
  - 阿里云服务自身基础设施的安全性。
  - 集群控制平面节点和etcd数据库的安全性。
  - 集群控制平面组件的安全合规性，并接受第三方合规审核人员的定期验收。
- 您负责的云服务安全项
  - 数据平面的安全配置，包括VPC的安全组配置等。
  - 节点和容器应用自身配置。
  - 节点操作系统（包括更新和安全补丁）。
  - 其他关联的应用程序软件。
  - 设备和管理网络侧的访问控制，例如防火墙规则。
  - 使用RAM或其他服务管理平台级身份和访问控制。
  - 敏感数据的安全合规保护。

误删了专有版集群的一个Master节点后，还能升级集群吗？

不能，删除专有版集群的Master节点后，没法添加Master节点，也不能进行集群的K8s版本升级。

5.ACK Pro集群

5.1. ACK Pro版集群概述

ACK Pro托管版集群是在ACK标准托管版基础上针对企业大规模生产环境进一步增强了可靠性、安全性，并且提供可赔付的SLA的Kubernetes集群。
ACK的Pro托管版集群是在原标准ACK托管版集群的基础上发展而来的集群类型，继承了原托管版集群的所有优势，例如Master节点托管、Master节点高可用等。同时，相比原托管版进一步增强了集群的可靠性、安全性和调度性，并且支持赔付标准的SLA，适合生产环境下有着大规模业务，对稳定性和安全性有高要求的企业客户。

使用场景

- 互联网企业，大规模业务上线生产环境，对管控的稳定性、可观测性和安全性有较高要求。
- 大数据计算企业，大规模数据计算、高性能数据处理、高弹性需求等类型业务，对集群稳定性、性能和效率有较高要求。
- 开展中国业务的海外企业，对有赔付标准的SLA以及安全隐私等非常重视。
- 金融企业，需要提供赔付标准的SLA。

功能特点

- 更可靠的托管Master节点：稳定支撑大规模集群的管控；etcd容灾和备份恢复，冷热备机制最大程度保障集群数据库的可用性；管控组件的关键指标可观测，助力您更好地预知风险。
- 更安全的容器集群：管控面etcd默认采用加密盘存储；数据面通过选择安装kms-plugin组件实现Secrets数据落盘加密。开放安全管理，并提供针对运行中容器更强检测和自动修复能力的安全管理高级版。
- 更智能的容器调度：集成更强调度性能的kube-scheduler，支持多种智能调度算法，支持NPU调度，优化在大规模数据计算、高性能数据处理等业务场景下的容器调度能力。
- SLA保障：提供赔付标准的SLA保障，集群API Server的可用性达到99.95%。

定价

ACK Pro版集群的计费详情，请参见产品计费。

容器服务ACK的集群管理费用除了支持按量付费之外，同时还支持以资源包的方式抵扣ACK使用费用。如果您准备购买或者已经购买了ACK Pro版集群，建议您使用ACK资源包，更具性价比。具体信息，请参见ACK资源包。

对比

ACK Pro版托管版集群和ACK标准托管版集群的对比详情如下表。

分类	功能	ACK Pro托管版集群	ACK标准托管版集群
集群规模	不涉及	默认配额最大1000节点。您可到配额平台提交申请至最大5000节点。	最大100节点（现有集群不受影响，可以升级到Pro版）
SLA	不涉及	99.95%（支持赔付）	99.9%（不支持赔付）
API Server	自定义参数设置	✓	✗
	可用性监控	✓	✗
	高频冷热备机制，异地容灾	✓	✗

分类	功能	ACK Pro托管版集群	ACK标准托管版集群
etcd	可观测性监控指标	✓	✗
Kube-scheduler	Gang scheduling调度策略	✓	✗
	CPU拓扑感知调度	✓	✗
	GPU拓扑感知调度	✓	✗
	共享GPU专业版调度	✓	✗
安全管理	开放高级版（支持数据加密，请参见使用阿里云KMS进行Secret的落盘加密）	✓	✗
托管节点池	托管节点池	✓	✗

> 说明
> ACK专有版集群仅支持共享GPU普通版调度。详细信息，请参见共享GPU调度概述。

5.2. 创建ACK Pro版集群

ACK的Pro托管版集群相比原标准托管版集群进一步提升了集群的可靠性、安全性和调度性能，并且支持赔付标准的SLA，适合生产环境下有着大规模业务，对稳定性和安全性有高要求的企业客户。本文介绍如何通过容器服务控制台创建ACK Pro托管版集群。

前提条件

登录RAM管理控制台和弹性伸缩控制台开通相应的服务。

> **说明**
> 您在使用集群过程中，请注意以下限制：
> - 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。
> - ACK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。如果您需要提高配额，请提交工单申请。
>   - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。
>   - 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的ACK集群的网络模式是Flannel时，集群的路由条目最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多路由条目数，您需要对目标VPC提交工单申请提高配额 。
>   - 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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**，选择**Pro托管集群**，然后单击**创建**。
5. 在**ACK托管版**页签，完成Pro版集群配置。
   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
集群规格	选择集群规格，支持**标准版**和**Pro版**。 选中**Pro版**创建Pro托管集群。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。

配置项	描述
账号全部资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
Kubernetes版本	显示当前ACK支持的Kubernetes版本。
容器运行时	支持**Containerd**、**Docker**和**安全沙箱**。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体请参见Terway与Flannel对比。 ■ 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，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。 **说明** ■ 在Terway模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。 ■ 网络插件选择Terway时，会使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。

配置项	描述
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 说明 ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Terway模式	设置**网络插件**为**Terway**时，需要配置**Terway模式**： ■ 设置是否选中**Pod独占弹性网卡以获得最佳性能** ■ 如果选中，Pod将独占一个专有的弹性网卡。 ■ 如果不选中，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。 说明 当前只有白名单用户可使用上述**Pod独占弹性网卡以获得最佳性能**功能。提交工单申请使用。 ■ 设置是否选中**IPvlan** ■ 只在弹性网卡共享模式支持选中。 ■ 如果选中，采用IPvlan eBPF作为网卡共享模式虚拟化技术，并且只能使用Alibaba Cloud Linux 2系统，性能优于默认模式。 ■ 如果不选中，则使用默认模式，采用策略路由作为网卡共享模式虚拟化技术，同时兼容Centos 7和Alibaba Cloud Linux 2的系统。 关于Terway模式的IPvlan模式的详细信息，请参见Terway IPvlan模式。 ■ 设置是否选中**NetworkPolicy支持** ■ 只在弹性网卡共享模式下支持选中，默认不选中。 ■ 如果选中，集群支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制。 ■ 如果不选中，集群将不会支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制，这样将不存在网络策略对Kubernetes的API Server产生过大的负载。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。

配置项	描述
配置SNAT	创建集群时，默认不开通公网。如果您选择的VPC不具备公网访问能力，选中**为专有网络配置SNAT**后，ACK将为您创建NAT网关并自动配置SNAT规则。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 **注意** 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 **说明** 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 **说明** ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。
集群删除保护	创建的集群将归属于选择的资源组。一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念。更多信息，请参见资源组。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。
标签	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。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。 具体操作，请参见自定义集群API Server证书SAN。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影
Secret落盘加密	在ACK Pro托管集群中，选中**选择KMS密钥**可以使用在阿里云密钥管理服务KMS（Key Management Service）中创建的密钥加密Kubernetes Secret密钥。设置Secret落盘加密的详情，请参见使用阿里云KMS进行Secret的落盘加密。

6. 单击**下一步：节点池配置**，完成Worker节点配置。

i. 设置**Worker实例**。

■ 如果您选择**新增实例**，需要进行以下配置。

配置项	描述

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	支持选择多个实例规格。详情请参见实例规格族。
已选规格	呈现选中的规格。
数量	新增Worker实例（ECS实例）的数量。 **说明** 您可以将**数量**设置为0。ACK Pro托管集群支持创建0个Worker节点。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。 ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。

配置项	描述
操作系统	ACK支持的节点操作系统包括： ■ Alibaba Cloud Linux 2.x（ACK默认的操作系统） 选中Alibaba Cloud Linux 2.x操作系统时，您可以进行以下安全加固的配置： ■ 不开启：不对Alibaba Cloud Linux 2.x操作系统进行安全加固。 ■ 等保加固：对Alibaba Cloud Linux 2操作系统进行等保加固。关于等保加固的详细信息，请参见ACK等保加固使用说明。 **注意** 等保加固在保障原生镜像兼容性和性能的基础上进行了等保合规适配，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》。 等保加固会禁止Root用户通过SSH远程登录。您可通过ECS控制台使用VNC方式登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。 ■ CIS加固：对Alibaba Cloud Linux 2.x操作系统进行CIS加固。关于CIS加固的使用说明，请参见ACK CIS加固使用说明。 ■ CentOS 7.x **说明** 暂不支持CentOS 8.x及以上的操作系统。
登录方式	■ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

■ 如果您选择**添加已有实例**，则需要预先在此地域下创建ECS云服务器，然后配置**操作系统**、**登录方式**和**密钥对**，配置描述请参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义镜像	允许您选择一个自定义ECS镜像。选择自定义镜像后，集群所有节点将基于此镜像进行部署 。有关创建自定义镜像操作，请参见使用自定义镜像创建Kubernetes集群。 **说明** ■ 目前自定义镜像的操作系统仅支持CentOS 7.x和Alibaba Cloud Linux 2.x。 ■ 当前只有白名单用户可使用该功能。提交工单申请使用。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

7. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认**不安装**。 ○ 若选择**Nginx Igress**，则默认选中**安装Ingress组件**。关于Nginx Igress的更多使用说明，请参见Ingress高级用法。 ○ 若选择**ALB Ingress（公测）**，则默认安装ALB Ingress Controller组件。关于如何在ACK集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。 NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume和存储管理-CSI。

配置项	描述
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。
报警配置	选中**使用默认报警模板配置报警**，开启默认报警规则。详细介绍，请参见容器服务报警管理。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**创建Ingress Dashboard**，您可以选择是否在日志服务控制台中**创建Ingress Dashboard**。更多信息，请参见Ingress访问日志分析与监控。 默认选中**安装node-problem-detector并创建事件中心**，您可以选择是否在日志服务控制台中添加事件中心。更多信息，请参见创建并使用Kubernetes事件中心。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

8. 单击**下一步：确认配置**。
9. 阅读并同意**服务协议**，然后单击**创建集群**。

说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

执行结果

- 集群创建成功后，您可以在容器服务管理控制台的**集群列表**页面查看所创建的集群。
- 您可以单击目标集群右侧**操作**列下的**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。
- 单击目标集群右侧**操作**列下的**详情**，然后单击**基本信息**和**连接信息**页签，查看集群的基本信息和连接信息。

5.3. CPU拓扑感知调度

本文介绍ACK基于新版的Scheduling framework实现了CPU拓扑感知调度，针对CPU敏感型的工作负载提供更好的性能。

CPU拓扑感知调度

Kubernetes的Node节点会运行多个Pod，其中会有部分的Pod属于CPU密集型的工作负载。在这种情况下，Pod之间会争抢节点的CPU资源。当争抢剧烈的时候，Pod会在不同的CPU Core之间进行频繁的切换，更糟糕的是在NUMA Node之间的切换。这种大量的上下文切换，会影响程序运行的性能。Kubernetes虽然有CPU Manager解决方案处理以上问题，但是因为CPU Manager特性是节点级别的CPU调度选择，所以无法在集群维度中选择最优的CPU Core组合。同时CPU Manager特性要求Pod是Guaranteed时（Pod中的每个容器必须指定CPU Request和CPU Limit，并且两者要相等）才能生效，且无法适用于所有类型的Pod。

在以下场景中，建议使用CPU拓扑感知调度：工作负载为计算密集型、应用程序对CPU敏感、运行在神龙裸金属等多核机器上。

更多信息，请参见CPU拓扑感知调度。

5.4. GPU拓扑感知调度

Kubernetes对节点的GPU拓扑信息不感知，调度过程中对GPU的选择比较随机，选择不同的GPU组合训练速度会存在较大的差异。基于以上问题，ACK基于Scheduling Framework机制，实现GPU拓扑感知调度，在节点的GPU组合中选择具有最优训练速度的组合。
关于GPU拓扑感知调度如何使用，请参见以下文档：

- GPU拓扑感知调度概述
- 安装GPU拓扑感知组件
- Tensorflow分布式训练使用GPU拓扑感知调度
- Pytorch分布式训练使用GPU拓扑感知调度

5.5. 共享GPU调度专业版

本文为您介绍共享GPU调度专业版的相关概念，如何开启共享GPU调度专业版以及如何使用共享GPU调度专业版，帮助您了解和更好地使用共享GPU调度专业版。

阿里云容器服务Kubernetes版ACK（Container Service for Kubernetes）开源了GPU共享调度之后，您能在阿里云、AWS、GCE和自己数据中心的容器集群上通过GPU共享调度框架实现多个容器运行在同一个GPU设备上的目标。ACK开源GPU共享调度降低了使用GPU的经济成本。通过隔离，限制运行在同一个GPU上的多个容器能够按照自己申请的资源使用量运行，避免因为其资源用量超标影响同一个GPU上的其他容器的正常工作，同时也使更小颗粒度的使用GPU提供了可能。想进一步了解：

- 关于什么是共享GPU调度，请参见共享GPU调度概述。
- 关于如何开启共享GPU调度专业版，请参见安装并使用共享GPU组件和资源工具。
- 关于如何使用共享GPU调度专业版，请参见运行共享GPU示例、通过共享GPU实现多卡共享策略、基于节点池管理共享GPU。
- 若ACK专有版集群已安装共享GPU调度基础版，在迁移至ACK Pro托管版集群后，需要将共享GPU调度基础版升级为共享GPU调度专业版。具体操作，请参见在ACK Pro版集群中将共享GPU基础版升级为共享GPU专业版。

5.6. 任务调度

本文主要介绍不同特性的任务调度，包括All-Or-Nothing类型任务的Gang Scheduling、通过弹性配额提高集群整体资源利用率的Capacity Scheduling。

Gang Scheduling

Gang Scheduling是在并发系统中将多个相关联的进程调度到不同处理器上同时运行的策略，其最主要的原则是保证所有相关联的进程能够同时启动，防止部分进程的异常，导致整个关联进程组的阻塞。例如，您提交一个批量Job，这个批量Job包含多个任务，多个任务会同时调度成功或都不能调度成功。这种All-or-Nothing调度场景，就被称作Gang Scheduling。关于如何使用Gang Scheduling，请参见Gang scheduling。

Capacity Scheduling

通过Kubernetes原生的ResourceQuota方式进行固定资源分配，集群的整体资源利用率不高。阿里云借鉴Yarn Capacity Scheduling的设计思路，基于Scheduling Framework的扩展机制，在调度侧通过引入弹性配额实现了Capacity Scheduling功能，在确保资源分配的基础上通过资源共享的方式来提升集群的整体资源利用率。本文介绍如何使用弹性配额组实现Capacity Scheduling。关于如何使用Capacity Scheduling，请参见Capacity Scheduling。

5.7. 使用负载感知调度

负载感知调度是ACK基于Scheduling Framework实现感知节点实际资源负载的调度策略。调度过程中，通过参考节点负载的历史统计，将Pod优先调度到负载较低的节点，实现节点负载均衡的目标，避免出现因单个节点负载过高而导致的应用程序或节点故障。本文介绍如何使用负载感知调度。

前提条件

- 本功能仅适应于ACK Pro版集群，关于如何创建ACK Pro版集群，请参见创建ACK Pro版集群。
- 系统组件版本要求具体如下表所示。

组件	版本要求
Kubernetes	≥v1.18
Helm	≥v3.0
Docker	v19.03.5

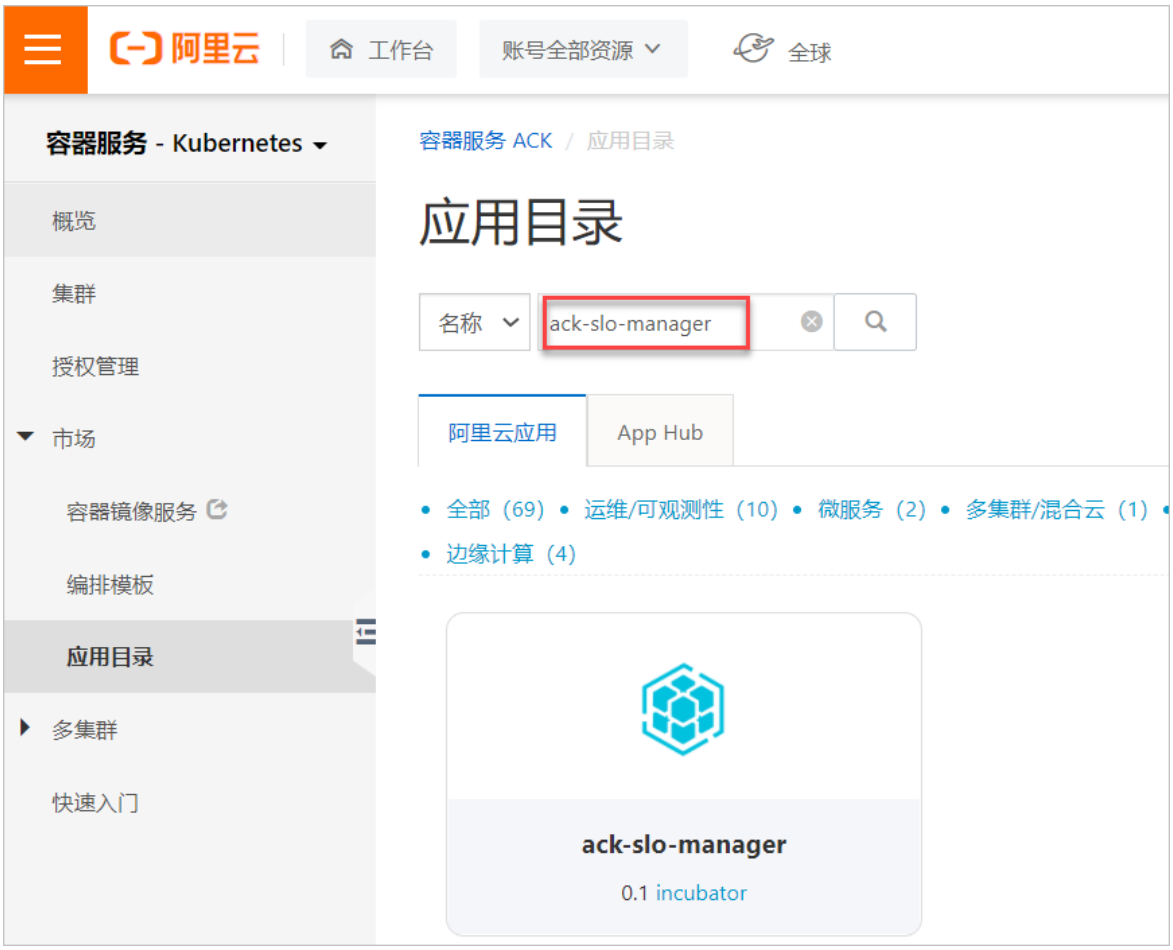
使用方式

通过添加Annotations来标识是否开启负载感知调度。

```
annotations:
 alibabacloud.com/loadAwareScheduleEnabled: "true" # 开启负载感知调度。
```

安装ack-slo-manager组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面，单击**阿里云应用**页签，在名称文本框中搜索**ack-slo-manager**。



4. 在**ack-slo-manager**区域单击，在**详情**页面的右下角单击**创建**。
5. 创建完成后，执行以下命令检查是否成功部署**ack-slo-manager**。

```
helm get manifest ack-slo-manager -n kube-system | kubectl get -n kube-system -f -
```

预期输出：

```
NAME                 DATA  AGE
configmap/ack-slo-manager-config  1    74s
NAME                  SECRETS  AGE
serviceaccount/ack-slo-controller  1     74s
serviceaccount/ack-slo-agent     1     74s
NAME                                          CREATED AT
customresourcedefinition.apiextensions.k8s.io/nodemetrics.nodes.alibabacloud.com  2021-08-06T06:54:11Z
customresourcedefinition.apiextensions.k8s.io/nodeslos.nodes.alibabacloud.com    2021-08-06T06:54:11Z
NAME                                       CREATED AT
clusterrole.rbac.authorization.k8s.io/ack-slo-manager-cluserrole  2021-08-06T06:54:11Z
NAME                                              ROLE                     AGE
clusterrolebinding.rbac.authorization.k8s.io/ack-slo-agent-custom-rolebinding     ClusterRole/ack-slo-manag
er-cluserrole  74s
clusterrolebinding.rbac.authorization.k8s.io/ack-slo-controller-custom-rolebinding  ClusterRole/ack-slo-man
ager-cluserrole  74s
NAME                                        CREATED AT
role.rbac.authorization.k8s.io/ack-slo-manager-role:leaderelection  2021-08-06T06:54:11Z
NAME                                                ROLE                     AGE
rolebinding.rbac.authorization.k8s.io/ack-slo-manager-custom-rolebinding:leaderelection  Role/ack-slo-mana
ger-role:leaderelection  74s
NAME               DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
daemonset.apps/ack-slo-agent  6     6     6    6       6      <none>      74s
NAME                  READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/ack-slo-controller  1/1   1       1      74s
```

如果输出信息与以上信息类似，说明您已成功部署ack-slo-manager。

使用负载感知调度

本文示例的集群有3台4核8 GiB的节点。

1. 使用以下YAML文件，创建一个视频转码作业。

```
apiVersion: batch/v1
kind: Job
metadata:
 generateName: ffmpeg
 namespace: default
spec:
 backoffLimit: 1
 parallelism: 1
 template:
  metadata:
  spec:
   containers:
   - image: registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/ffmpeg:latest
     name: ffmpeg
     resources:
       requests:
         cpu: 1
         memory: 2Gi
       limits:
         cpu: 2
         memory: 4Gi
     command: ["/bin/sh"]
     args: ["-c","while true;do ffmpeg -i /root/mp4/statefulset.mp4 -c:v libx264 -crf 100 -threads 200 destination.
flv -y;sleep 1;done"]
    restartPolicy: OnFailure
```

2. 执行以下命令，检查Pod的运行和部署的情况。

```
# 镜像较大，pod启动大概需要2min
kubectl get pods -o wide
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE   IP         NODE            NOMINATED NODE  READINESS GATES
ffmpegjjpv2-xcl6t  1/1    Running  0      2m6s  172.20.177.8  cn-beijing.10.0.0.46  <none>      <none>
```

从上述输出可知，该作业的Pod运行在节点 `cn-beijing.10.0.0.46` 上。

3. 执行以下命令，查看该集群所有节点的负载详情。

```
kubectl top node
```

预期输出：

```
NAME           CPU(cores)  CPU%  MEMORY(bytes)  MEMORY%
cn-beijing.10.0.0.44  251m     6%    1531Mi      22%
cn-beijing.10.0.0.45  352m     8%    1881Mi      27%
cn-beijing.10.0.0.46  1802m     45%   1689Mi      25%
```

从上述输出可知，此时节点 `cn-beijing.10.0.0.46` 的负载最高。

4. 使用以下YAML文件，创建没有开启负载感知调度的Pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 namespace: default
 labels:
  app: nginx
spec:
 replicas: 6
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   name: nginx
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
    resources:
     limits:
      cpu: 500m
     requests:
      cpu: 500m
```

5. 执行以下命令，查看Pod调度详情。

```
kubectl get pods | grep nginx
```

预期输出：

```
nginx-6976ccb95-96xmf  1/1   Running  0     108s  172.20.177.10   cn-beijing.10.0.0.46  <none>     <none>
nginx-6976ccb95-b6xtp  1/1   Running  0     108s  172.20.177.70   cn-beijing.10.0.0.44  <none>     <none>
nginx-6976ccb95-hmqcw  1/1   Running  0     108s  172.20.177.69   cn-beijing.10.0.0.44  <none>     <none>
nginx-6976ccb95-l2gnn  1/1   Running  0     108s  172.20.177.9    cn-beijing.10.0.0.46  <none>     <none>
nginx-6976ccb95-rzsfc  1/1   Running  0     108s  172.20.176.200  cn-beijing.10.0.0.45  <none>     <none>
nginx-6976ccb95-shv52  1/1   Running  0     108s  172.20.177.71   cn-beijing.10.0.0.44  <none>     <none>
```

从上述输出可知，由于集群没有开启负载感知调度功能，不能感知节点负载，Pod被均衡调度到集群的所有节点上。

6. 执行以下命令，删除没有开启负载感知调度的Pod。

```
kubectl delete deployment nginx
```

7. 使用以下YAML文件，创建开启负载感知调度的Pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 namespace: default
 labels:
  app: nginx
spec:
 replicas: 6
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   name: nginx
   labels:
    app: nginx
   annotations:
    alibabacloud.com/loadAwareScheduleEnabled: "true" # 开启负载感知调度。
  spec:
   containers:
   - name: nginx
    image: nginx
    resources:
     limits:
      cpu: 500m
     requests:
      cpu: 500m
```

8. 执行以下命令，查看Pod调度详情。

```
kubectl get pods | grep nginx
```

预期输出：

```
nginx-7759585d4c-hfblh  1/1   Running  0     72s   172.20.177.74   cn-beijing.10.0.0.44  <none>     <none>
nginx-7759585d4c-hzzt9  1/1   Running  0     72s   172.20.176.203  cn-beijing.10.0.0.45  <none>     <none>
nginx-7759585d4c-m4gmm  1/1   Running  0     72s   172.20.177.72   cn-beijing.10.0.0.44  <none>     <none>
nginx-7759585d4c-v7wfb  1/1   Running  0     72s   172.20.176.201  cn-beijing.10.0.0.45  <none>     <none>
nginx-7759585d4c-wx59r  1/1   Running  0     72s   172.20.177.73   cn-beijing.10.0.0.44  <none>     <none>
nginx-7759585d4c-xbwh6  1/1   Running  0     72s   172.20.176.202  cn-beijing.10.0.0.45  <none>     <none>
```

从上述输出可知，由于集群开启了负载感知调度功能，能感知节点负载，通过运用调度策略，此时Pod被优先调度到除 `cn-beijing.10.0.0.46` 以外的节点上。

5.8. 使用阿里云KMS进行Secret的落盘加密

在ACK Pro托管集群中，您可以使用在阿里云密钥管理服务KMS（Key Management Service）中创建的密钥加密Kubernetes Secret密钥。本文主要介绍如何使用阿里云密钥管理服务（KMS）中管理的密钥对ACK Pro集群中的Kubernetes Secret密钥数据进行落盘加密。

前提条件

- 在KMS控制台已创建用户主密钥。更多信息，请参见管理密钥。

> **说明** 当前开启Pro集群的Secret落盘加密只支持使用Aliyun_AES_256类型的主密钥，同时不支持开启自动轮转周期。

- 主账号需要授权容器服务账号使用AliyunCSManagedSecurityRole系统角色的权限。如果您使用的账号未授权，在开启Secret落盘加密时，系统会提示您进行安全系统角色授权。
- 如果当前登录账号是RAM用户，请确保该RAM用户有AliyunKMSCryptoAdminAccess系统权限。具体操作，请参见为RAM用户授权。
- KMS对API调用（以万次调用为单位）和用户上传密钥的托管会收取一定费用。当您的Pro集群开启了Secret落盘加密特性后，kube-apiserver对Secret实例的读写操作均会调用KMS服务的加解密API，请务必保证您的账号内有足够的余额。当您欠费超过七天后，会影响整个集群的管控能力。关于KMS服务计费的详细说明，请参见计费说明。

背景信息

在Kubernetes集群中，通常使用Secrets密钥模型存储和管理业务应用涉及的敏感信息，例如应用密码、TLS证书、Docker镜像下载凭据等敏感信息。Kubernetes会将所有的Secrets密钥对象数据存储在集群对应的ETCD中。关于密钥的更多信息，请参见Secrets。

在Pro集群中，您可以使用在KMS中创建的密钥加密Kubernetes Secret密钥，加密过程基于Kubernetes提供的KMS Encryption Provider机制，使用信封加密的方式对存储在ETCD中的Kubernetes Secret密钥进行自动加密和解密。Kubernetes Secret密钥进行加密和解密的过程如下：

- 当一个业务密钥需要通过Kubernetes Secret API存储时，数据会首先被API Server生成的一个随机的数据加密密钥加密，然后该数据密钥会被指定的KMS密钥加密为一个密文密钥存储在ETCD中。
- 解密Kubernetes Secret密钥时，系统会首先调用KMS的解密OpenAPI进行密文密钥的解密，然后使用解密后的明文密钥对Secret数据解密并最终返回给用户。

更多信息，请参见KMS Encryption Provider机制和什么是信封加密。

在新建的ACK Pro集群中开启Secret落盘加密

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 单击页面右上角的**创建集群**，在弹出的**选择集群模板**页面，选择**Pro版集群**，并单击**创建**。
4. 在**ACK托管版**页签找到**Secret落盘加密**，选中**选择KMS密钥**，在下拉框中选择KMS密钥ID。创建ACK Pro集群的其他配置请参见创建ACK Pro版集群。



登录操作审计控制台，在左侧导航栏单击**历史事件查询**，在**历史事件查询**页面有使用aliyuncsmanagedsecurityrole系统角色的加密和解密事件日志，则说明该集群后台已成功开启Secret落盘加密特性。



在已创建的ACK Pro集群中开启Secret落盘加密

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面单击目标Pro集群名称。
4. 在集群详情页面单击**基本信息**页签，在**基本信息**区域中打开**Secret落盘加密**开关。

> **说明** 如果当前登录用户为子账号，请确保该子账号对该集群有RBAC的管理员或运维人员权限，授权流程请参考配置RAM用户RBAC权限。

当集群状态由**更新中**变为**运行中**时，说明该集群Secret落盘加密的特性已变更完成。

在已创建的ACK Pro集群中关闭Secret落盘加密

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面单击目标Pro集群名称。
4. 在集群详情页面单击**基本信息**页签，在**基本信息**区域中关闭**Secret落盘加密**开关。

> **说明** 如果当前登录用户为子账号，请确保该子账号对该集群有RBAC的管理员或运维人员权限，授权流程请参考配置RAM用户RBAC权限。

当集群状态由**更新中**变为**运行中**时，说明该集群Secret落盘加密的特性已变更完成。

5.9. 自定义ACK Pro集群的管控面参数

为了满足生产环境调整管控面参数的需求，ACK Pro托管版集群为您提供了管控面参数自定义功能。您可以根据需要修改托管组件Kube API Server和Kube Controller Manager（KCM）的参数。本文介绍如何自定义ACK Pro托管版集群的管控面参数。

注意事项

为了确保顺利完成管控面的参数修改，请仔细阅读以下注意事项：

- 修改参数之后，管控面会进行重启，建议您选择业务低谷期进行参数修改操作。
- 修改参数之后，您输入的参数会覆盖ACK Pro提供的默认参数，从而导致默认参数失效。
- 为了保证管控面的稳定性，当前ACK Pro仅支持自定义部分参数。
- 请确保您输入的参数正确且完整，错误参数可能会导致管控面启动失败。关于参数设置的详细信息，请参见kube-apiserver和kube-controller-manager。

自定义Pro集群的管控面参数

下文以修改Kube API Server组件为例，说明如何修改组件参数。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**核心组件**区域，单击目标组件右下方的**配置**。
6. 在**kube-apiserver参数配置**对话框中，输入您的自定义参数，然后单击**确定**。

> **说明** 请确保您输入的参数完整性和正确性。目前ACK Pro集群只支持修改Kube API Server和KCM组件的参数。关于Kube API Server和KCM参数的具体格式和合法值，请参见kube-apiserver和kube-controller-manager（需选择对应的Kubernetes版本）。

默认参数列表

当您进入组件的参数配置操作之后，组件原有的默认参数会被覆盖。您可以参考以下表格将参数恢复为默认值。

Kubernetes版本	组件名	参数	默认值
1.16	kube-apiserver	ServiceNodePortRange	可选范围10000~65535，默认30000~32767。
		EnableAdmissionPlugins	• 如果开启了PodSecurityPolicy：默认值为 `NodeRestriction,PodSecurityPolicy` 。 • 如果关闭了PodSecurityPolicy：默认值为 `NodeRestriction` 。
	kube-controller-manager	HorizontalPodAutoscalerSyncPeriod	15s
1.18和1.20 **说明** Kubernetes 1.18和1.20版本的组件名、参数和默认值是相同的。	kube-apiserver	ServiceNodePortRange	30000~32767
		EnableAdmissionPlugins	• 如果开启了PodSecurityPolicy：默认值为 `NodeRestriction,PodSecurityPolicy` 。 • 如果关闭了PodSecurityPolicy：默认值为 `NodeRestriction` 。
	kube-controller-manager	HorizontalPodAutoscalerSyncPeriod	15s
		LargeClusterSizeThreshold	默认值50。
		UnhealthyZoneThreshold	默认值0.55。
		SecondaryNodeEvictionRate	默认值0.01。
		PodEvictionTimeout	默认值5 m。
		NodeEvictionRate	默认值0.1。
	cloud-controller-manager	RouteTableIDS	默认为空。如果VPC内有多个路由表，可以手动设置CCM支持多个路由表ID，以半角逗号（,）分隔，例如vtb-**,vtb***。

6.节点与节点池

6.1. 节点

6.1.1. 添加已有节点

您可以通过ACK控制台将已有的ECS实例添加到已创建的Kubernetes集群中，目前仅支持添加Worker节点。本文介绍自动添加节点和手动添加节点两种方式。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 您需要将待添加的ECS实例添加到Kubernetes集群自动创建的节点安全组中。具体操作，请参见ECS实例加入安全组。

使用限制

- 请确保您的集群可添加足够的节点数。如果您需要添加更多节点，请提交工单申请扩大配额。关于ACK集群的配额限制，请参见使用限制。
- 添加的云服务器必须与集群在同一地域同一VPC下。
- ACK仅支持添加同一账号下云服务器。
- ACK支持添加以下操作系统的节点：
  - Alibaba Cloud Linux 2
  - CentOS 7.x

    > **说明** 暂不支持CentOS 8.x及以上的操作系统。

  - Windows Server 2019和Windows Server Core, version 1809及以上

添加实例限制

安全组数量限制

当添加已有实例到节点池时，会将实例加入到节点池安全组中，对于实例已经加入的安全组不会重复加入。由于ECS实例能够加入的安全组有一定的限制，因此需要保证实例加入节点池后实例所加入的安全组数量小于其限制。关于云服务器ECS的使用限制，请参见使用限制。

安全组规则限制

安全组是一种虚拟防火墙，用于控制安全组内ECS实例的入流量和出流量，从而提高ECS实例的安全性。

安全组分为普通安全组、企业安全组和托管安全组，其中托管安全组由云产品创建，且只有创建托管安全组的云产品才有对其规则更改的权限，因此创建节点池时应避免选择托管安全组，否则会造成创建节点池失败。普通安全组和企业安全组的详情和区别如下表所示：

功能	普通安全组	企业安全组
未添加任何规则时的访问策略	• 入方向：拒绝所有访问请求 • 出方向：允许所有访问请求	• 入方向：拒绝所有访问请求 • 出方向：拒绝所有访问请求
能容纳的私网IP地址数量	2000	65536
同一个安全组内实例之间的网络连通策略	默认内网互通	默认内网隔离，需要您手动添加安全组规则

功能	普通安全组	企业安全组
授权给其他安全组	支持组组授权	不支持组组授权

创建节点池时，会为节点池安全组默认添加规则，用于集群内的通信。关于如何配置普通安全组和企业安全组的集群访问规则，请参见最小化集群访问规则。

添加实例到安全组时，需要注意实例已有的安全组规则不应与以下安全组规则冲突，否则将导致Pod之间无法通信：

- Pod网络CIDR
- VPC的附加IPv4网段

自动添加节点

自动添加节点方式会列出当前账号下可用的ECS云服务器，在Web界面进行安装部署，并自动添加到集群。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面，单击目标节点池右侧**操作**列的**添加已有节点**。
6. 在**选择已有云服务器实例**配置项页面，选择自动添加已有云服务器实例。

   选择**添加方式**为**自动添加**，在已有云服务器的列表中，选择所需的ECS云服务器。
7. 单击**下一步**，完成**填写实例信息**。

配置项	说明
集群ID/名称	当前要添加的集群信息，已默认配置。
数据盘挂载	设置是否将容器和镜像存储在数据盘。 ○ 如果ECS已挂载数据盘，且最后一块数据盘的文件系统未初始化，系统会自动将该数据盘格式化为ext4，用来存放内容*/var/lib/docker*、*/var/lib/kubelet*。 **说明** 数据盘内原有数据将丢失，请注意备份数据。 ○ 如果ECS未挂载数据盘，则不会挂载新的数据盘。
保留实例名称	添加节点时，默认开启**保留实例名称**。如果您不需要保留实例名称，您可以关闭**保留实例名称**，此时会按照自定义节点名称指定的规则来重命名节点。
实例信息	添加的云服务器实例的实例ID及实例名称。

8. 单击**下一步**，在**添加已有实例到集群**对话框中单击**确定**。

手动添加节点

注意 手动添加到ACK集群的ECS节点不会随集群删除而被释放。

手动添加节点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。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面，单击目标节点池右侧**操作**列的**添加已有节点**。
6. 在**选择已有云服务器实例**配置项页面，选择手动添加已有云服务器实例。

   选择**添加方式**为**手动添加**，在已有云服务器的列表中，选择所需的ECS云服务器，
7. 单击**下一步**，完成**填写实例信息**。

配置项	说明
集群ID/名称	当前要添加的集群信息，已默认配置。
数据盘挂载	设置是否将容器和镜像存储在数据盘。 ○ 如果ECS已挂载数据盘，且最后一块数据盘的文件系统未初始化，系统会自动将该数据盘格式化为ext4，用来存放内容*/var/lib/docker*、*/var/lib/kubelet*。 **说明** 数据盘内原有数据将丢失，请注意备份数据。 ○ 如果ECS未挂载数据盘，则不会挂载新的数据盘。
保留实例名称	添加节点时，默认开启**保留实例名称**。如果您不需要保留实例名称，您可以关闭**保留实例名称**，此时会按照自定义节点名称指定的规则来重命名节点。
实例信息	添加的云服务器实例的实例ID及实例名称。

8. 单击**下一步**，进入**添加完成**页面。在**添加完成**页面复制执行命令，单击**完成**。
9. 登录ECS管理控制台ECS管理控制台，单击左侧导航栏中的**实例与镜像 > 实例**，选择集群所在的地域，选择需要添加的ECS实例。
10. 单击ECS实例右侧的**远程连接**。在**远程连接与命令**对话框，选择远程连接方式后进入ECS远程连接界面。

    关于远程连接方式，请参见下表：

远程连接方式	说明
Workbench远程连接	关于如何使用Workbench远程连接ECS实例，请参见通过密码或密钥认证登录Linux实例或通过密码或密钥认证登录Windows实例。
VNC远程连接	关于如何使用VNC远程连接ECS实例，请参见通过密码认证登录Linux实例或通过密码认证登录Windows实例。
发送远程命令（云助手）	推荐使用，发送远程命令可以帮助您在实例内部快速执行命令，即无需远程连接登录实例，即可完成查看硬盘空间、安装软件、启动停止服务等操作。该功能通过云助手的命令执行功能实现。关于如何安装或激活云助手客户端，请参见安装云助手客户端。

11. 在ECS实例远程连接界面，根据页面指导，输入步骤8保存的命令，单击**执行**开始执行脚本。
    等待脚本执行成功，该云服务器就添加成功。

执行结果

1. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
2. 在**节点池**页面，单击目标节点池右侧**操作**列的**详情**。
3. 在目标节点池页面，单击**节点管理**页签。
   可查看刚才添加的节点信息。

6.1.2. 最小化集群访问规则

创建集群或节点池指定已有安全组时，系统默认不会为安全组配置额外的访问规则，请自行管理安全组规则。本文介绍如何配置普通安全组和企业安全组的集群访问规则。

您可以通过添加安全组规则，允许或禁止安全组内的ECS实例对公网或私网的访问。更多信息，请参见添加安全组规则。

普通安全组

入方向

集群访问规则	协议	端口	授权对象
推荐范围	ICMP	-1/-1（不限制端口）	0.0.0.0/0
	所有协议	-1/-1（不限制端口）	• 集群默认安全组ID • 集群Pod网络地址段（Flannel网络模式添加安全组规则，Terway网络模式不添加安全组规则）
最小范围	所有协议	53/53（DNS）	• 集群默认安全组ID • 集群Pod网络地址段（Flannel网络模式添加安全组规则，Terway网络模式不添加安全组规则）
	ICMP	-1/-1（不限制端口）	
	TCP	• 10250（Kubelet） • 10255（Kubelet） • 443（Webhook）	
	所有协议	所有应用或组件期望被访问的端口	所有应用或组件期望被访问的来源地址或者来源安全组

出方向

集群访问规则	协议	端口	授权对象
推荐范围	所有协议	-1/-1（不限制端口）	0.0.0.0/0
最小范围	所有协议	-1/-1（不限制端口）	100.96.0.0/10 (云产品网段)
	所有协议	53/53（DNS）	• 集群APIServer SLB地址 • 集群默认安全组ID • 集群Pod网络地址段（Flannel网络模式添加安全组规则，Terway网络模式不添加安全组规则）
	TCP	• 10250（Kubelet） • 10255（Kubelet） • 443（APIServer） • 6443（APIServer）	
	所有协议	所有应用或组件期望访问的端口	所有应用或组件期望访问的目的地址或者目的安全组

企业安全组

入方向

集群访问规则	协议	端口	授权对象
推荐范围	ICMP	-1/-1（不限制端口）	0.0.0.0/0
	所有协议	-1/-1（不限制端口）	• 集群所属的VPC网段 • 集群所属的附加VPC网段 • 集群Pod网络地址段（Flannel网络模式添加安全组规则，Terway网络模式不添加安全组规则）
最小范围	所有协议	53/53（DNS）	• 集群内所有关联的vSwitch网段，包括Node vSwitch和Pod vSwitch • 集群Pod网络地址段（Flannel网络模式添加安全组规则，Terway网络模式不添加安全组规则）
	ICMP	-1/-1（不限制端口）	
	TCP	• 10250（Kubelet） • 10255（Kubelet） • 443（Webhook）	
	所有协议	所有应用或组件期望被访问的端口	所有应用或组件期望被访问的来源地址或者来源安全组

出方向

集群访问规则	协议	端口	授权对象
推荐范围	所有协议	-1/-1（不限制端口）	0.0.0.0/0
最小范围	所有协议	-1/-1（不限制端口）	100.96.0.0/10 (云产品网段)
	所有协议	53/53（DNS）	• 集群APIServer SLB地址 • 集群内所有关联的vSwitch网段，包括Node vSwitch和Pod vSwitch • 集群Pod网络地址段（Flannel网络模式添加安全组规则，Terway网络模式不添加安全组规则）
	TCP	• 10250（Kubelet） • 10255（Kubelet） • 443（APIServer） • 6443（APIServer）	
	所有协议	所有应用或组件期望访问的端口	所有应用或组件期望访问的目的地址或者目的安全组

6.1.3. 监控节点

阿里云容器服务ACK集群集成了监控服务，可通过云监控管理控制台查看对应ECS实例的基本监控信息。本文介绍如何查看Kubernetes集群下ECS实例的节点的监控信息。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。

5. 在**节点**页面，单击目标节点右侧**操作**列的**监控**。
6. 进入云监控管理控制台，在**主机监控**页面单击**基础监控**页签，查看对应ECS实例的基本监控信息，包括CPU使用率、公网流入带宽、公网流出带宽、磁盘平均BPS、磁盘平均IOPS等指标。



后续步骤

- 查看关于操作系统级别的监控指标，需要安装云监控组件。具体操作，请参见插件概览。
- 查看关于进程级别的监控指标，需要安装云监控组件。具体操作，请参见插件概览。
- Kubernetes集群新增了关于应用分组的监控功能。具体操作，请参见基础资源监控进行升级。

6.1.4. 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

您已创建一个ACK集群，请参见创建Kubernetes托管版集群。

背景信息

关于如何使用节点标签实现节点调度，请参见设置节点调度。

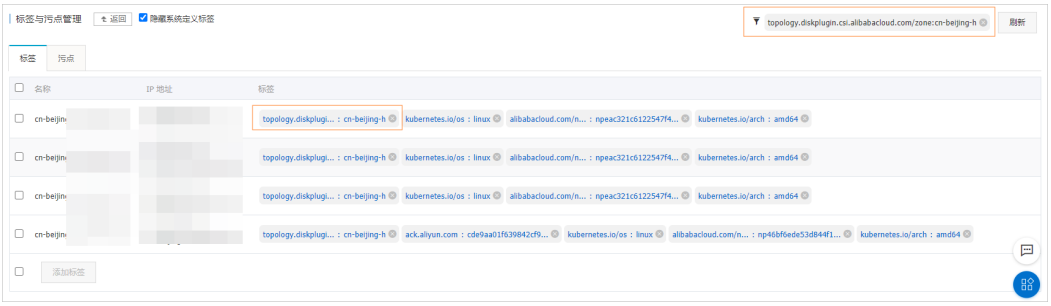
批量添加节点标签

1. 登录容器服务管理控制台。
2. 进入**标签与污点管理**页面。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面右上角单击**标签与污点管理**。
3. 单击**标签**页签，批量选择节点，然后单击**添加标签**。
4. 在**添加**对话框中，输入标签的**名称**和**值**，然后单击**确定**。
   在**标签**页面，可以看到为批量选择的节点添加了相同的标签。

通过标签筛选节点

1. 登录容器服务管理控制台。

2. 进入**标签与污点管理**页面。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面右上角单击**标签与污点管理**。
3. 单击**标签**页签，选择某个节点，单击右侧的标签，可通过标签来筛选节点。
   您可以看到通过标签成功筛选出所需的节点。



删除节点标签

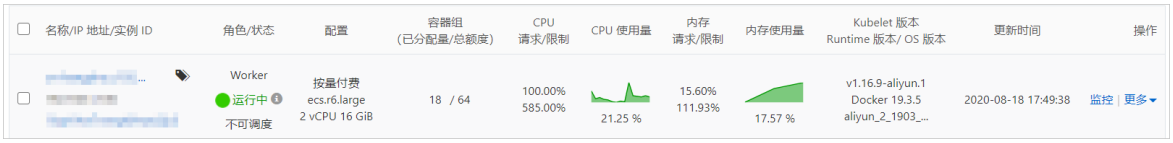
1. 登录容器服务管理控制台。
2. 进入**标签与污点管理**页面。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面右上角单击**标签与污点管理**。
3. 单击**标签**页签，选择某个节点，单击标签的删除图标。
   您可以看到该节点右侧的标签消失，节点标签被删除。

6.1.5. 设置节点调度

您可以通过控制台设置节点调度，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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点**页面中，在目标节点**操作**列进行调度设置。
   - 选择**更多 > 调度设置**，您在后续进行应用部署时，Pod不会再调度到该节点。
   - 选择**更多 > 节点排水**，您在后续进行应用部署时，则Pod不会再调度到该节点，并且该节点上由DaemonSet控制的Pod不会被排空。
6. 单击**确定**。
   可以看到节点的调度状态发生变化。



后续步骤

如果您需要将不可调度的节点重新上线，需要进行以下操作：

1. 在目标节点**操作**列，选择**更多 > 调度设置**。
2. 在**调度设置**对话框中，单击**确定**。

此时该节点调度状态变为**可调度**。

6.1.6. 批量运维节点

您可以基于运维编排服务OOS对容器服务Kubernetes版（ACK）集群的节点进行批量操作来帮助您提升运维效率，例如批量获取节点日志。本文介绍如何批量运维ACK节点。

前提条件

- 您已创建一个ACK集群，具体操作参见创建Kubernetes托管版集群。
- 您已在OOS控制台开通运维编排服务OOS。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 选择目标集群，在该集群下选中需要批量维护的节点，单击**批量运维**。
6. 在**批量运维**对话框中选择目标运维操作，然后单击**确定**。

   支持的批量运维操作包括：
   - 安全更新操作系统内核。
   - 安装自定义软件包。
   - 安装或卸载YUM或APT包。
   - 执行Shell命令。
7. 设置执行运维编排服务（OOS）的配置项。

   关于执行OOS的配置项具体描述，请参见创建执行。
8. 完成执行OOS配置项设置后，单击**确定**。

   OOS执行创建后，自动返回OOS控制台的**执行管理**页签。您可查看提交的执行。
   - 如果选择了**自动执行**，则自动开始执行。执行完毕后，执行状态显示**成功**。
   - 如果选择了**单步执行**，则需要手动开始执行。

   单击创建的执行ID名称或在**操作列**下的**详情**，查看执行状态。

6.1.7. Master节点的资源升配

Kubernetes专有版集群的Master节点由用户自己管控，随着集群规模的不断扩大，用户可能会遇到Master节点的配置不足的情况，本文为您介绍如何升级Master的配置。

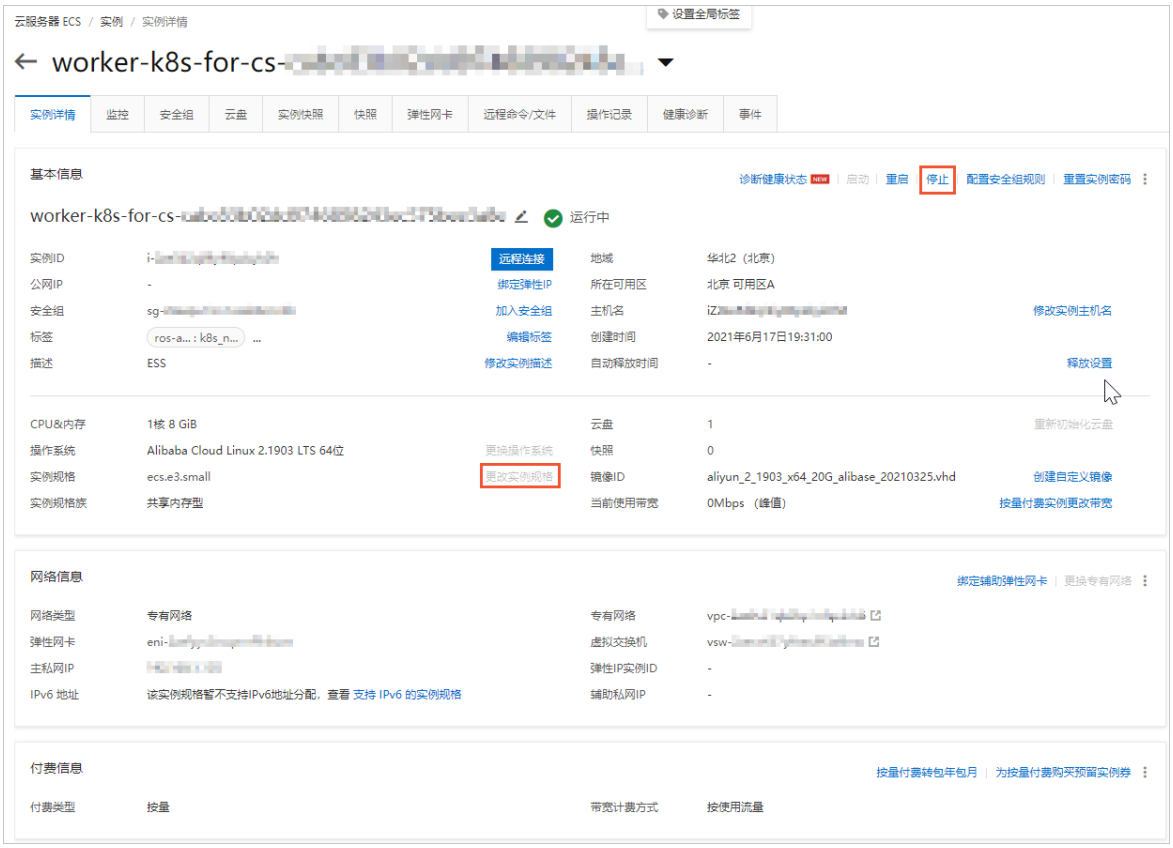
前提条件

创建Kubernetes专有版集群

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 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（本文以master-01为例）。
   此时，您可以看到实例规格等信息。



6. 在实例的配置信息区域，单击**更改实例规格**。



如果**更改实例规格**为灰显，说明实例正在运行，您需要单击界面右上角的**停止**，停止运行该实例，再进行下一步。

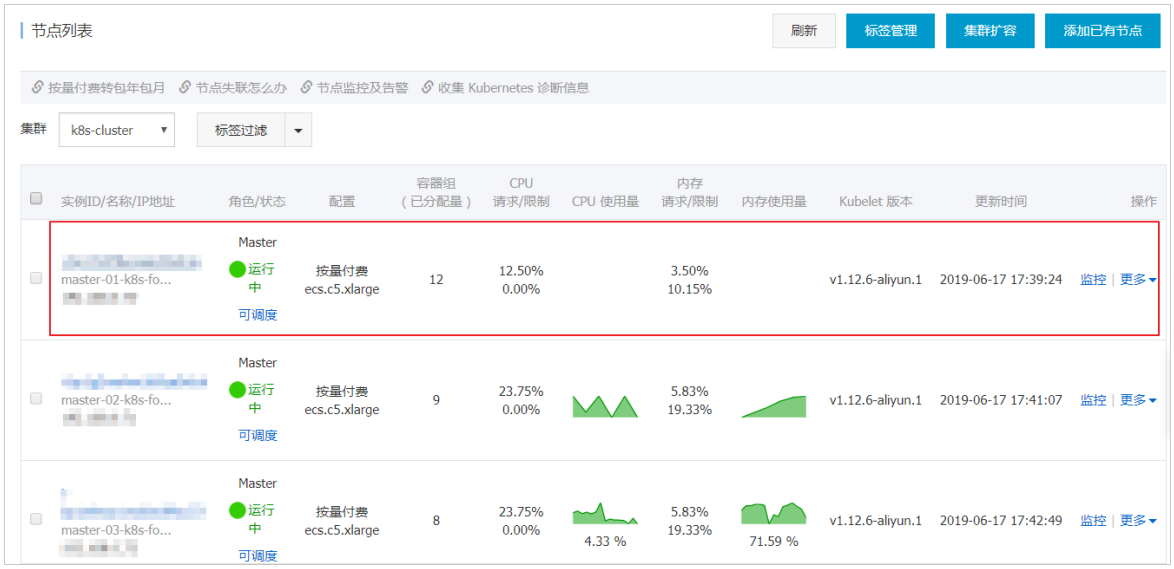
7. 在弹出的**调整规格**页面，选择合适的规格，单击**确认调整**。节点规格请参见选择Master节点规格。
8. 在弹出的**调整规格成功**页面，单击**确认**。

   此时自动跳转到**实例列表**页面。



可以看到master-01节点的规格已经调整成功。

9. 选中目标实例，单击**启动**，等待升配的master-01节点自动加入集群且状态变成**运行中**，如图所示。



10. 重复步骤4-7，直至所有Master节点均已升级到期望的配置。

> **说明** 本例仅介绍了按量付费类型的Master节点的升配方式，关于更多类型升配方式，请参见升降配方式概述。

6.1.8. Worker节点的资源升配

随着Kubernetes集群负载的增加，用户常常面临资源不足的情况，此时您可以对集群的基础资源做一些扩展。例如，当集群的节点数达到10个以上后，您可以优先考虑升级Worker节点的配置，这样既可以提高资源的利用率，也可以降低集群运维的复杂度。本文为您介绍如何升级Worker节点的配置。

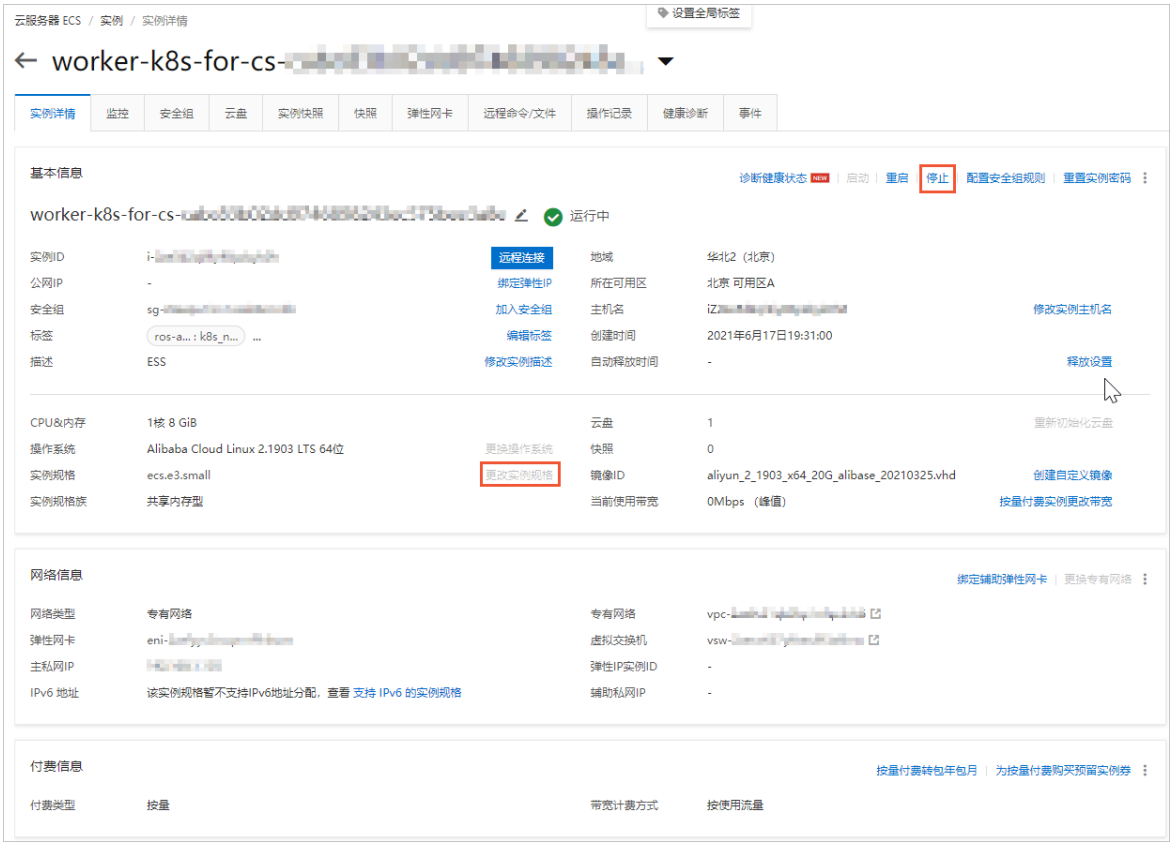
前提条件

已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 单击目标Worker节点的实例ID，进入**实例详情**页面（本文以worker-k8s-for-cs实例为例）。

   此时，您可以看到目标节点的实例规格等信息。
   本例仅介绍了按量付费类型的Worker节点的升配方式，关于更多类型升配方式，请参见升降配方式概述。
6. 若**实例规格**右侧的**更改实例规格**为灰显，说明实例正在运行 ，您需要单击界面的**停止**。



在**停止实例**对话框选中**停止方式**及**停止模式**后，单击**确定**。

7. 在**实例规格**右侧，单击**更改实例规格**。



8. 在**更改实例规格**页面，选择合适的实例规格，单击**立即变更**。

   关于节点实例规格，请参见选择Worker节点规格。

9. 在**调整规格成功**对话框，单击**管理控制台**。

   此时自动跳转到**实例**页面。



可以看到worker-k8s-for-cs节点的规格已经调整成功。

10. 在目标实例右侧**操作**列，选择**更多 > 实例状态 > 启动**，在**启动实例**对话框单击**确定**后，等待升配的worker-k8s-for-cs节点自动加入集群且状态变成**运行中**即可完成Worker节点的资源升配。

6.1.9. 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
```

预期输出：

```
7.9G    /var/lib/docker
```

增加数据盘

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。

1. 创建ECS数据盘，并挂载到需要扩容的机器上。具体操作，请参见创建云盘。
   i. 通过云服务器ECS控制台创建需要配置的云盘。
   ii. 选择左侧导航栏中的**实例与镜像 > 实例**。
   iii. 单击目标ECS实例ID，进入**实例详情**页面。
   iv. 单击**云盘**页签。
   v. 单击右上角的**挂载云盘**。
   vi. 在**挂载云盘**对话框中，选择创建的磁盘作为**目标磁盘**。单击**确定**。
   vii. 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。
   i. 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。

ii. 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。

```
root@                                   :~# ll /dev/xvd*
brw-rw---- 1 root disk 202,  0 May 26 15:44 /dev/xvda
brw-rw---- 1 root disk 202,  1 May 26 15:44 /dev/xvda1
brw-rw---- 1 root disk 202, 16 May 27 13:03 /dev/xvdb
root@                                   :~# fdisk -S 56 /dev/xvdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x446953ae.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-62914559, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-62914559, default 62914559):
Using default value 62914559

Command (m for help): wq
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@                                   :~# ll /dev/xvd*
brw-rw---- 1 root disk 202,  0 May 26 15:44 /dev/xvda
brw-rw---- 1 root disk 202,  1 May 26 15:44 /dev/xvda1
brw-rw---- 1 root disk 202, 16 May 27 13:08 /dev/xvdb
brw-rw---- 1 root disk 202, 17 May 27 13:08 /dev/xvdb1
root@                                   :~# mkfs.ext4 /dev/xvdb1
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
1966080 inodes, 7864064 blocks
393203 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
240 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

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。关于迁移Swarm集群的操作，请参见指定多节点调度。关于迁移Kubernetes集群的操作，请参见Safely Drain a Node while Respecting Application SLOs。

   i. 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。

   ii. 移动Docker的目录到一个备份的目录。例如：**mv /var/lib/docker /var/lib/docker_data**。

iii. 将已格式化的新磁盘挂载到*/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
```

iv. 将之前备份的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。

i. 启动Docker Daemon和kubelet，命令分别是**service docker start**和**service kubelet start**。

ii. 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。

```
root@                                   :/var/lib# df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev             497280       4    497276   1% /dev
tmpfs            101628     712    100916   1% /run
/dev/xvda1     41151808 1928420  37109960   5% /
none                 4       0         4   0% /sys/fs/cgroup
none              5120       0      5120   0% /run/lock
none            508136     288    507848   1% /run/shm
none            102400       0    102400   0% /run/user
/dev/xvdb1     30831612  667168  28575248   3% /var/lib/docker
```

iii. 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。

```
root@                                   :/var/lib# docker ps
CONTAINER ID        IMAGE                                                COMMAND                  CREATED             STATUS
4f564091bffa        registry.aliyuncs.com/acs/logspout:0.1-41e0e21        "/bin/logspout"          21 hours ago        Up 3 minutes
gspout_2
a5aba5fbedae        registry.aliyuncs.com/acs/ilogtail:0.9.9             "/bin/sh -c 'sh /usr/"   21 hours ago        Up 3 minutes
gtail_2
5e3d8fe154bb        registry.aliyuncs.com/acs/monitoring-agent:0.7-1cf85e6   "acs-mon-run.sh --hel"   21 hours ago        Up 3 minutes
_acs-monitoring-agent_1
fb72c2388b0e        registry.aliyuncs.com/acs/volume-driver:0.7-252cb09   "acs-agent volume_exe"   21 hours ago        Up 3 minutes
er_volumedriver_2
604fcb4ad720        registry.aliyuncs.com/acs/routing:0.7-c8c15f0         "/opt/run.sh"            21 hours ago        Up 3 minutes
uting_1
8fe1d6ed15b5        registry.aliyuncs.com/acs/agent:0.7-6967e86           "acs-agent join --nod"   21 hours ago        Up 3 minutes
999da3883264        registry.aliyuncs.com/acs/tunnel-agent:0.21           "/acs/agent -config=c"   21 hours ago        Up 3 minutes
```

5. 您可以通过调度的方式使被迁移走的容器回归到该节点上。

更多容器服务的相关内容，请参见容器服务。

6.1.10. 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。

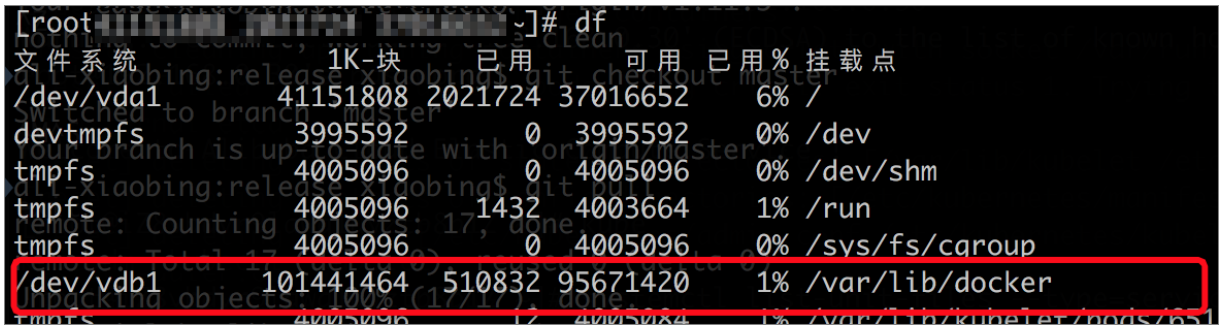
挂载数据盘

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，请参见为容器服务的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. 节点排水。

   具体操作，请参见设置节点调度。

2. 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。

   i. 登录容器服务管理控制台。

   ii. 在控制台左侧导航栏中，单击**集群**。

   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。

   iv. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。

   v. 选择要移除的节点，单击**批量移除**或者选择**更多 > 移除**。

   vi. 在**移除节点**对话框中，单击**确定**。

3. 重新添加刚才移除的节点。

   i. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。

   ii. 在**节点**页面，单击**添加已有节点**。

   iii. 选择**自动添加**或**手动添加**节点。本文中以自动添加为例。

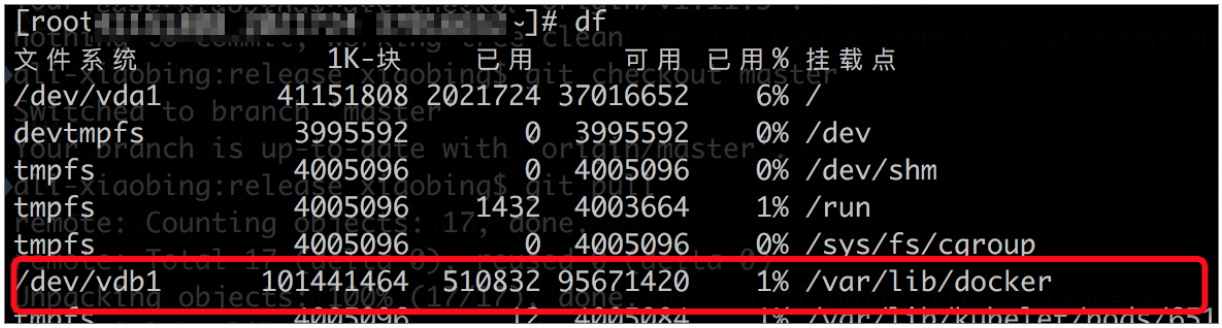
   iv. 选择已有云服务器并单击**下一步**。

   v. 按照界面提示完成其余操作，然后单击**下一步**。

   vi. 在**添加已有实例到集群**对话框中单击**确定**。

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。

6.1.11. 管理污点

污点可以使Pod排斥一类特定的节点，每个节点上都可以应用一个或多个污点。本文介绍如何批量添加污点和删除污点。

前提条件

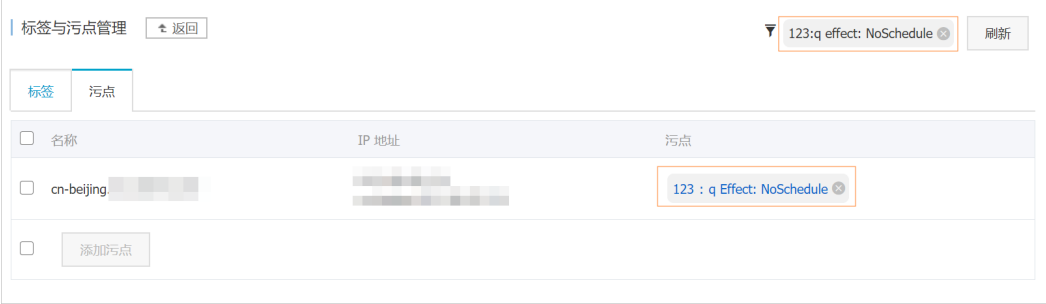
您已创建一个ACK集群，请参见创建Kubernetes托管版集群。

批量添加污点

1. 登录容器服务管理控制台。
2. 进入**标签与污点管理**页面。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面右上角单击**标签与污点管理**。
3. 单击**污点**页签，批量选择节点，单击**添加污点**。
4. 在弹出的对话框中设置**名称**、**值**和**Effect**。
5. 单击**确定**。
   在**污点**页面，可以看到批量选择的节点添加了相同的污点。

通过污点筛选节点

1. 登录容器服务管理控制台。
2. 进入**标签与污点管理**页面。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面右上角单击**标签与污点管理**。
3. 单击**污点**页签，选择某个节点，单击右侧的污点，可通过污点来筛选节点。
   您可以看到通过污点成功筛选出所需的节点。



删除污点

1. 登录容器服务管理控制台。
2. 进入**标签与污点管理**页面。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面右上角单击**标签与污点管理**。
3. 在**标签与污点管理**页面单击**污点**页签，选择某个节点，单击，在弹出的对话框中单击**确定**。

   您可以看到该节点右侧的污点消失，污点被删除。

6.1.12. 一键采集节点的诊断日志

您可以通过容器服务Kubernetes版控制台同时诊断多个节点并采集相关日志信息。本文介绍如何一键采集被诊断节点的日志。

前提条件

- 创建Kubernetes托管版集群
- 开通OSS服务

（可选）

授权将日志采集到OSS

如果您希望将诊断日志采集到OSS中，需要给节点授权，向OSS指定目录写入数据。操作步骤如下。

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏，选择**权限管理 > 权限策略**。
3. 在**权限策略**页面，单击目标权限策略名称。
4. 在**策略内容**页签下，单击**修改策略内容**。
5. 在**修改策略内容**面板中增加以下授权规则，并填入您的Bucket名称和存放目录。




```
{
  "Action": [
    "oss:GetBucket",
    "oss:PutObject",
    "oss:GetObject"
  ],
  "Resource": [
    "acs:oss:*:*:<bucket名称>/<存放诊断日志的目录>/*"
  ],
  "Effect": "Allow"
},
{
  "Action": [
    "oss:GetBucketInfo"
  ],
  "Resource": [
    "acs:oss:*:*:<bucket名称>"
  ],
  "Effect": "Allow"
}
```

6. 单击**确定**。

选择节点进行诊断

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在控制台左侧导航栏，选择**集群 > 节点**。
6. 在**节点**页面，选择需要被诊断的节点，然后单击**节点诊断**。
7. 在**节点诊断**对话框中，执行以下操作。
   - 选中**上传到OSS**，然后执行以下操作。
     您可以将采集日志上传到OSS Bucket中。进行此步骤前，需要授权角色将日志采集到OSS，请参见授权将日志采集到OSS。

a. 填写OSS Bucket的名称和目录。
例如，希望在*myBucket*的*/acs/diagnose*目录中上传诊断日志，您可以填入*myBucket/acs/diagnose*。



b. 设置是否选中**共享诊断日志**。
如果选中系统会为诊断日志生成一个临时链接，您可以直接下载链接，也可以将链接共享给ACK支持团队协助诊断。

c. 单击**确定**。
完成上述操作后，您可以在控制台获取云助手**任务ID**， 根据此ID在ECS云助手控制台查看诊断脚本的执行日志。

d. 在**节点诊断**对话框，单击**在云助手控制台查看诊断脚本执行日志**链接。

e. 在**云助手**页面，单击**命令执行结果**页签。

f. 找到目标**命令执行ID**，然后在其右侧单击**查看**。
您可以在OSS控制台中对应的目录查看采集的诊断文件。



- 不选中**上传到OSS**，然后单击**确认**。
如果不选中**上传到OSS**，您无需授权角色将日志采集到OSS。您可登录ECS控制台，选择**运维与监控 > 发送命令/文件（云助手） > 命令执行结果**，查看相关诊断日志文件。

6.1.13. 查看节点资源请求量和使用量

本文介绍如何通过容器服务控制台查看Kubernetes集群各节点资源占用情况。

前提条件

已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

操作步骤

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，单击**节点管理 > 节点**。
5. 在**节点**页面目标节点的**操作**列选择**更多 > 详情**。

   您可以查看各个节点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请求值）/ 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的内存请求值）/ 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。具体操作，请参见设置节点调度。
   > - 当节点的请求量和使用量达到100%时，新的Pod不会被调度到当前节点上。

6.1.14. 查看节点列表

您可以通过命令或者容器服务管理控制台查看Kubernetes集群的节点列表。

通过命令查看

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

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

```
kubectl get nodes
```

输出示例：

```
kubectl get nodes
NAME            STATUS   AGE    VERSION
iz2ze2n6ep53tch701y****  Ready    19m    v1.6.1-2+ed9e3d33a07093
iz2zeafr762wibijx39****  Ready    7m     v1.6.1-2+ed9e3d33a07093
iz2zeafr762wibijx39****  Ready    7m     v1.6.1-2+ed9e3d33a07093
iz2zef4dnn9nos8elyr****  Ready    14m    v1.6.1-2+ed9e3d33a07093
iz2zeitvvo8enoreufs****  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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。

6.1.15. 移除节点

当您不再需要该节点继续工作时，可以通过以下操作移除节点。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes专有版集群。

- 已通过kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

使用说明

- 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。
- 移除节点请通过控制台进行操作，如果使用**kubectl delete node**方式手动移除节点，则：
  - 移除后的节点无法再添加到其他集群上。
  - 删除集群时，该节点所在的ECS实例会被释放。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点管理**页面中，单击目标节点右侧**操作**列下的**更多 > 移除**。

   > **说明** 如果需要同时移除多个节点，可在**节点列表**页面，同时选中要移除的节点，单击**批量移除**。

6. 在**移除节点**对话框中，可选中**同时释放ECS**和**自动排空节点（drain）**，单击**确定**。
   - **同时释放ECS**：
     - 仅释放按量付费类型的ECS实例，不被释放的ECS会继续收费。
     - 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，请参见包年包月转按量付费。
     - 若不选择**同时释放ECS**，该节点所在的ECS实例会继续计费。
   - **自动排空节点（drain）**：把待移除节点上的Pod转移到其他节点。请确保集群其他节点的资源充足。
     您还可以通过执行命令 `kubectl drain node-name [options]` 的方式把待移除节点上的Pod转移到其他节点。

     > **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。例如，*cn-hangzhou.i-xxx*。
     > - options 为该命令的可选配置参数。例如，可配置为 --force --ignore-daemonsets --delete-local-data 。更多信息，请执行命令 kubectl drain --help 查看。

6.2. 节点池

6.2.1. 节点池概述

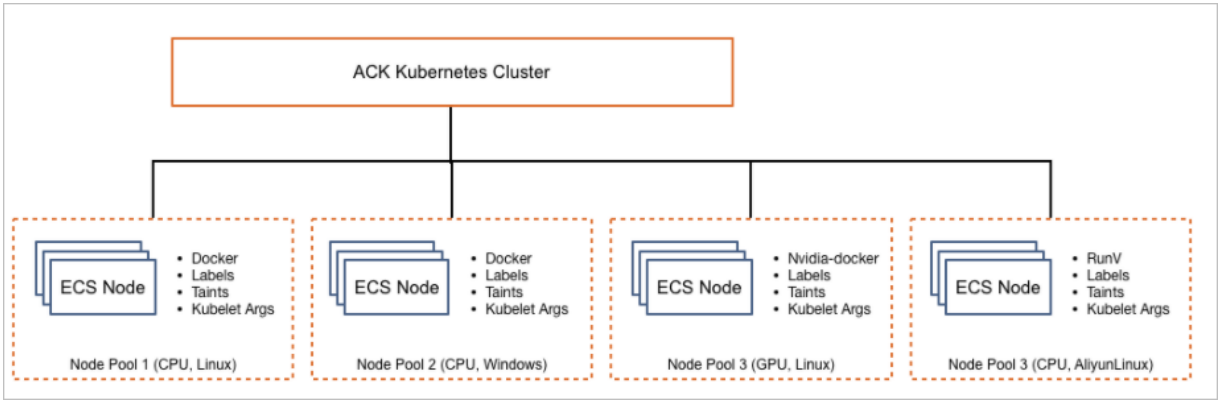
通过节点池，您可以更方便地对节点进行分组管理，比如节点运维，节点配置，开启节点自动弹性伸缩，批量管理，指定调度等。本文介绍节点池的概念、使用说明、使用限制，及节点池分类。

节点池概念

节点池是集群中具有相同配置的一组节点，节点池可以包含一个或多个节点。创建集群时指定的节点数和配置将成为默认节点池，您可以向集群添加其他不同大小和类型的节点池。您可以创建和升级节点池而不会影响整个集群。

通常情况下，在节点池内的节点均具有以下相同属性：

- 节点操作系统及版本。
- 容器运行时类型及版本。
- 计费类型（目前支持按量付费和包年包月，弹性节点池支持抢占式实例）。
- 节点规格。
- 节点CPU Policy。
- 节点标签、污点（Taints）。
- 节点自定义数据。



节点池相关的术语如下：

术语	描述
弹性伸缩组	一个弹性伸缩组包含ECS实例集合，这些实例被视为逻辑组以用于自动扩展和管理用途。节点池基于弹性伸缩组来进行节点管理，节点池中的资源（包括ECS实例和弹性伸缩组）在您的阿里云账户中运行。更多信息，请参见伸缩组概述。 说明 请勿直接通过弹性伸缩组进行配置及节点操作，请通过容器服务节点池进行节点配置及管理，否则将导致节点池部分功能不正常运作。
替换系统盘	通过替换节点系统盘的方式初始化节点，该节点的IaaS属性不发生改变，如节点名称、实例ID、IP等，但会删除节点系统盘上的数据，执行节点的重新初始化流程。额外挂载到该节点上的数据盘不受影响。 注意 请勿在节点系统盘中存储持久化数据，建议保存在数据盘中。
原地升级	与替盘相对应的一种升级方式，直接在原节点上更新替换所需组件。原地升级不会替换系统盘，也不会重新初始化节点，因此不会破坏原节点的数据。

节点池使用说明

在您使用节点池功能前，请阅读以下使用说明：

- 每个集群可以创建多个节点池，可创建的ACK节点池类型有节点池和托管节点池。关于节点池类型定义，请参见节点池类型。
- 创建集群时，指定的节点数和类型将成为默认节点池配置，默认节点池不可删除。通过未指定节点池的API的扩

容和添加节点方式添加的实例，都将归属于默认节点池。关于通过API扩容节点池和添加已有ECS节点的详细信息，请参见扩容节点池和添加已有实例至节点池。

> **说明** 存量的3可用区专有集群，由于创建集群时同时分配3个弹性伸缩组，所以系统会分别同步3个默认节点池。

- 通过节点池，可以在同一个集群中实现不同类型的节点混合部署管理。
  - 同一个集群中，允许创建不同操作系统（CentOS、Windows、AliyunLinux）的节点池。
  - 同一个集群中，允许创建不同容器运行时（安全沙箱、Docker）的节点池。
  - 同一个集群中，允许创建不同计费类型（按量付费、包年包月、抢占式实例）的节点池。
  - 同一个集群中，允许创建多个开启自动弹性伸缩的节点池。
- 每个节点池可以为节点指定不同的配置。
  - 资源相关：比如节点规格、交换机、系统盘和数据盘、安全组、登录方式等。
  - 节点配置：比如节点自定义数据、CPU管理策略、标签、污点（Taints）等。

节点池使用限制

- 目前节点池对Kubernetes版本为1.12.6-aliyun.1及以上的ACK集群开放。
- 每个ACK集群默认20个节点池，每个节点池可申请创建1000个节点（可提交工单增加配额）。

节点池计费

每个节点池对应一个弹性伸缩组实例，节点池本身不额外收费，但节点池使用的ECS实例等云资源由对应的云产品计费。

关于弹性伸缩组的计费详情，请参见费用说明。

节点池类型

ACK节点池类型分为节点池和托管节点池。

节点池类型	描述
节点池	节点池是集群中具有相同配置的一组节点，节点池可以包含一个或多个节点。节点池后端与弹性伸缩组实例一比一对应。当对节点池进行扩容和缩容时，ACK通过弹性伸缩服务下发扩容和移除节点的操作。您可以根据自己的需要创建和管理多个节点池。 **注意** 由于默认节点池中安装了部分系统组件，弹性伸缩时可能会造成集群功能的不稳定。如果您需要实现弹性伸缩功能，建议您另建节点池。
托管节点池	托管节点池是ACK全新推出的具有自动升级、节点自愈能力的免运维型节点池，为您提供统一的、全托管的节点生命周期管理。您无需关心节点运维操作，如节点组件版本更新，OS版本更新，CVE更新等，ACK会自行修复节点池内的故障节点。 更多信息，请参见托管节点池概述。

节点池和托管节点池的功能对比

对比项	节点池	托管节点池
手动扩缩容	✓	✓
可被删除	✓	✓

对比项	节点池	托管节点池
按量或者包年包月实例	✓	✓
自动故障修复	✗	✓
开启自动弹性伸缩	✓	✓
手动触发升级	✓	✓
升级Kubelet版本	✓	✓
升级Docker版本和CVE漏洞	✗	✓
在维护窗口内自动升级	✗	✓
升级方式为替换系统盘	✗	✓
密钥对登录集群	✓	✓
密码登录集群	✓	✗

说明

- 在删除节点池前，您需清空节点池内所有节点。
- 只能在创建节点池的时候开启自动弹性伸缩功能。开启了自动弹性伸缩功能的节点池有以下特性：
  - 不支持手动扩容。
  - 付费类型支持抢占式实例。
  - 弹性模式上除了普通CPU实例，还支持GPU实例和GPU共享实例。
  - 允许您关闭弹性伸缩配置，将弹性节点池切换为节点池（反向操作不允许）。

  更多信息，请参见节点自动伸缩。
- 非托管节点池升级方式为原地升级。

相关文档

- 管理节点池
- 托管节点池概述
- 节点自动伸缩

6.2.2. 管理节点池

您可以通过节点池管理集群中的一组节点资源，例如在节点池中统一管理节点的标签和污点。本文介绍如何基于容器服务ACK集群中的节点池功能创建节点池、调整节点池中的节点数量。

前提条件

已创建ACK集群，且集群Kubernetes版本大于1.9。具体操作，请参见创建Kubernetes托管版集群。

> 注意
>
> - 请确保您的集群可添加足够的节点数。如果您需要添加更多节点，请提交工单申请扩大配额。关于ACK集群的配额限制，请参见使用限制。
> - 添加已有云服务器时，请确保您的专有网络中的ECS实例已绑定一个弹性公网IP（EIP），或者相应VPC已经配置了NAT网关。您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。

背景信息

ACK节点池类型分为节点池和托管节点池，其中节点池分为默认节点池和自定义节点池。自定义节点池和托管节点池都可以开启自动弹性伸缩功能。更多信息，请参见节点池概述。

注意事项

由于默认节点池中安装了部分系统组件，弹性伸缩时可能会造成集群功能的不稳定。如果您需要实现弹性伸缩功能，建议您另建节点池。

创建节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面右上角，单击**创建节点池**。

   您还可以在**节点池**页面右上角，单击**创建托管节点池**或者**自动弹性伸缩配置**创建托管节点池或者具有弹性功能的节点池。
6. 在**创建节点池**页面，设置创建节点池的配置项。

   有关配置项的详细说明，请参见创建Kubernetes托管版集群。部分配置项说明如下：

参数	描述
期望节点数	设置节点池期望的节点数量。如不需要创建节点，可以填写为0。更多信息，请参见调整期望节点数。
付费类型	支持按量付费、包年包月、抢占式实例。更多信息，请参见实例购买方式。
自动伸缩	选中**开启自动弹性伸缩**开启具备弹性伸缩功能的节点池。更多信息，请参见节点自动伸缩。
操作系统	选择节点的操作系统，包括CentOS、Alibaba Cloud Linux、Windows。
公网IP	选中**公网IP**后，节点自动具备公网IPv4地址。您可以通过这个IP直接访问该节点。关于公网IP地址详情，请参见公网IP地址。 **注意** 如果选中**公网IP**，请勿在弹性节点池的配置中同时选中**EIP**，否则将导致节点弹出失败。

参数	描述
ECS标签	为ECS实例添加标签。
节点标签	为集群节点添加标签。 **说明** 选中**节点扩容后设置为不可调度**后，新添加的节点注册到集群时默认设置为不可调度。如果想打开调度选项，可以在节点列表中开启。
资源组	您可以指定节点池所扩容节点的资源组信息。默认为集群的资源组。
自定义安全组	支持选择**普通安全组**或**企业级安全组**。关于安全组的更多信息，请参见安全组概述。 **说明** ○ 自定义安全组为白名单功能，若您需要自定义安全组，请提交工单。 ○ 只能同时选择一种安全组（企业安全组或普通安全组）。 ○ 最多选择5个安全组。 ○ 编辑节点池时不支持更改节点池安全组。 ○ 创建节点池指定已有安全组时，默认不会为安全组配置集群访问规则。若您需要访问集群，请手动配置集群访问安全组规则。具体操作，请参见添加实例限制。 ○ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。

7. 单击**确认配置**。
   在**节点池**页面，如果节点池**状态**显示**初始化中**，则说明节点池正在创建中。创建完成后，**状态**显示为**已激活**。



调整期望节点数

您可以通过调整期望节点数，达到扩容或缩容节点池的目的。在指定节点池的期望节点数后，节点池会自动触发扩容或缩容，将节点数目维持在期望数量，无须人工干预。

- 通过期望节点数扩容节点：设定期望节点数大于当前节点池的节点数，系统将触发节点池扩容。推荐您通过增大期望节点数的方式进行扩容。这样在出现手动添加节点失败的情况下，系统也会自动进行重试以使得节点池现有节点数满足期望节点数。
- 通过期望节点数缩容节点：设定期望节点数小于当前节点池的节点数，系统将触发节点池的缩容并移除节点。

> **说明** 由于手动添加的节点无法自动移出伸缩组，因此如果对于手动添加节点（例如通过添加已有节点方式添加的节点），通过修改期望节点数无法自动移除，需要进行手动移除。关于手动移除的具体操作，请参见移除节点。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在目标节点池右侧的**操作**列中，单击**调整期望节点数**。
6. 授权CS获取云资源的权限。
   i. 单击AliyunOOSLifecycleHook4CSRole。



   ii. 在**云资源访问授权**页面，单击**同意授权**。
7. 在节点对话框中，设置配置项。

   有关配置项的详细说明，请参见创建节点池。以下为部分配置项说明：
   - **期望节点数**：期望的节点数量。单次操作可扩容最多500个节点。
   - **ECS标签**：为ECS实例添加标签。
   - **节点标签**：集群节点添加标签。

     > **说明**
     > - 选中**同步更新存量节点标签及污点**后，可以将此次修改的节点标签及污点同步更新到已有节点和新节点上，但存量节点上的您自行设置的标签和污点不会受影响。
     > - 选中**节点扩容后设置为不可调度**后，新添加的节点注册到集群时默认设置为不可调度。如果想打开调度选项，可以在节点列表中开启。

   - **污点（Taints）**：为节点添加污点。
8. 单击**提交**。
   - 在**节点池**页面，如果节点池**状态**显示**扩容中**，则说明节点池正在扩容中。扩容完成后，**状态**显示为**已激活**。
   - 在**节点池**页面，如果节点池**状态**显示**移除节点中**，则说明节点池正在缩容中。移除节点完成后，**状态**显示为**已激活**。

编辑节点池

在目标节点池右侧的**操作**列中，单击**编辑**，修改节点池配置。关于更多参数配置描述，请参见创建节点池。部分配置项的说明如下：

- **操作系统**：更新节点的操作系统镜像版本。
- **ECS标签**：为ECS实例添加标签。
- **节点标签**：集群节点添加标签。

  > **说明**
  > - 选中**同步更新存量节点标签及污点**后，可以将此次修改的节点标签及污点同步更新到已有节点和新节点上。，但存量节点上的您自行设置的标签和污点不会受影响。
  > - 选中**节点扩容后设置为不可调度**后，新添加的节点注册到集群时默认设置为不可调度。如果想打开调度选项，可以在节点列表中开启。

- **污点（Taints）**：为节点添加污点。

相关操作

在**节点池**页面可以进行以下操作：

- 在**名称**下拉列表右侧输入节点池名称，单击图标查找目标节点池。
- 在目标节点池右侧**操作**列，单击**详情**查看节点池详情。
- 单击目标节点池名称，可以进行以下操作：
  - 单击**基本信息**页签，查看集群、节点池信息、节点配置及自动弹性伸缩配置信息。
  - 单击**节点管理**页签，在节点列表中可移除选中节点。
  - 单击**节点管理**页签，然后在页面右上方，单击**导出**，可通过导出的CSV文件查看节点的各项信息。
  - 单击**节点管理**页签，选中**仅显示失败节点**可以筛选创建失败的节点。
  - 单击**伸缩活动**页签，查看最近节点实例伸缩活动的变更记录。

6.2.3. 调度应用至指定节点池

标签（labels）是Kubernetes中一个重要的概念。Service、Deployment和Pod都是通过labels实现关联。而在节点上，您也可以通过设置标签相关的策略使得Pod管理到对应标签的节点上。本文介绍如何调度应用到指定的节点池。

操作步骤

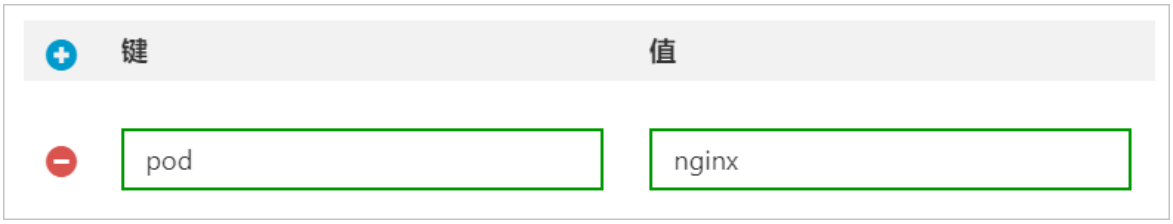
1. 给节点池设置节点标签。

   您可以在ACK中通过节点池管理集群中的一组节点资源。例如在节点池中统一管理节点的标签和污点。有关创建节点池的具体步骤，请参见管理节点池。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在目标集群右侧，单击**节点池**。
   iv. 在**节点池**页面，单击**创建节点池**。
   v. 在创建节点池配置页，单击**显示高级选项**，然后单击给节点设置**节点标签**。

   本文示例中添加的节点标签为pod: nginx。

   您也可以在目标节点池右侧单击**扩容**为节点更新或者添加标签。对于已开启自动伸缩的节点池，单击目标节点池右侧的**修改**为节点更新或者添加标签。



2. 为应用设置调度策略。

   上述步骤已经为节点池中的节点设置了pod:nginx的标签。您可以利用nodeSelector或者nodeAffinity保证您的应用限定在指定节点池上运行。具体操作步骤如下。

   - 设置nodeSelector。
     nodeSelector是Podspec中一个字段。您只需要将上述的pod: nginx标签填充到nodeSelector中。示例如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment-basic
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   nodeSelector:
    pod: nginx    #节点池中的标签添加到这里以保证您的应用只可以运行在节点池的节点上。
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

- 设置nodeAffinity。
  您还可以使用nodeAffinity实现应用的调度需求。nodeAffinity包含以下四种调度策略：

  ```
  - requiredDuringSchedulingIgnoredDuringExecution
  ```

  表示Pod必须部署到满足条件的节点上。如果没有满足条件的节点，调度操作就不停重试。其中IgnoreDuringExecution表示Pod部署之后运行时，如果节点标签发生了变化，不再满足Pod指定的条件，Pod也会继续运行。

  ```
  - requiredDuringSchedulingRequiredDuringExecution
  ```

  表示Pod必须部署到满足条件的节点上，如果没有满足条件的节点，调度操作就不停重试。其中RequiredDuringExecution表示Pod部署之后运行时，如果节点标签发生了变化，不再满足Pod指定的条件，则重新选择符合要求的节点。

  ```
  - preferredDuringSchedulingIgnoredDuringExecution
  ```

  表示优先部署到满足条件的节点上，如果没有满足条件的节点，就忽略这些条件，按照正常逻辑部署。

  ```
  - preferredDuringSchedulingRequiredDuringExecution
  ```

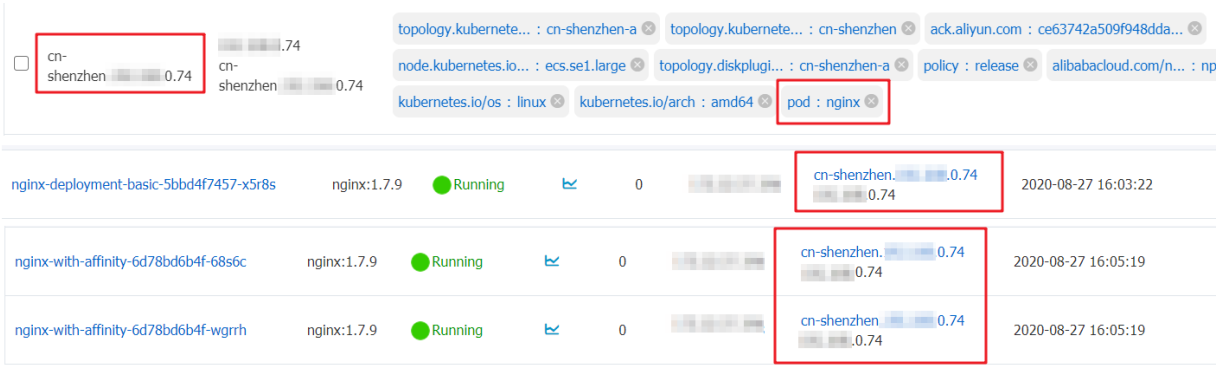
  表示优先部署到满足条件的节点上，如果没有满足条件的节点，就忽略这些条件，按照正常逻辑部署。其中RequiredDuringExecution表示如果后面节点标签发生了变化，满足了条件，则重新调度到满足条件的节点。

  本文示例使用requiredDuringSchedulingIgnoredDuringExecution策略保证应用一直运行在指定的节点池上。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-with-affinity
 labels:
  app: nginx-with-affinity
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx-with-affinity
 template:
  metadata:
   labels:
    app: nginx-with-affinity
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
       - key: pod
         operator: In    # 这里就相当于应用将运行在标了pod:nginx的节点上。
         values:
         - nginx
   containers:
   - name: nginx-with-affinity
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

执行结果

上述部署的应用全部被调度到了xxx.xxx.0.74的节点上，这个节点正是本文示例中在节点池中打了标签（pod: nignx）的节点。



6.2.4. 配置节点池内抢占式实例和存量实例的比例

通过在节点池内配置抢占式实例和存量实例的比例，在保证节点池内有稳定的存量实例的前提下，规划抢占式实例比例可降低成本。本文介绍如何基于容器服务Kubernetes版（ACK）集群中的节点池功能按规划配置抢占式实例和存量实例的比例。

前提条件

已创建Kubernetes集群，且集群版本大于1.9。具体操作，请参见创建Kubernetes托管版集群。

> **注意**
>
> - 请确保您的集群可添加足够的节点数。如果您需要添加更多节点，请提交工单申请扩大配额。关于ACK集群的配额限制，请参见使用限制。
> - 添加已有云服务器时，请确保您的专有网络中的ECS实例绑定一个弹性公网IP（EIP），或者相应VPC已经配置了NAT网关。您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。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 选择目标节点池或创建节点池。
   - 在目标节点池右侧**操作**列下单击**编辑**。
   - 在**节点池**页面右上角，单击**创建节点池**。具体操作，请参见管理节点池。
6. 在**创建节点池**或**编辑节点池**对话框中，选择**虚拟交换机**为多个虚拟交换机，选择**付费类型**为**抢占式实例**。
7. 单击**显示高级选项**，在**多可用区扩缩容策略**选择**成本优化策略**，并根据规划设置**按量实例所占比例（%）**和**允许按量实例补偿**。

   参数说明如下所示：

参数	描述	默认值
多可用区扩缩容策略	○ 优先级策略：选择配置顺序靠前的实例类型优先创建。 ○ 成本优化策略：选择成本最低的实例类型优先创建。 ○ 均衡分布策略：选择能达到可用区均衡分布的实例类型优先创建。	均衡分布策略
按量实例所占比例（%）	创建的实例中按量实例所占比例，计算时不包括组内最小按量实例数对应的按量实例。	20
允许按量实例补偿	如果因价格、库存等原因无法创建足够的抢占式实例，开启此开关表示允许自动尝试创建按量实例满足ECS实例数量要求。	开启

   > **说明**
   > 单击**确认配置**后：
   > - 不支持修改**多可用区扩缩容策略**的选择项。
   > - 若**多可用区扩缩容策略**选择为**成本优化策略**，可以修改**按量实例所占比例（%）**和**允许按量实例补偿**的配置。

8. 单击**确认配置**。

后续步骤

配置完成后，在**节点池**页面找到目标节点池，单击**操作**列中的**详情**，可查看**节点配置**列表中的**按量实例所占比例**。

6.2.5. 查看Spot实例到期状态

为了防止Spot实例（即抢占式实例）到期引发节点意外退出，阿里云容器服务ACK可通过ack-node-problem-detector组件获取实例即将释放的信息并通知给您。本文主要介绍如何查看Spot实例到期状态。

前提条件

创建Kubernetes集群且安装最新版本的ack-node-problem-detector组件。具体操作，请参见创建Kubernetes托管版集群。

- 若需要新建集群，请选中**安装node-problem-detector并创建事件中心**。
- 若使用已有集群，请确认ack-node-problem-detector组件为最新版本。具体操作，请参见管理组件。

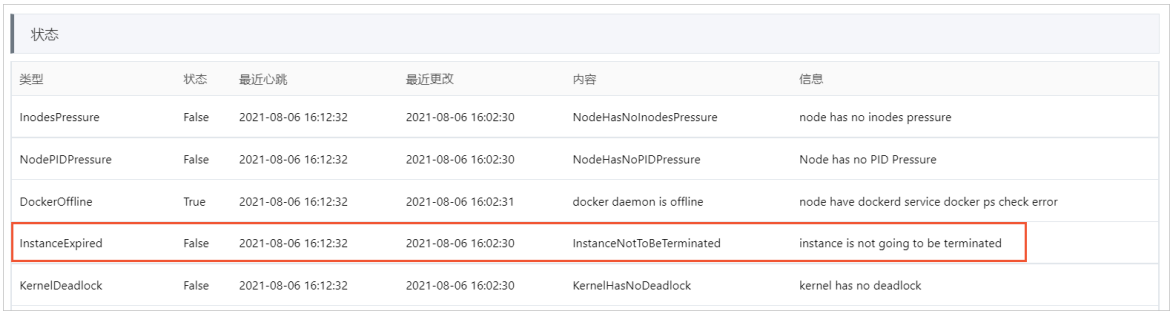
背景信息

在阿里云容器服务ACK中，ECS实例作为节点支持着集群以及集群上运行的服务。根据ECS实例创建时的策略，某些实例（例如Spot实例、包年包月实例）存在到期自动释放问题。若在实例自动释放时未做Pod驱逐、节点排水、节点替换等前置处理，可能影响集群服务运行情况，Master节点意外退出可能引发集群级别的故障。为了防止Spot实例到期引发的节点意外退出问题，可通过ack-node-problem-detector组件的InstanceExpired状态获取实例即将释放的信息。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点**页面，单击目标节点的名称或在目标节点右侧**操作**列选择**更多 > 详情**。
6. 在节点详情页面查看InstanceExpired类型的状态。

   在**状态**区域查看InstanceExpired类型的状态。



状态

类型	状态	最近心跳	最近更改	内容	信息
InodesPressure	False	2021-08-06 16:12:32	2021-08-06 16:02:30	NodeHasNoInodesPressure	node has no inodes pressure
NodePIDPressure	False	2021-08-06 16:12:32	2021-08-06 16:02:30	NodeHasNoPIDPressure	Node has no PID Pressure
DockerOffline	True	2021-08-06 16:12:32	2021-08-06 16:02:31	docker daemon is offline	node have dockerd service docker ps check error
InstanceExpired	False	2021-08-06 16:12:32	2021-08-06 16:02:30	InstanceNotToBeTerminated	instance is not going to be terminated
KernelDeadlock	False	2021-08-06 16:12:32	2021-08-06 16:02:30	KernelHasNoDeadlock	kernel has no deadlock

InstanceExpired的状态说明如下所示：

InstanceExpired的状态	说明
True	若**InstanceExpired**的状态为True，**内容**为**InstanceToBeTerminated**，表示Spot实例将到期，会被释放。
False	若**InstanceExpired**的状态为False，**内容**为**InstanceNotToBeTerminated**，表示Spot实例还未到期，可持续使用。
Unknown	表示插件运行错误，请提交工单解决。

若InstanceExpired的状态为**True**，可在**事件**区域看到以下的事件。



后续步骤

若InstanceExpired的状态为True，表示Spot实例将到期释放。若您需要继续使用该节点上的服务，请将应用调度到其他节点上。具体操作，请参见调度应用Pod至指定节点。

6.2.6. 查看节点池弹性强度

对节点池扩容时，可能由于实例库存不足、指定可用区过少或ECS实例规格不足等原因导致扩容失败。实例资源的实时供应会随着已有的虚拟交换机的消耗供给能力变弱，弹性能力也会变弱。因此可通过评估节点池弹性强度，帮助您调整集群资源。本文主要介绍如何查看节点池弹性强度。

功能介绍

阿里云容器服务ACK通过ECS提供的DescribeAvailableResource接口实时查询某一地域的可用资源及供应状态，其中规格状态分类StatusCategory中包含<可用区，实例规格>资源的补货情况，进而评估相应的弹性能力。更多信息，请参见DescribeAvailableResource。

资源状态（Status）	资源类别（StatusCategory）	状态说明
Available：有资源	WithStock：库存充足 （优选）。	当前有库存，会持续补货。
	ClosedWithStock：库存供应保障能力低。	当前有库存，但不会继续补货。
SoldOut：资源已售罄	WithoutStock：库存售罄，但会补充资源。	当前没库存，但会补货。临时断供。
	ClosedWithoutStock：库存售罄，且不补充资源。	当前没库存，也不会补货。当部分用户资源释放，状态会回到ClosedWithStock。

弹性强度可由可用资源池个数决定。可用资源池包括WithStock状态资源池及ClosedWithStock状态资源池，权重分别为1和0.5。

可用资源池个数=WithStock状态资源池个数*1+ClosedWithStock状态资源池个数*0.5

- 若可用资源池个数<2，则弹性强度为弱。
- 若可用资源池个数为[2, 4)，则弹性强度为中。
- 若可用资源池个数为[4, +)，则弹性强度为强。

节点池弹性强度评估示例

1. 在OpenAPI Explorer中直接运行DescribeAvailableResource接口，获取目标地域实例规格的可用资源及供应状态。
   关于OpenAPI Explorer中的DescribeAvailableResource接口，请参见DescribeAvailableResource接口。
   请求示例：

   ```
   https://ecs.aliyuncs.com/?Action=DescribeAvailableResource
   &RegionId=cn-beijing
   &DestinationResource=InstanceType
   &<公共请求参数>
   ```

   返回示例：

   ```
   {
   ```

```
  "RequestId": "735E42EB-197F-5331-8C83-68ECA68D****",
  "AvailableZones": {
    "AvailableZone": [
      {
        "Status": "Available",
        "StatusCategory": "WithStock",
        "ZoneId": "cn-beijing-k",
        "AvailableResources": {
          "AvailableResource": [
            {
              "Type": "InstanceType",
              "SupportedResources": {
                "SupportedResource": [
                  {
                    "Status": "Available",
                    "StatusCategory": "WithStock",
                    "Value": "ecs.s6-c1m2.small"
                  },
                  {
                    "Status": "Available",
                    "StatusCategory": "WithStock",
                    "Value": "ecs.hfr7.xlarge"
                  },
                  {
                    "Status": "Available",
                    "StatusCategory": "ClosedWithoutStock",
                    "Value": "ecs.sn1ne.2xlarge"
                  },
{
                    "Status": "Available",
                    "StatusCategory": "ClosedWithStock",
                    "Value": "ecs.hfg6.2xlarge"
                  }
                ]
              }
            }
          ]
        },
        "RegionId": "cn-beijing"
      },
      {
        "Status": "Available",
        "StatusCategory": "WithStock",
        "ZoneId": "cn-beijing-h",
        "AvailableResources": {
          "AvailableResource": [
            {
              "Type": "InstanceType",
              "SupportedResources": {
                "SupportedResource": [
                  {
                    "Status": "Available",
                    "StatusCategory": "WithStock",
                    "Value": "ecs.s6-c1m2.small"
                  },
                  {
                    "Status": "Available",
                    "StatusCategory": "WithStock"
```

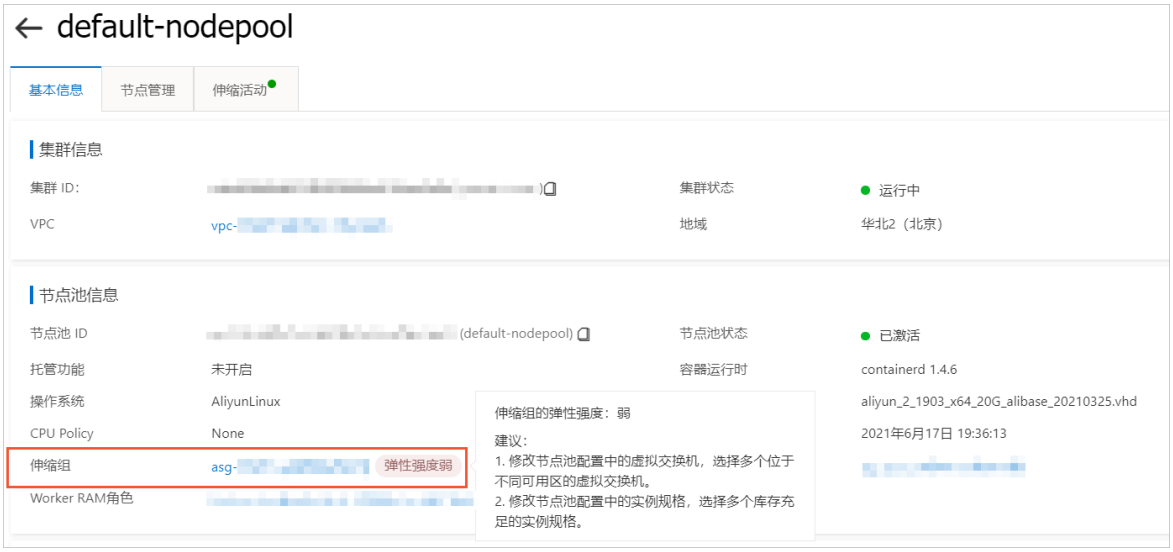
```
                "StatusCategory": "WithStock",
                "Value": "ecs.n4.small"
              },
              {
                "Status": "Available",
                "StatusCategory": "WithoutStock",
                "Value": "ecs.video-trans.26xhevc"
              },
                            {
                "Status": "Available",
                "StatusCategory": "ClosedWithStock",
                "Value": "ecs.r6.2xlarge"
              },
            ]
          }
        }
      ]
    },
    "RegionId": "cn-beijing"
  }
 ]
 }
}
```


从以上返回示例可得，WithStock状态资源池个数为4，ClosedWithStock状态资源池个数为2，无效资源池个数为2（1个WithoutStock状态资源池，1个ClosedWithoutStock状态资源池）。

2. 通过可用资源池个数评估节点池弹性强度。
   通过上一步的资源池个数可得，可用资源池个数为5，则弹性强度为强。

查看节点池的弹性强度

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面，单击目标集群名称或目标集群**操作**列的**详情**。
6. 在节点池详情页面，单击**基本信息**页签，在**节点池信息**区域的**伸缩组**后可看到当前弹性强度信息。



后续步骤

若当前弹性强度弱，建议进行以下操作：

- 修改节点池配置中的虚拟交换机，选择多个位于不同可用区的虚拟交换机。
- 修改节点池配置中的实例规格，选择多个库存充足的实例规格。

具体操作，请参见编辑节点池。

6.3. 托管节点池

6.3.1. 托管节点池概述

托管节点池是ACK全新推出的具有自动升级、节点自愈能力的免运维型节点池，为您提供统一的、全托管的节点生命周期管理。您无需关心节点运维操作，如节点组件版本更新、OS版本更新及OS镜像CVE修复等，ACK会自行修复节点池内的故障节点。

背景信息

节点池的相关概念，如下所示：

- **普通节点池**：为您提供管理一组同质节点的能力，同一个节点池内具有相同的节点配置，例如规格、标签（Label）、污点（Taint）。您可以自行运维普通节点池内节点。
- **托管节点池**：在普通节点池基础上，提供自动节点升级、故障修复等能力，从而实现托管的自动化节点运维。
- **替换系统盘**：通过替换节点系统盘（下文简称替盘）的方式初始化节点，该节点的IaaS属性不发生改变，如节点名称、实例ID、IP等，但会删除节点系统盘上的数据，执行节点的重新初始化流程。额外挂载到该节点上的数据盘不受影响。

  > **注意** 请勿在节点系统盘中存储持久化数据，建议保存在数据盘中。

- **原地升级**：与替盘相对应的一种升级方式，直接在原节点上更新替换所需组件。原地升级不会替换系统盘，也不会重新初始化节点，因此不会破坏原节点的数据。

适用场景

- 只关注上层应用开发，不希望主动运维worker节点。
- 需要快速响应OS镜像CVE安全漏洞。当新的CVE发布后，能够迅速升级，从而修补漏洞。
- 对底层节点的变更不敏感，业务Pod对迁移有较高的容忍度，更加关注业务的弹性而非不可变性。
- 需要升级节点上的Docker版本及OS镜像版本。

托管节点池与普通节点池对比

对比项	托管节点池	普通节点池
运维能力	托管于容器服务	用户自行管理
节点升级	• 维护窗口内自动升级。 • 也可以手动触发升级，支持升级OS、Docker版本、Kubelet版本、CVE漏洞等。 关于如何升级托管节点池，请参见升级托管节点池。 • 升级方式为替换系统盘。	• 仅支持手动触发升级。 • 仅支持升级Kubelet版本。 • 升级方式为原地升级。
自动故障修复	支持	不支持

对比项	托管节点池	普通节点池
密钥管理	仅支持密钥方式	同时支持密码和密钥方式
操作系统	支持以下操作系统： • ContainerOS • CentOS • Alibaba Cloud Linux • Windows	支持以下操作系统： • CentOS • Alibaba Cloud Linux • Windows

主要特征

- 您可以为同一个集群创建多个托管节点池，不同节点池具有不同的配置，从而实现不同规格节点的支持。
- 执行替盘轮转升级前会先尝试通过**cordon**命令将节点设置为不可调度，然后驱逐该节点上的Pod。如果超时15分钟后，Pod仍未被驱逐，容器服务将强制执行替盘操作。
- 托管节点池会监控节点的运行状态，如果节点超过10分钟未上报节点状态，或者状态为NotReady，容器服务会通过尝试重启节点来恢复。
- 您可以关闭托管节点池的自动升级功能。此时当有新的可用Docker及OS版本后，容器服务不会自动为节点池升级。
- 托管节点池的自动升级的时候需要设置维护窗口。容器服务会尽可能遵守维护窗口计划，在维护窗口内已经开始的升级会继续完成，未开始的升级计划会被暂停。

注意事项

- 托管节点池通过替换系统盘的方式执行节点升级操作，该操作会删除节点系统盘上的数据。数据盘内的数据不受影响，请勿在系统盘上存储任何持久化数据。
- 托管节点池在替盘升级前会执行节点下线与排水，因此会造成Pod的重启、长链接的中断。
- 托管节点池会尝试重启故障节点来修复可能的故障，因此节点上的Pod会被重启。

6.3.2. 管理托管节点池

托管节点池具有自动升级和节点自愈能力，您无需关心节点的运维操作，从而帮助您实现统一和全托管的节点生命周期管理。本文介绍如何创建、克隆和删除托管节点池，以及调整节点池中的节点数量。

前提条件

- 已创建一个ACK Pro版集群，请参见创建ACK Pro版集群。
- 请确保您集群的Kubernetes版本大于1.9。

注意事项

- 托管节点池通过替换系统盘的方式执行节点升级操作，该操作会删除节点系统盘上的数据。数据盘内的数据不受影响，请勿在系统盘上存储任何持久化数据。
- 托管节点池在替盘升级前会执行节点下线与排水，因此会造成Pod的重启、长链接的中断。
- 托管节点池会尝试重启故障节点来修复可能的故障，因此节点上的Pod会被重启。

创建托管节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面右上角，单击**创建托管节点池**。

6. 在**创建托管节点池**对话框，设置托管节点池的配置项。

   有关配置项的详细说明，请参见创建ACK Pro版集群。部分配置项说明如下：

   - 维护窗口：单击**设置**后，在**维护窗口**对话框中为托管节点池设置执行自动升级的时间。
   - 自动伸缩：设置是否开启节点的自动伸缩功能。更多信息，请参见节点自动伸缩。
   - 期望节点数：设置节点池的期望节点数量。如不需要创建节点，可以填写为0。
   - 操作系统：选择节点的操作系统，包括ContainerOS、CentOS、Alibaba Cloud Linux、Windows。

     > **说明** ContainerOS仅在容器运行时为Containerd时支持。

   - 节点标签：您可以为集群节点添加标签。
   - ECS标签：您可以为ECS实例添加标签。
   - 自定义资源组：您可以指定节点池所扩容节点的资源组信息。

7. 单击**确认配置**。

   在**节点池**页面，如果节点池**状态**显示**初始化中**，则说明节点池正在创建中。创建完成后，**状态**显示为**已激活**。

升级托管节点池

当托管节点池有可升级的操作系统时，您可以升级托管节点池。

1. 在目标节点池右侧的**操作**列下，单击**升级**。
2. 在**升级节点池**对话框中，设置以下参数。

参数	描述
升级版本	默认选中**系统镜像**和**容器运行时**： - **系统镜像**：选中后，升级节点池中的所有节点的系统镜像至新版本。 - **容器运行时**：选中后，升级节点池中的所有节点的容器运行时至新版本。
最大不可用节点数	表示最多处于不可用状态的节点数。
额外节点数	表示升级节点池前，集群会自动扩容相应数量的节点，待升级完成后自动删除这几个节点。目的是保证升级的过程中有足够的节点供排水使用。

> **说明** **最大不可用节点数**和**额外节点数**之和等于最大可以并行升级的数量。

调整期望节点数

您可以通过调整期望节点数，达到扩容或缩容节点池的目的。在指定节点池的期望节点数后，节点池会自动触发扩容或缩容，将节点数目维持在期望数量，无须人工干预。

- 通过期望节点数扩容节点：设定期望节点数大于当前节点池的节点数，系统将触发节点池扩容。推荐您通过增大期望节点数的方式进行扩容。这样在出现手动添加节点失败的情况下，系统也会自动进行重试以使得节点池现有节点数满足期望节点数。
- 通过期望节点数缩容节点：设定期望节点数小于当前节点池的节点数，系统将触发节点池的缩容并移除节点。

> **说明** 由于手动添加的节点无法自动移出伸缩组，因此如果对于手动添加节点（例如通过添加已有节点方式添加的节点），通过修改期望节点数无法自动移除，需要进行手动移除。关于手动移除的具体操作，请参见移除节点。

1. 在目标节点池右侧的**操作**列中，单击**调整期望节点数**。

2. 授权CS获取云资源的权限。
   i. 单击AliyunOOSLifecycleHook4CSRole。



   ii. 在**云资源访问授权**页面，单击**同意授权**。
3. 在节点池对话框的**期望节点数**中，输入期望的节点数量，然后单击**提交**。

> **说明** 如果您需要修改节点池配置，展开**修改节点池配置**。关于参数的详细信息，请参见创建托管节点池。

   - 在**节点池**页面，如果节点池**状态**显示**扩容中**，则说明节点池正在扩容中。扩容完成后，**状态**显示为**已激活**。
   - 在**节点池**页面，如果节点池**状态**显示**移除节点中**，则说明节点池正在缩容中。移除节点完成后，**状态**显示为**已激活**。

克隆托管节点池

通过节点池的克隆功能，您可以基于已有的节点池，克隆一个具有相同配置的节点池。

1. 在目标节点池右侧的**操作**列中，单击**克隆**。
2. 在克隆节点池对话框中，输入新克隆节点池的名称并设置相关参数，单击**提交**。

   在**节点池**页面，节点池的**状态**刚开始显示为**初始化中**。克隆完成后，**状态**会显示为**已激活**。

删除托管节点池

为了防止误删除导致业务中断，删除节点池前需要清空节点池中的节点。

1. 在节点池列表页面，单击目标节点池名称。
2. 在**节点配置**区域的**节点管理**页签中，选中所有节点，单击**移除节点**。
3. 在**移除节点**对话框中，可选中**同时释放ECS**和**自动排空节点（drain）**，单击**确定**。
   - **同时释放ECS**：
     - 仅释放按量付费类型的ECS实例，不被释放的ECS会继续收费。
     - 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，请参见包年包月转按量付费。
     - 若不选择**同时释放ECS**，该节点所在的ECS实例会继续计费。
   - **自动排空节点（drain）**：把待移除节点上的Pod转移到其他节点。请确保集群其他节点的资源充足。
     您还可以通过执行命令 `kubectl drain node-name [options]` 的方式把待移除节点上的Pod转移到其他节点。

> **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。例如，*cn-hangzhou.i-xxx*。
> - options 为该命令的可选配置参数。例如，可配置为 --force --ignore-daemonsets --delete-local-data 。更多信息，请执行命令 kubectl drain --help 查看。

4. 返回节点池列表页面，在目标节点池右侧的**操作**列中，单击**删除**。
5. 在**节点池删除**对话框中，单击**确定**。

开启和关闭托管节点池

您可以通过开启节点池的托管功能将默认节点池、自定义节点池、弹性伸缩节点池（开启自动伸缩的自定义节点池）转换成托管节点池，也可以关闭托管节点池的托管功能。开启和关闭托管节点池操作如下。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在目标节点池右侧的**操作**列中，进行以下操作。
   - 单击**开启托管**将节点池转换成托管节点池。
   - 单击**关闭托管**将托管节点池转换成其他类型节点池。
   关闭托管节点池的托管功能后，原本就是托管节点池类型的托管节点池将被转换成自定义节点池，由其他类型节点池转换成托管节点池类型的托管节点池将被转换成其原本的节点池类型。例如，默认节点池开启托管功能转换成托管节点池后，然后再关闭托管功能，托管节点池将被转换成默认节点池。

6.3.3. 调度应用至指定节点池

标签（labels）是Kubernetes中一个重要的概念。Service、Deployment和Pod都是通过labels实现关联。而在节点上，您也可以通过设置标签相关的策略使得Pod管理到对应标签的节点上。本文介绍如何调度应用到指定的节点池。

操作步骤

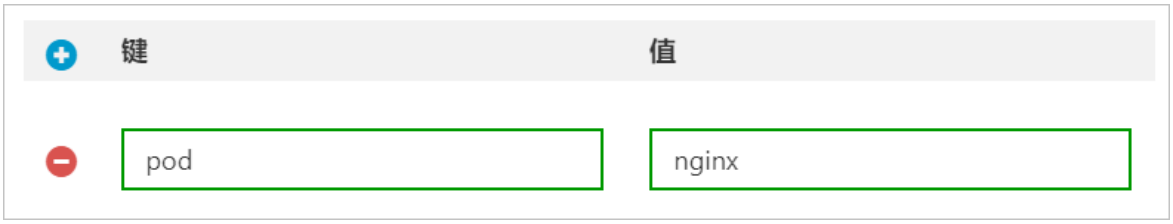
1. 给节点池设置节点标签。

   您可以在ACK中通过节点池管理集群中的一组节点资源。例如在节点池中统一管理节点的标签和污点。有关创建节点池的具体步骤，请参见管理节点池。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在目标集群右侧，单击**节点池**。
   iv. 在**节点池**页面，单击**创建节点池**。
   v. 在创建节点池配置页，单击**显示高级选项**，然后单击 给节点设置**节点标签**。

      本文示例中添加的节点标签为pod: nginx。

   您也可以在目标节点池右侧单击**扩容**为节点更新或者添加标签。对于已开启自动伸缩的节点池，单击目标节点池右侧的**修改**为节点更新或者添加标签。



2. 为应用设置调度策略。

   上述步骤已经为节点池中的节点设置了pod:nginx的标签。您可以利用nodeSelector或者nodeAffinity保证您的应用限定在指定节点池上运行。具体操作步骤如下。

   - 设置nodeSelector。
     nodeSelector是Podspec中一个字段。您只需要将上述的pod: nginx标签填充到nodeSelector中。示例如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment-basic
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   nodeSelector:
    pod: nginx    #节点池中的标签添加到这里以保证您的应用只可以运行在节点池的节点上。
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

   - 设置nodeAffinity。
     您还可以使用nodeAffinity实现应用的调度需求。nodeAffinity包含以下四种调度策略：

     - requiredDuringSchedulingIgnoredDuringExecution

     表示Pod必须部署到满足条件的节点上。如果没有满足条件的节点，调度操作就不停重试。其中IgnoreDuringExecution表示Pod部署之后运行时，如果节点标签发生了变化，不再满足Pod指定的条件，Pod也会继续运行。

     - requiredDuringSchedulingRequiredDuringExecution

     表示Pod必须部署到满足条件的节点上，如果没有满足条件的节点，调度操作就不停重试。其中RequiredDuringExecution表示Pod部署之后运行时，如果节点标签发生了变化，不再满足Pod指定的条件，则重新选择符合要求的节点。

     - preferredDuringSchedulingIgnoredDuringExecution

     表示优先部署到满足条件的节点上，如果没有满足条件的节点，就忽略这些条件，按照正常逻辑部署。

```
- preferredDuringSchedulingRequiredDuringExecution
```

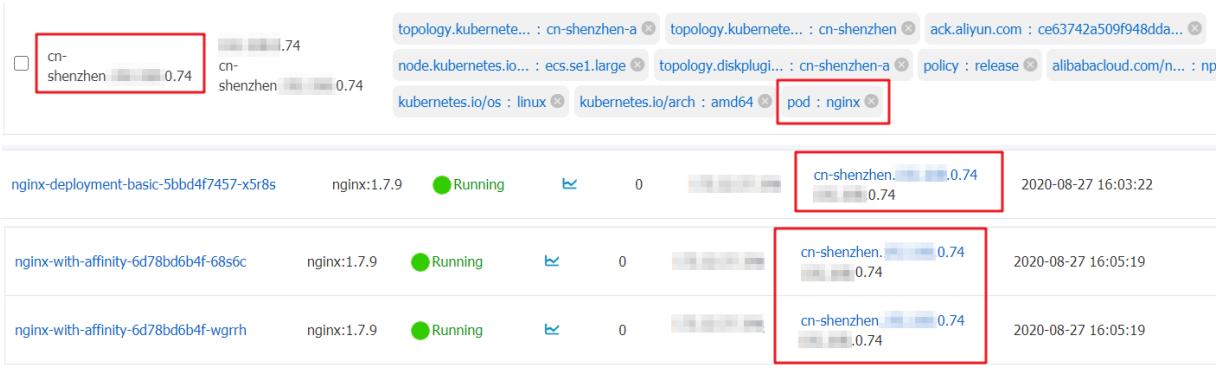
表示优先部署到满足条件的节点上，如果没有满足条件的节点，就忽略这些条件，按照正常逻辑部署。其中RequiredDuringExecution表示如果后面节点标签发生了变化，满足了条件，则重新调度到满足条件的节点。

本文示例使用requiredDuringSchedulingIgnoredDuringExecution策略保证应用一直运行在指定的节点池上。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-with-affinity
 labels:
  app: nginx-with-affinity
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx-with-affinity
 template:
  metadata:
   labels:
    app: nginx-with-affinity
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: pod
          operator: In    # 这里就相当于应用将运行在标了pod:nginx的节点上。
          values:
          - nginx
   containers:
   - name: nginx-with-affinity
     image: nginx:1.7.9
     ports:
     - containerPort: 80
```

执行结果

上述部署的应用全部被调度到了xxx.xxx.0.74的节点上，这个节点正是本文示例中在节点池中打了标签（pod:nignx）的节点。



6.4. 节点资源预留策略

ACK需要占用一定的节点资源来运行相关组件（例如kubelet、kube-proxy、Terway、Container Runtime等），从而使节点作为集群的一部分来运行。这会造成节点的资源总数与ACK集群中可分配的资源数之间存在差异。本文介绍ACK的节点资源预留策略、相关注意事项，以便在部署应用时合理设置Pod的请求资源量和限制资源量。

查询节点可分配资源

执行以下命令，查看节点的资源总量和可分配资源。

```
kubectl describe node [NODE_NAME] | grep Allocatable -B 7 -A 6
```

预期输出：

```
Capacity:
 cpu:                4          #节点的CPU总核数。
 ephemeral-storage:  123722704Ki    #节点的临时存储总量，单位KiB。
 hugepages-1Gi:      0
 hugepages-2Mi:      0
 memory:             7925980Ki      #节点的内存总量，单位KiB。
 pods:               64
Allocatable:
 cpu:                3900m          #节点可分配的CPU核数。
 ephemeral-storage:  114022843818    #节点可分配的临时存储，单位KiB。
 hugepages-1Gi:      0
 hugepages-2Mi:      0
 memory:             5824732Ki      #节点可分配的内存，单位KiB。
 pods:               64
```

计算节点可分配资源

可分配资源的计算公式：可分配资源（Allocatable）= 总资源（Capacity）-预留资源（Reserved）-驱逐阈值（Eviction-Threshold）

公式说明：

- 总资源对应查询节点命令输出中的 `Capacity` 字段。
- 关于预留资源的相关信息，请参见资源预留策略。
- 关于驱逐阈值的相关信息，请参见节点压力驱逐。

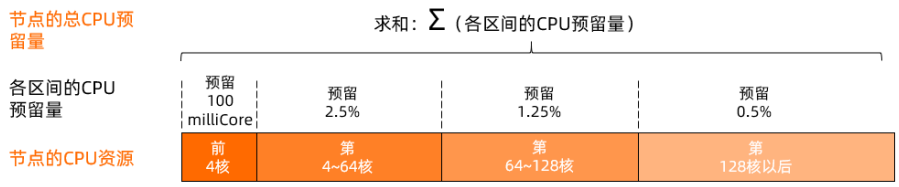
资源预留策略

节点的预留资源量通常要考虑以下因素：

- 由于规格较高的ECS节点通常会运行更多的Pod，为了保证节点的性能，ACK会为Kubernetes组件预留更多资源。
- 由于Windows节点需要使用额外的资源来运行Windows操作系统和Windows Server相关组件，Windows节点通常会比Linux节点需要更多的预留资源。更多信息，请参见创建Windows节点池。

ACK会根据CPU和内存所在的不同区间来计算预留的资源量，节点的总预留资源量等于各区间的预留资源量之和。

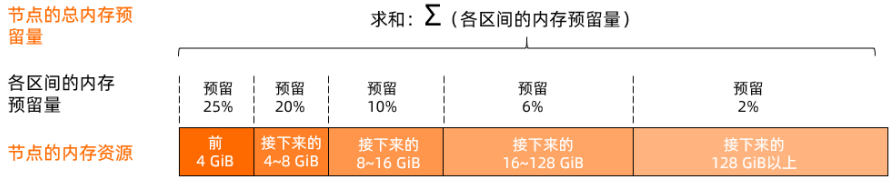
- CPU：计算节点的总CPU预留量示意图如下。



以32核的节点为例，总的CPU预留量计算如下：

100+（32000-4000）×2.5%=800 milliCore

- 内存：计算节点的总内存预留量示意图如下所示。



以256 GiB内存的节点为例，总的内存预留量计算如下：
4×25%+（8-4）×20%+（16-8）×10%+（128-16）×6%+（256-128）×2%=11.88 GiB

资源预留注意事项

- 该预留策略仅适用于1.18.8版本及以上的集群。
- 该预留策略对新添加的节点自动生效，无需手动配置。
- 出于稳定性考虑，该预留策略不会对已有节点自动生效。因为该资源预留的计算方式可能会造成节点的可分配资源变少，对于资源水位较高的节点，可能会触发节点驱逐。
- 如果您希望对已有节点应用该资源预留策略，您可以通过容器服务管理控制台将该节点移除出集群，然后重新添加节点，新添加的节点会默认执行该资源预留策略。

6.5. 节点管理FAQ

本文主要为您介绍节点管理常见问题。

- 如何在已有集群的GPU节点上手动升级Kernel?
- 修复GPU节点容器启动问题
- 添加Kubernetes集群节点的常见问题
- 移除节点时提示”drain-node job execute timeout”错误
- 如何更改ACK集群中Worker节点的主机名称?

如何在已有集群的GPU节点上手动升级Kernel?

下面为您介绍如何在已有集群的GPU节点上手动升级Kernel。

> **说明** 当前kernel版本低于 3.10.0-957.21.3 。
> 请确认需要升级的目标kernel版本，并谨慎操作。
> 本文提供方案并不涉及kernel升级，仅针对在kernel升级的前提下对应的Nvidia驱动升级。

1. 通过kubectl工具连接集群。
2. 将GPU节点设置为不可调度（本例以节点 cn-beijing.i-2ze19qyi8votgjz12345为例）。

```
kubectl cordon cn-beijing.i-2ze19qyi8votgjz12345
node/cn-beijing.i-2ze19qyi8votgjz12345 already cordoned
```

3. 将要升级驱动的GPU节点进行排水。

```
kubectl drain cn-beijing.i-2ze19qyi8votgjz12345 --grace-period=120 --ignore-daemonsets=true
node/cn-beijing.i-2ze19qyi8votgjz12345 cordoned
WARNING: Ignoring DaemonSet-managed pods: flexvolume-9scb4, kube-flannel-ds-r2qmh, kube-proxy-worker-l62sf, logtail-ds-f9vbg
pod/nginx-ingress-controller-78d847fb96-5fkkw evicted
```

4. 卸载当前的nvidia-driver。

> **说明** 本步骤中卸载的是版本为384.111的驱动包，如果您的驱动版本不是384.111，则需要在Nvidia官网下载对应的驱动安装包，并将本步骤中的 `384.111` 替换成您实际的版本。

i. 登录到该GPU节点，通过 `nvidia-smi` 查看驱动版本。

```
nvidia-smi -a | grep 'Driver Version'
Driver Version                  : 384.111
```

ii. 下载Nvidia驱动安装包。

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

> **说明** 需要在安装包中卸载Nvidia。

iii. 卸载当前Nvidia驱动。

```
chmod u+x NVIDIA-Linux-x86_64-384.111.run
./NVIDIA-Linux-x86_64-384.111.run --uninstall -a -s -q
```

5. 升级Kernel。

   您可以根据需要升级Kernel。

6. 重启GPU机器。

```
reboot
```

7. 重新登录GPU节点，安装对应的kernel devel。

```
yum install -y kernel-devel-$(uname -r)
```

8. 请到Nvidia官网下载和安装您需要的nvidia驱动， 本文以410.79为例。

```
cd /tmp/
curl -O https://cn.download.nvidia.cn/tesla/410.79/NVIDIA-Linux-x86_64-410.79.run
chmod u+x NVIDIA-Linux-x86_64-410.79.run
sh ./NVIDIA-Linux-x86_64-410.79.run -a -s -q
warm up GPU
nvidia-smi -pm 1 || true
nvidia-smi -acp 0 || true
nvidia-smi --auto-boost-default=0 || true
nvidia-smi --auto-boost-permission=0 || true
nvidia-modprobe -u -c=0 -m || true
```

9. 查看 */etc/rc.d/rc.local*，确认其中是否包含以下配置，如果没有请手动添加。

```
nvidia-smi -pm 1 || true
nvidia-smi -acp 0 || true
nvidia-smi --auto-boost-default=0 || true
nvidia-smi --auto-boost-permission=0 || true
nvidia-modprobe -u -c=0 -m || true
```

10. 重启kubelet和docker。

```
service kubelet stop
service docker restart
service kubelet start
```

11. 将这个GPU节点重新设置为可调度。

```
kubectl uncordon cn-beijing.i-2ze19qyi8votgjz12345
node/cn-beijing.i-2ze19qyi8votgjz12345 already uncordoned
```

12. 在GPU节点上的device plugin pod验证版本。

```
kubectl exec -n kube-system -t nvidia-device-plugin-cn-beijing.i-2ze19qyi8votgjz12345 nvidia-smi
Thu Jan 17 00:33:27 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 P100-PCIE...  On   | 00000000:00:09.0 Off |                    0 |
| N/A  27C    P0    28W / 250W |    0MiB / 16280MiB |    0%     Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
| GPU     PID   Type   Process name                             Usage      |
|=============================================================================|
| No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

> **说明** 如果通过 `docker ps` 命令，发现GPU节点没有容器被启动，请参见修复GPU节点容器启动问题。

修复GPU节点容器启动问题

在某些特定Kubernetes版本中的GPU节点上，重启Kubelet和Docker时，发现没有容器被启动。

```
service kubelet stop
Redirecting to /bin/systemctl stop kubelet.service
service docker stop
Redirecting to /bin/systemctl stop docker.service
service docker start
Redirecting to /bin/systemctl start docker.service
service kubelet start
Redirecting to /bin/systemctl start kubelet.service
docker ps
CONTAINER ID     IMAGE         COMMAND         CREATED         STATUS         PORTS         NAMES
```

执行以下命令，查看Docker的Cgroup Driver。

```
docker info | grep -i cgroup
Cgroup Driver: cgroupfs
```

此时发现的Cgroup Driver类型是cgroupfs。
您可以按照以下操作，修复该问题。

1. 备份*/etc/docker/daemon.json*，完成后，执行以下命令更新*/etc/docker/daemon.json*。

```
cat >/etc/docker/daemon.json <<-EOF
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  },
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "10"
  },
  "oom-score-adjust": -1000,
  "storage-driver": "overlay2",
  "storage-opts":["overlay2.override_kernel_check=true"],
  "live-restore": true
}
EOF
```

2. 执行以下命令，重启Docker和Kubelet。

```
service kubelet stop
Redirecting to /bin/systemctl stop kubelet.service
service docker restart
Redirecting to /bin/systemctl restart docker.service
service kubelet start
Redirecting to /bin/systemctl start kubelet.service
```

3. 执行以下命令，确认Docker的Cgroup Driver的类型为systemd。

```
docker info | grep -i cgroup
Cgroup Driver: systemd
```

如何更改ACK集群中Worker节点的主机名称？

集群创建完成后，不支持自定义Worker节点的主机名称，但是您可以通过节点池的节点命名规则来修改Worker节点的主机名称。

> **说明** 创建集群时，您可以在**自定义节点名称**参数中定义Worker节点的主机名称。具体操作，请参见创建Kubernetes托管版集群。

1. 移除节点。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   iv. 在**节点**页面单击目标节点右侧**操作**列下的**更多 > 移除**。
   v. 在弹出的对话框中选中**我已了解上述说明，确认移除节点**，然后单击**确定**。
2. 将移除的节点再添加到节点池。具体操作，请参见手动添加节点。

   添加的节点将根据节点池的节点命名规则进行命名。

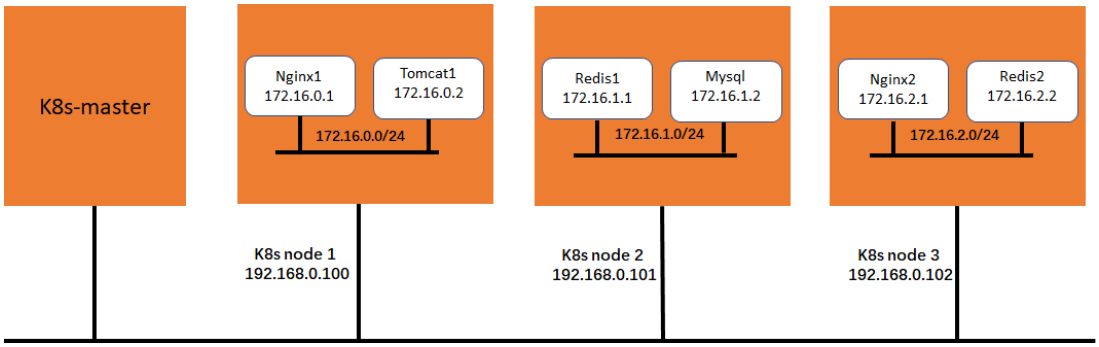
7.网络

7.1. 网络概述

阿里云容器服务Kubernetes版ACK集成Kubernetes网络、阿里云VPC、阿里云SLB，提供稳定高性能的容器网络。本文介绍ACK集群网络及阿里云网络底层基础设施的重要概念，如容器网络CNI、Service、Ingress、提供服务发现能力的DNS等。您可以通过了解这些概念，更合理地设计应用部署模型和网络访问的方式。

容器网络CNI

容器网络模型



容器化应用会在同一个节点上部署多个业务，而每个业务都需要自己的网络空间。为避免与其他业务网络冲突，需要Pod有自己独立的网络空间，而Pod中应用需要和其他网络进行通信，就需要Pod能够跟不同的网络互相访问。可见容器网络特点如下：

- 每个Pod都拥有自己独立的网络空间和IP地址。不同Pod的应用可以监听同样的端口而不冲突。
- Pod可以通过各自的IP地址互相访问。
  集群中Pod可以通过它独立的IP地址与其他应用互通：
  - 同一个集群中，Pod之间相互访问。
  - 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** Pod能直接访问同一个VPC下的ECS和同一个VPC下的ECS能直接访问Pod的前提是您需要正确设置安全组规则。关于如何设置安全组规则，请参见添加安全组规则。

在ACK中，您可以通过两种网络模型实现上述容器网络的能力，分别是Flannel网络模式和Terway网络模式。这两种网络模式采用不同的网络模型，其特点如下：

关于阿里云专有网络VPC网段和Kubernetes网段关系，请参见专有网络VPC网段和Kubernetes网段关系。

关于如何选择合适的网络模式，请参见使用Terway网络插件。

Terway网络模式 Flannel网络模式

Service

由于云原生的应用，通常需要敏捷的迭代和快速的弹性，且单一容器和其相关的网络资源的生命非常短暂，所以需要固定的访问地址，以及自动负载均衡实现快速的业务弹性。ACK采用Service方式为一组容器提供固定的访问入口，并对这一组容器做负载均衡。实现原理如下：

- 创建Service对象时，ACK会分配一个相对固定的Service IP地址。
- 通过字段selector选择一组容器，以将这个Service IP地址和端口负载均衡到这一组容器IP和端口上。

Service网络支持以下多种模式分别对接不同来源和类型的客户端的访问：

- *ClusterIP*
  ClusterIP类型的Service用于集群内部的应用间访问，如果您的应用需要暴露到集群内部提供服务，需使用ClusterIP类型的Service暴露。

  > **说明** 创建Service时默认的Service类型为ClusterIP。

- *NodePort*
  NodePort类型的Service将集群中部署的应用向外暴露，通过集群节点上的一个固定端口暴露出去，这样在集群外部就可以通过节点IP和这个固定端口来访问。

- *LoadBalancer*
  LoadBalancer类型的Service同样是将集群内部部署的应用向外暴露，不过它是通过阿里云的负载均衡进行暴露的，相对于NodePort方式，有更高的可用性和性能。关于如何通过LoadBalancer类型Service暴露应用，请参见通过使用已有SLB的服务公开应用和通过使用自动创建SLB的服务公开应用。

- *Headless Service*
  Headless Service类型的Service是在Service属性中指定clusterIP字段为None类型。采用Headless Service类型后，Service将没有固定的虚拟IP地址，客户端访问Service的域名时会通过DNS返回所有的后端Pod实例的IP地址，客户端需要采用DNS负载均衡来实现对后端的负载均衡。

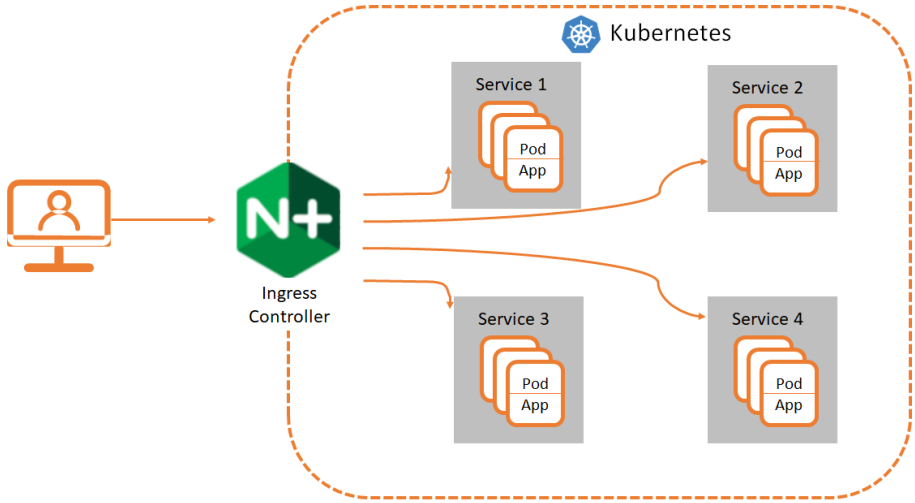
- *ExternalName*
  ExternalName类型的Service将集群外部的域名映射到集群内部的Service上，例如将外部的数据库域名映射到集群内部的Service名，那么就能在集群内部通过Service名直接访问。

关于Service的负载均衡配置需要注意内容，请参见Service的负载均衡配置注意事项。

Ingress

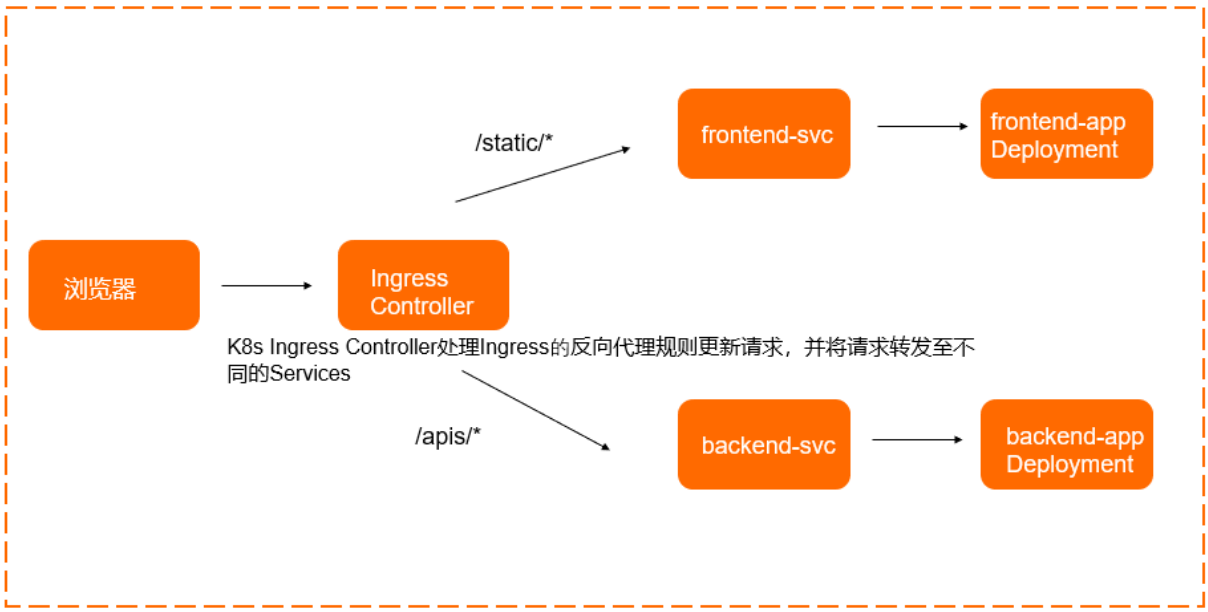
在ACK集群中，与Service的4层负载均衡不同，Ingress是集群内Service对外暴露7层的访问接入点。您可以通过Ingress资源来配置不同的7层的转发规则，例如通过域名或者访问路径来路由到不同的Service上，从而达到7层的负载均衡作用。更多信息，请参见Ingress概述。

Ingress和Service关系



示例

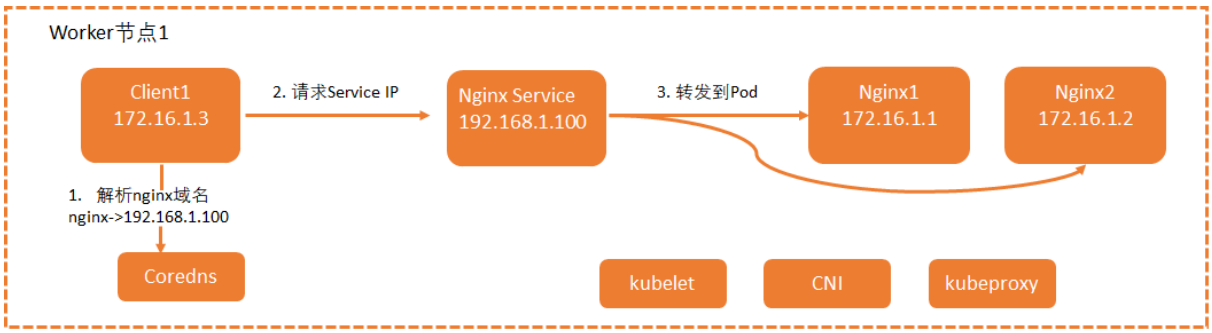
常见的前后端分离的架构方式中，前后端的访问地址分别使用不同的访问路径。对应这种场景，可以采用Ingress，根据7层的访问路径负载到不同的应用实例上。



服务发现DNS

ACK使用DNS来实现应用的服务发现能力，例如客户端应用可以通过Service的服务名解析出它的ClusterIP访问、可以通过StatefulSet的Pod名解析出Pod的IP地址。采用DNS服务发现的能力让集群中应用间的调用与IP地址和部署环境相解耦。

集群的Coredns会将Service名自动转换成对应的Service的IP地址，来实现不同部署环境中同样的访问入口。关于集群DNS组件的使用和调优详细信息，请参见ACK集群内DNS原理和配置说明。

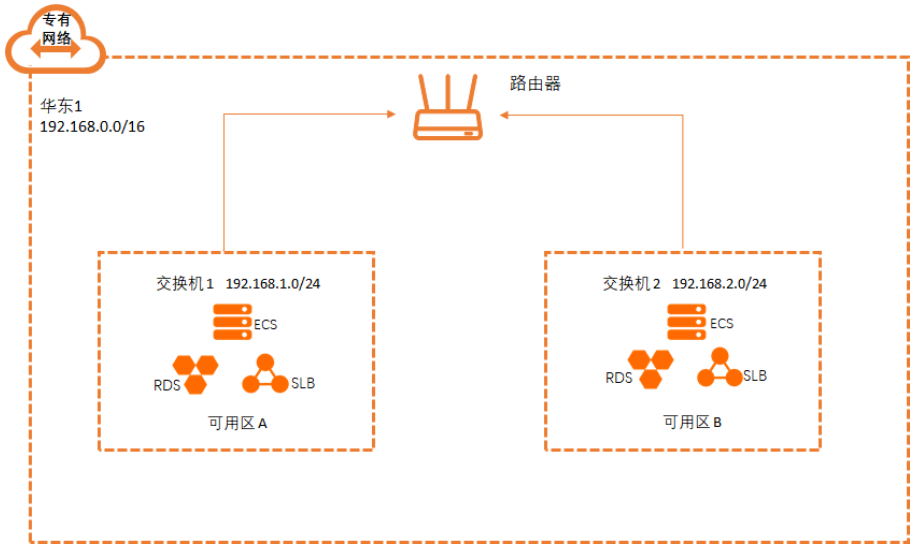


网络底层基础设施

- *VPC*

  阿里云专有网络VPC（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。您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都由一个路由器、至少一个私网网段和至少一个交换机组成。



- 私网网段
  在创建专有网络和交换机时，您需要以CIDR地址块的形式指定专有网络使用的私网网段。
  您可以使用下表中标准的私网网段及其子网作为VPC的私网网段，也可以使用自定义地址段作为VPC的私网网段。更多信息，请参见网络规划。

网段	说明
192.168.0.0/16	可用私网IP数量（不包括系统保留地址）：65,532
172.16.0.0/12	可用私网IP数量（不包括系统保留地址）：1,048,572
10.0.0.0/8	可用私网IP数量（不包括系统保留地址）：16,777,212
自定义地址段	除100.64.0.0/10、224.0.0.0/4、127.0.0.0/8、169.254.0.0/16及其子网外的自定义地址段

- 路由器
  路由器（vRouter）是专有网络的枢纽。作为专有网络中重要的功能组件，它可以连接专有网络内的各个交换机，同时也是连接专有网络和其他网络的网关设备。每个专有网络创建成功后，系统会自动创建一个路由器。每个路由器关联一张路由表。
  更多信息，请参见路由表概述。
- 交换机
  交换机（vSwitch）是组成专有网络的基础网络设备，用来连接不同的云资源。创建专有网络后，您可以通过创建交换机为专有网络划分一个或多个子网。同一专有网络内的不同交换机之间内网互通。您可以将应用部署在不同可用区的交换机内，提高应用的可用性。
  更多信息，请参见交换机。

- *SLB*
  阿里云负载均衡SLB（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。

相关文档

- 管理配额
- 使用网络策略Network Policy
- Kubernetes集群网络规划
- 使用Terway网络插件
- 创建安全组

7.2. Kubernetes集群网络规划

在创建ACK Kubernetes集群时，您需要指定专有网络VPC、虚拟交换机、Pod网络CIDR（地址段）和Service CIDR（地址段）。因此建议您提前规划ECS地址、Kubernetes Pod地址和Service地址。本文将介绍阿里云专有网络VPC环境下ACK Kubernetes集群里各种地址的作用，以及地址段该如何规划。

专有网络VPC网段和Kubernetes网段关系

专有网络VPC（下文简称为VPC或专有网络）的网段规划包含VPC自身网段和虚拟交换机网段，Kubernetes网段规划包含Pod地址段和Service地址段。ACK网络支持Terway和Flannel两种网络模式。

Terway网络模式 Flannel网络模式

网络规划

在阿里云环境下使用ACK支持的Kubernetes集群，首先需要根据业务场景、集群规模进行网络规划。您可以按下表规格进行规划（未包含场景，请根据实际需要自行调整）。

集群节点规模	目的	VPC规划	可用区
小于100个节点	一般性业务	单VPC	1个
任意	需要多可用区	单VPC	2个及以上
任意	对可靠性有极致要求、需要多地域	多VPC	2个及以上

本文针对Flannel和Terway网络场景，规划容器网络：

- **Flannel配置示例**

专有网络网段	虚拟交换机网段	Pod网络CIDR网段	Service CIDR网段	最大可分配Pod地址数
192.168.0.0/16	192.168.0.0/24	172.20.0.0/16	172.21.0.0/20	65536

- **Terway配置示例**

- Terway Pod独占模式或IPVlan模式

专有网络网段	虚拟交换机网段	Pod虚拟交换机网段	Service CIDR网段	最大可分配Pod地址数
192.168.0.0/16	192.168.0.0/19	192.168.32.0/19	172.21.0.0/20	8192

- **Terway多可用区配置**

专有网络网段	虚拟交换机网段	Pod虚拟交换机网段	Service CIDR网段	最大可分配Pod地址数
192.168.0.0/16	可用区I 192.168.0.0/19	192.168.32.0/19	172.21.0.0/20	8192
	可用区J 192.168.64.0/19	192.168.96.0/19		8192

VPC网络规划

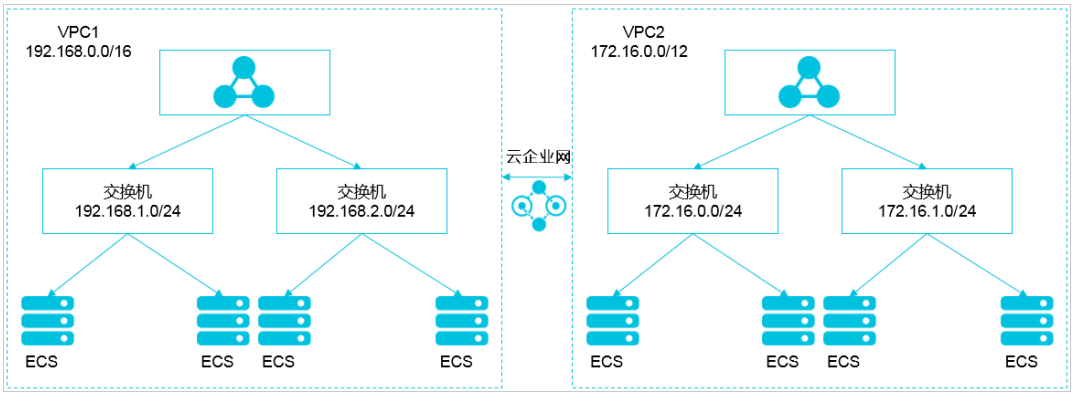
容器网络规划

如何选择地址段？

- **场景1：单VPC+单Kubernetes集群**
  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，选择的Pod及Service地址网段和当前VPC不一样的地址段即可。
- **场景2：单VPC+多Kubernetes集群**
  一个VPC下创建多个Kubernetes集群。
  - VPC地址是在创建VPC时已经确定。创建Kubernetes集群时，每个集群内的VPC地址段、Service地址段和Pod地址段彼此间不能重叠。
  - 所有Kubernetes集群之间的Pod地址段不能重叠，但Service地址段可以重叠。
  - 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。

  > **说明** 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。

- **场景3：VPC互联**
  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。如下表所示，VPC 1使用地址段192.168.0.0/16，VPC 2使用地址段172.16.0.0/12，您可以通过路由表，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。



VPC互联场景

类别	地址段	目的端	转发到
VPC 1	192.168.0.0/16	172.16.0.0/12	VPC 2
VPC 2	172.16.0.0/12	192.168.0.0/16	VPC 1

在这种情况下，VPC 1和VPC 2里创建的Kubernetes集群有以下限制：

- 不能和VPC 1的地址段重叠
- 不能和VPC 2的地址段重叠
- 不能和其他集群的地址段重叠
- 不能和Pod的地址段重叠
- 不能和Service的地址段重叠

此例子中，Kubernetes集群Pod地址段可以选择10.0.0.0/8下的某个子段。

> **说明** 您需特别关注转发到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VPC 1 Kubernetes集群Pod地址的路由。

- **场景4：VPC网络到IDC**
  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。

相关文档

- 网络概述
- 使用Terway网络插件
- 使用网络策略Network Policy
- 网络管理FAQ

7.3. 容器网络CNI

7.3.1. 使用Terway网络插件

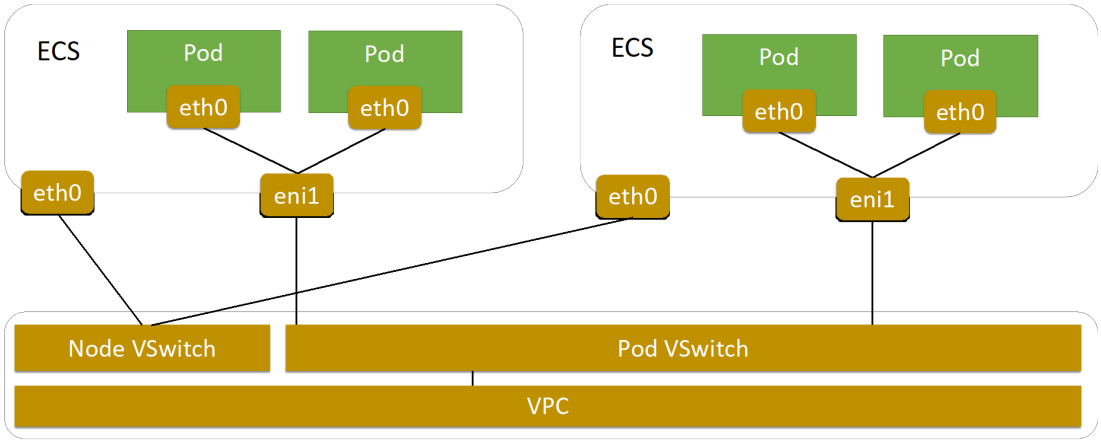
Terway是阿里云开源的基于专有网络VPC的容器网络接口CNI（Container Network Interface）插件，支持基于Kubernetes标准的网络策略来定义容器间的访问策略。您可以通过使用Terway网络插件实现Kubernetes集群内部的网络互通。本文介绍如何使用阿里云容器服务Kubernetes版ACK集群的Terway网络插件。

背景信息

Terway网络插件是ACK自研的网络插件，将原生的弹性网卡分配给Pod实现Pod网络，支持基于Kubernetes标准的网络策略（Network Policy）来定义容器间的访问策略，并兼容Calico的网络策略。

在Terway网络插件中，每个Pod都拥有自己网络栈和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的弹性网卡直接转发。由于不需要使用VxLAN等的隧道技术封装报文，因此Terway模式网络具有较高的通信性能。

Terway网络模式图



Terway与Flannel对比

在创建集群时，ACK提供Terway和Flannel两种网络插件：

- Terway：是阿里云容器服务ACK自研的网络插件。Terway将阿里云的弹性网卡分配给容器，支持基于Kubernetes标准的网络策略来定义容器间的访问策略，支持对单个容器做带宽的限流。Terway拥有更为灵活的IPAM（容器地址分配）策略，避免地址浪费。如果您不需要使用网络策略（Network Policy），可以选择Flannel，其他情况建议选择Terway。
- Flannel：使用的是简单稳定的社区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。Flannel功能偏简单，支持的特性少。例如，不支持基于Kubernetes标准的网络策略（Network Policy）。更多信息，请参见Flannel。

对比项	Terway	Flannel
性能	Pod地址即为VPC中地址，无NAT损耗 支持独占ENI模式，几乎无损。	配合阿里云VPC路由，Pod地址为虚拟地址，存在NAT转换损耗。
安全	支持使用网络策略Network Policy。	不支持使用网络策略Network Policy。
地址管理	无需按节点分配地址段，随用随分配，地址无浪费。	节点维度划分地址段，大规模集群下地址浪费多。
SLB	SLB后端直接对接Pod，支持业务无中断升级。	SLB后端不能直接对接Pod，需要通过NodePort转发。

注意事项

- 如果要使用Terway插件，建议您选择较高规格和较新类型的ECS机型，即5代或者6代的8核以上机型。更多信息，请参见实例规格族。
- 单节点所支持的最大Pod数取决于该节点的弹性网卡（ENI）数。
  - 共享ENI支持的最大Pod数=（ECS支持的ENI数-1）×单个ENI支持的私有IP数
  - 独占ENI支持的最大Pod数=ECS支持的ENI数 - 1

> **说明** 您在创建集群或扩容集群的**实例规格**区域，可以查看ECS支持的ENI数量。关于创建集群的具体步骤，请参见创建ACK Pro版集群。

步骤一：规划和准备集群网络

在创建ACK Kubernetes集群时，您需要指定专有网络VPC、虚拟交换机、 Pod网络CIDR（地址段）和Service CIDR（地址段）。如果使用Terway网络插件，您需要先创建在一个专有网络VPC，然后在VPC下创建两个虚拟交换机。这两个虚拟交换机需要在同一个可用区内。关于使用Terway网络模式时规划配置网段需要注意的更多信息，请参见Kubernetes集群网络规划。

您可以使用以下网段配置，以快速搭建Terway网络。

专有网络网段	虚拟交换机	Pod虚拟交换机	Service CIDR
192.168.0.0/16	192.168.0.0/19	192.168.32.0/19	172.21.0.0/20

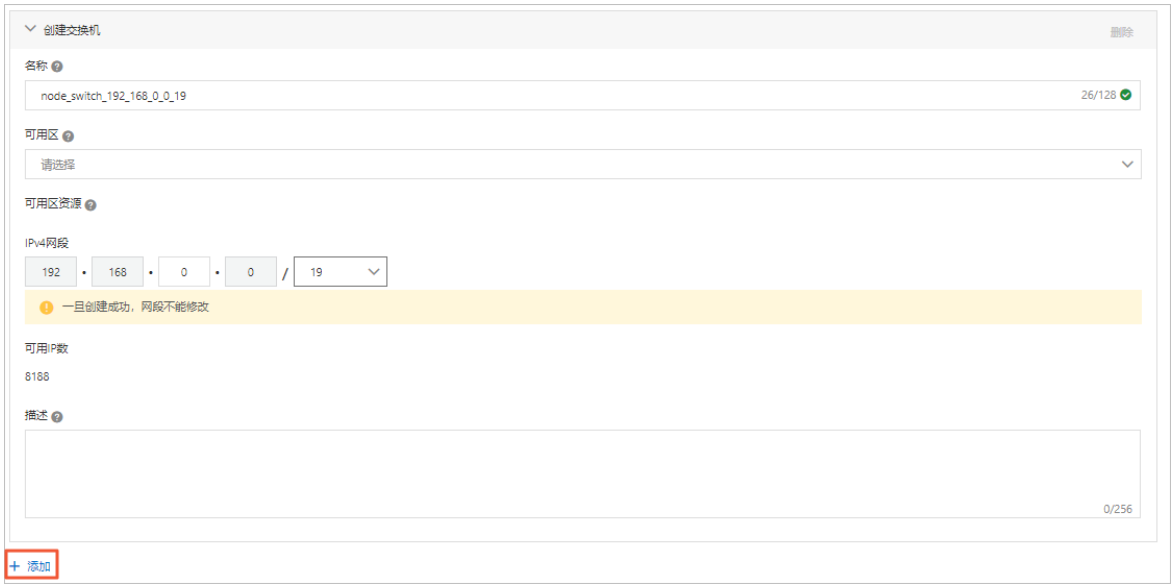
> **说明**
> - 虚拟交换机网段，对应节点所使用的网段。
> - Pod虚拟交换机网段，对应Pod所使用的网段。

本文使用上述推荐网段，说明如何创建一个专有网络和两个虚拟交换机。

1. 登录专有网络管理控制台。
2. 在顶部菜单栏处，选择专有网络的地域，然后单击**创建专有网络**。

   > **说明** 专有网络的地域和要部署的云资源的地域必须相同。

3. 在**创建专有网络**页面，设置**名称**为*vpc_192_168_0_0_16*，在**IPv4网段**文本框中，输入*192.168.0.0/16*。
4. 在**创建交换机**区域，设置虚拟交换机名称为*node_switch_192_168_0_0_19*，设置可用区，及设置**IPv4网络**为**192.168.0.0/19**，单击**+添加**创建第二个虚拟交换机。



> **注意** 请确保两个交换机在同一个可用区。

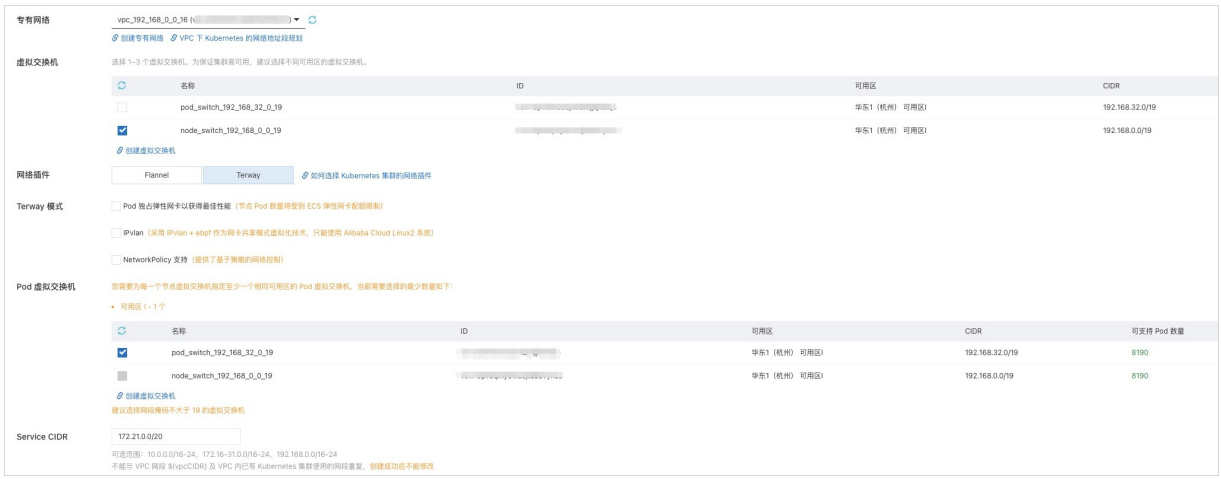
5. 在第二个**创建交换机**区域，配置Pod虚拟交换机，设置名称为**pod_switch_192_168_32_0_19**，IPv4网段

为192.168.32.0/19。

6. 单击**确定**。

步骤二：配置Terway网络

为Terway网络插件配置集群网络的关键参数配置说明如下：



> **说明** 本文以创建标准托管版集群为例，介绍为Terway网络插件配置集群网络涉及的参数。关于创建集群的具体步骤，请参见创建Kubernetes托管版集群。

- **专有网络**：选择步骤一：规划和准备集群网络中创建的专有网络。
- **虚拟交换机**：选择步骤一：规划和准备集群网络中创建的虚拟交换机。
- **网络插件**：选择**Terway**。
  设置**网络插件**为**Terway**时，需要配置**Terway模式**：
  - 设置是否选中**Pod独占弹性网卡以获得最佳性能**
    - 如果选中，Pod将独占一个专有的弹性网卡。
    - 如果不选中，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。

    > **说明** 当前只有白名单用户可使用上述**Pod独占弹性网卡以获得最佳性能**功能。提交工单申请使用。

  - 设置是否选中**IPvlan**
    - 只在弹性网卡共享模式支持选中。
    - 如果选中，采用IPvlan eBPF作为网卡共享模式虚拟化技术，并且只能使用Alibaba Cloud Linux 2系统，性能优于默认模式。
    - 如果不选中，则使用默认模式，采用策略路由作为网卡共享模式虚拟化技术，同时兼容Centos 7和Alibaba Cloud Linux 2的系统。

    关于Terway模式的IPvlan模式的详细信息，请参见Terway IPvlan模式。
  - 设置是否选中**NetworkPolicy支持**
    - 只在弹性网卡共享模式下支持选中，默认不选中。
    - 如果选中，集群支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制。
    - 如果不选中，集群将不会支持使用Kubernetes的NetworkPolicy策略对Pod进行网络控制，这样将不存在网络策略对Kubernetes的API Server产生过大的负载。
- **Pod虚拟交换机**：选择步骤一：规划和准备集群网络中创建的Pod虚拟交换机。
- **Service CIDR**：保留默认值。

Terway IPvlan模式

在创建集群时，如果选择Terway网络插件，可以配置其使用Terway IPvlan模式。Terway IPvlan模式采用IPvlan虚拟化和eBPF内核技术实现高性能的Pod和Service网络。

不同于默认的Terway的网络模式，IPvlan模式主要在Pod网络、Service、网络策略（NetworkPolicy）做了性能的优化：

- Pod的网络直接通过ENI网卡的IPvlan L2的子接口实现，大大简化了网络在宿主机上的转发流程，让Pod的网络性能几乎与宿主机的性能无异，延迟相对传统模式降低30%。
- Service的网络采用eBPF替换原有的kube-proxy模式，不需要经过宿主机上的iptables或者IPVS转发，在大规模集群中性能几乎无降低，扩展性更优。在大量新建连接和端口复用场景请求延迟比IPVS和iptables模式的大幅降低。
- Pod的网络策略（NetworkPolicy）也采用eBPF替换掉原有的iptables的实现，不需要在宿主机上产生大量的iptables规则，让网络策略对网络性能的影响降到最低。

Terway IPvlan模式使用限制

- 需要使用容器服务ACK中的Alibaba Cloud Linux 2的操作系统镜像。
- 尚未支持安全沙箱运行时。
- 网络策略（NetworkPolicy）实现和原有Terway有差异：
  - CIDR选择器优先级低于Pod选择器。如果CIDR包含Pod网段，需要额外增加Pod选择器。
  - 对CIDR选择器中的except关键字支持不佳，不建议使用except关键字。
  - 使用Egress类型的NetworkPolicy会导致访问集群中Host网络类型的Pod和集群中节点的IP失败。
- 集群内部访问对外暴露的LoadBalancer类型Service对应的SLB时可能存在回环问题而导致网络不通。更多信息，请参见为什么无法访问负载均衡。

适用场景

- 中间件与微服务
  避免了大规模部署时带来的性能衰减，微服务全链路延迟降低。
- 游戏、直播类应用
  显著降低网络延迟，多个实例之间的争抢降低。
- 高性能计算
  高性能计算会有大量的网络吞吐，IPvlan模式网络本身带来CPU等的开销减小，留出更多的计算资源给业务本身。

相关文档

- 网络概述
- Kubernetes集群网络规划
- 扩容Terway场景下的虚拟交换机

7.3.2. 扩容Terway场景下的虚拟交换机

Terway网络场景下，当虚拟交换机（VSwitch）IP资源不足时，您需要对VSwitch扩容。本文介绍如何新增虚拟交换机以增加新的IP地址资源供ACK集群使用。

虚拟交换机VSwitch的IP资源不足特征

Terway网络场景下，VSwitch IP资源不足的特征如下：

- 如果您发现Pod创建不成功，且状态显示为ContainerCreating，执行以下命令查看Pod所在节点的Terway的日志。

```
kubectl logs --tail=100 -f terway-eniip-zwjwx -n kube-system -c terway
```

若系统输出类似以下错误信息，说明该节点的Terway所使用的VSwitch没有空余的IP地址，Pod会因为没有IP资源而一直处于ContainerCreating状态。

```
time="2020-03-17T07:03:40Z" level=warning msg="Assign private ip address failed: Aliyun API Error: RequestId: 2095E971-E473-4BA0-853F-0C41CF52651D Status Code: 403 Code: InvalidVSwitchId.IpNotEnough Message: The specified VSwitch \"vsw-AAA\" has not enough IpAddress., retrying"
```

- 登录专有网络管理控制台，在控制台左侧导航栏中，单击**交换机**查看该VSwitch的**可用IP数**为0。

新增虚拟交换机VSwitch

您可通过执行以下操作步骤新增VSwitch。

1. 在专有网络管理控制台创建新的VSwitch，该VSwitch必须与IP资源不足的VSwitch在同一个区域。具体关于如何创建VSwitch，请参见使用交换机。

   > **说明** 因Pod密度越来越大，为了满足Pod对IP地址日益增长的需求，建议您创建给Pod使用的VSwitch的网络位小于等于19，即网段中至少包含8192个IP地址。

2. 执行以下命令，添加VSwitch到Terway的ConfigMap配置中。

   ```
   kubectl edit cm eni-config -n kube-system
   ```

   添加VSwitch具体样例如下：

   ```
   eni_conf: |
   {
   "version": "1",
   "max_pool_size": 25,
   "min_pool_size": 10,
   "vswitches": {"cn-shanghai-f":["vsw-AAA", "vsw-BBB"]},
   "service_cidr": "172.21.0.0/20",
   "security_group": "sg-CCC"
   }
   ```

   样例中添加 `vsw-BBB` 到 `VSwitches` 部分，其中 `vsw-AAA` 是已经存在的且IP资源不足的VSwitch。

3. （可选）升级Terway版本。

   如果Terway版本不是最新的，您需将Terway版本升到最新版本，否则跳转此步骤。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在控制台左侧导航栏中，选择**运维管理 > 组件管理**，然后在**组件管理**页面，单击**网络**页签。
   v. 在**网络**页签中选择需要升级的目标Terway组件，单击**升级**。

4. 执行以下命令删除全部Terway Pod后，系统自动重建全部Terway Pod。
   - 针对ENI多IP场景，执行以下命令删除全部Terway Pod：

     ```
     kubectl delete -n kube-system pod -l app=terway-eniip
     ```

   - 针对ENI单IP场景，执行以下命令删除全部Terway Pod：

     ```
     kubectl delete -n kube-system pod -l app=terway-eni
     ```

   执行以下命令检查全部是否成功重建Terway Pod。

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

5. 创建Pod，验证新创建的Pod是否可以从新添加的VSwitch成功分配获得IP地址。

如果您在上述操作中遇到异常情况，请提交工单。

常见问题

Terway网络下，增加虚拟交换机（VSwitch）后，为何不能正常访问公网？

问题现象：在Terway网络下，因Pod没有IP资源而手动增加虚拟交换机，在增加虚拟交换机后，发现集群不能正常访问公网。

问题原因：Pod IP所属的虚拟交换机不具备公网访问的能力。

解决方法：您可以通过NAT网关的SNAT功能，为Pod IP所属的虚拟交换机配置公网SNAT规则。更多信息，请参见为已有集群开启SNAT公网访问能力。

相关文档

- 使用Terway网络插件
- 使用网络策略Network Policy

7.3.3. 使用VPC附加网段扩充集群的虚拟交换机

在创建容器服务ACK的集群时需要为其选择VPC，后续对集群的扩容只能在集群所属的VPC中进行。如果该VPC的网段之前被规划得太小没有足够IP供扩容时，您可以采用VPC附加网段的方式对VPC进行扩容来满足业务扩容需求。本文介绍如何使用VPC附加网段扩充集群的虚拟交换机（VSwitch）。

前提条件

- 集群所属的VPC现有网段无法分配足够的IP给业务扩容。
- 已创建ACK专有版集群或者于2021年2月及之后创建ACK托管版集群。具体操作，请参见创建Kubernetes托管版集群或创建Kubernetes专有版集群。

> **注意** 2021年2月之前创建的ACK托管版集群不支持VPC附加网段扩容。针对这种情况，您可以升级集群至ACK Pro托管版集群后再使用VPC附加网段扩容集群的虚拟交换机。关于如何升级标准托管版集群至Pro托管版集群，请参见热迁移ACK标准托管集群至Pro托管集群。

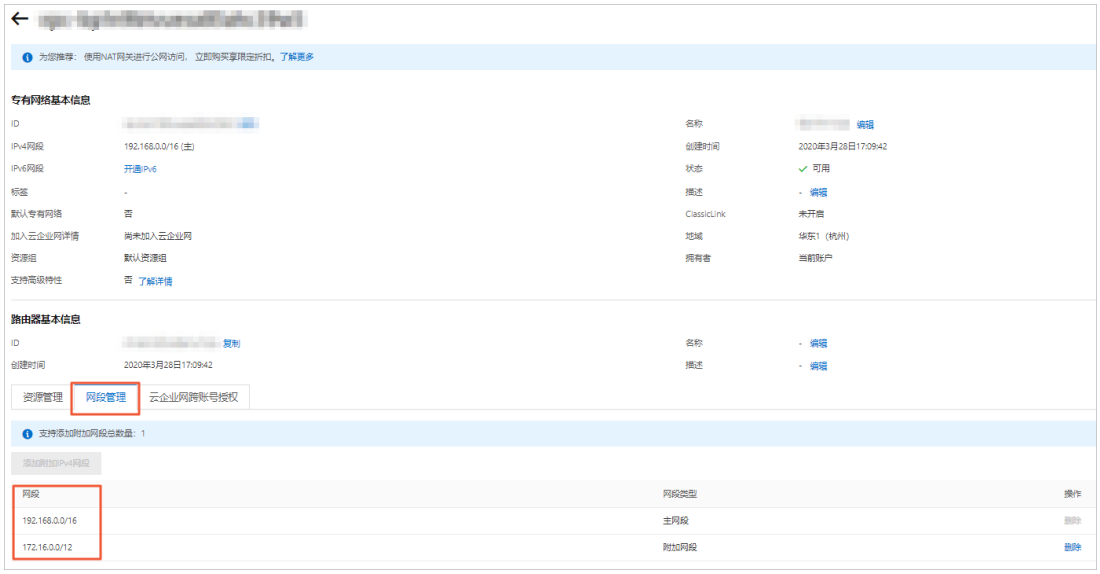
操作步骤

1. 选择可以扩容的网段。

i. 统计现在使用的网段。

包括但不限于以下网段类型：

- 现有的VPC网段。
  关于如何查看现有的VPC网段的具体步骤，请参见查看专有网络。



- 该VPC中集群的Pod网段和Service网段。
  关于如何查看Pod网段及Service网段，请参见查看基本信息。

> **说明**
> - 如果集群使用的模式网络为Terway，查看Service网段。
> - 如果集群使用的模式网络为Flannel，查看Pod网段和Service网段。

- 与该VPC打通的专线、VPN、CEN等外部网段。

ii. 选择一个与上述网段都不重叠的网段作为VPC附加网段。

例如：在Flannel网络下，集群使用的网段如下：

- VPC网段为192.168.0.0/16。
- Pod网段为172.20.0.0/16。
- Service网段为172.21.0.0/16。
- VPC没有加入到专线、VPN、CEN等外部网络中。

则，您可以选择附加网段为10.0.0.0/8。

2. 在VPC控制台上创建附加网段和对应的虚拟交换机。

i. 登录专有网络管理控制台。

ii. 在顶部菜单栏处，选择专有网络的地域。

iii. 在**专有网络**页面，找到目标专有网络，单击专有网络的ID。

iv. 在**专有网络基本信息**页面，单击**网段管理**页签，然后单击**添加附加IPv4网段**，然后添加上一步中找到的合适的IPv4网段。

v. 按照业务对可用区的需求，您可在这个附加IPv4网段下创建对应可用区的虚拟交换机。

关于如何创建虚拟交换机的具体步骤，请参见创建交换机。

3. 在集群的安全组中放开对上述选择的附加网段的出、入方向的所有协议允许规则。

关于如何添加安全组规则的具体步骤，请参见添加安全组规则。

4. 在集群扩容界面选择新的虚拟交换机用于扩容节点。

   在上述创建的附加网段中的虚拟交换机下即可扩容新的网段的节点。具体步骤，请参见扩容ACK集群的节点。

   > **注意** 在Terway网络模式下，您可以修改Terway的交换机配置用于扩容Pod。关于如何把创建的附加网段中的虚拟交换机用于Pod网络，请参见扩容Terway场景下的虚拟交换机。

   扩容完成后：

   - ACK专有版的管控节点与附加网段的节点可以直接通信，无需做额外配置。
   - ACK标准版和Pro托管版集群需要对管控面做重新配置，否则新的节点上的Pod无法通过API Server执行**kubectl exec**和**kubectl logs**命令。您需要提交工单联系技术人员重新配置管控面。

5. 验证添加的VPC附加网段。

   方式如下：

   - 新分配的节点或者Pod使用了新添加的VPC附加网段中的IP地址。
   - 新分配的节点状态为**Ready**状态。
   - 在新的节点和Pod中分别测试与集群中原有节点和Pod的网络的联通性。

相关文档

- Kubernetes集群网络规划
- 添加附加IPv4网段
- 扩容Terway场景下的虚拟交换机
- 热迁移ACK标准托管集群至Pro托管集群

7.3.4. 使用网络策略Network Policy

Kubernetes网络策略（Network Policy）提供基于策略的网络控制。当您使用Terway容器网络时，如果您希望在IP地址或者端口层面控制网络流量，您可以为集群中特定应用使用网络策略。本文介绍如何使用ACK集群的网络策略及常用的使用场景。

前提条件

- 创建Kubernetes托管版集群。

  > **注意** 您在创建集群时，需设置**网络插件**为Terway。

- 通过kubectl工具连接集群。

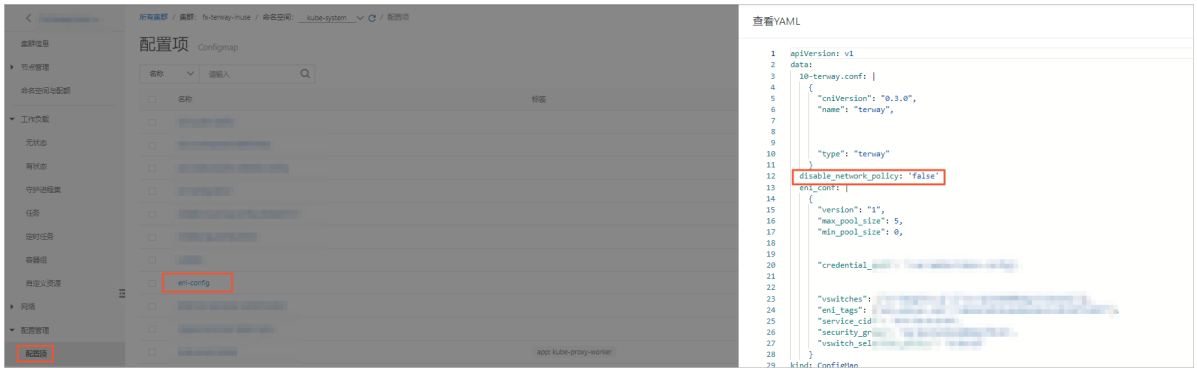
开启网络策略

> **说明**
>
> - 如果您想通过控制台的方式使用网络策略功能，请提交工单申请成为白名单用户。
> - 如果通过命令行方式使用网络策略功能，无需申请成为白名单用户。

您可以在创建Terway集群时开启网络策略功能，也可以在创建Terway集群后再开启网络策略功能：

- 如果您在创建Terway集群时，选中了**NetworkPolicy支持**配置，可自动开启网络策略功能。具体操作，请参见创建ACK Pro版集群。

- 如果您在创建Terway集群时，未开启网络策略功能，可以通过**配置管理**入口，在kube-system命名空间下的**eni-config**配置项中，将disable_network_policy设置成*false*。关于配置项的设置，请参见修改配置项。



修改**eni-config**配置项后，需要重启Terway以开启网络策略功能。关于如何重启Terway的具体操作，请参见重建全部Terway Pod。

使用场景

本文介绍在以下场景中如何设置网络策略：

- 通过网络策略限制服务只能被带有特定标签的应用访问
- 通过网络策略限制可访问公网服务的来源IP网段
- 通过网络策略限制一个Pod只能访问指定地址
- 通过网络策略控制命名空间下Pod公网访问权限

以上场景均使用在创建可以被其他Pod正常访问的nginx测试应用中创建的应用。

创建可以被其他Pod正常访问的nginx测试应用

通过控制台操作

1. 创建名称为nginx的应用，并通过Service将其公开。具体步骤，请参见使用镜像创建无状态Deployment应用。

   部分示例配置如下。

配置项	参数	示例值
应用基本信息	应用名称	nginx
	副本数量	1
容器配置	镜像名称	nginx
服务创建	名称	nginx
	类型	○ 负载均衡 ○ 公网访问 ○ 新建SLB
	端口映射	○ 名称：nginx ○ 服务端口：80 ○ 容器端口：80 ○ 协议：TCP

2. 创建用于测试访问名称为busybox的客户端应用，用于测试访问上一步中创建的名为nginx的Service。具体步

骤，请参见使用镜像创建无状态Deployment应用。

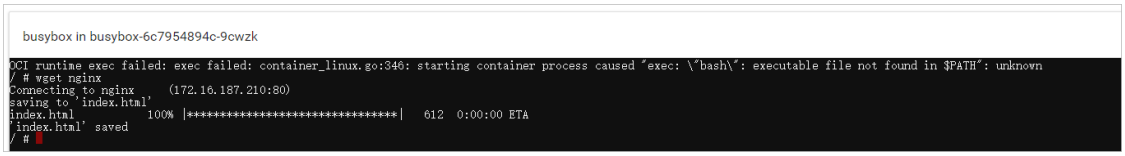
部分示例配置如下。

配置项	参数	示例值
应用基本信息	应用名称	busybox
	副本数量	1
容器启动项	镜像名称	busybox
	容器启动项	选中stdin和tty

3. 验证busybox客户端应用是否可以访问nginx Service。
   i. 在**无状态**页面，找到并单击busybox名称。
   ii. 在**容器组**页签下的busybox-{hash值}容器组右侧，单击**终端**。



   iii. 在busybox的命令行终端中执行 `wget nginx` 测试访问nginx。



   当出现以上信息时，说明您可以在busybox中访问nginx Service。

通过命令行操作

1. 执行以下命令，创建一个nginx的应用，并通过名称为nginx的Service与其通信。

   创建nginx应用：

```
kubectl run nginx --image=nginx
```

   预期输出：

```
pod/nginx created
```

   查看Pod是否启动：

```
kubectl get pod
```

   预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx         1/1    Running  0         45s
```

创建名为nginx的Service：

```
kubectl expose pod nginx --port=80
```

预期输出：

```
service/nginx exposed
```

```
kubectl get service
```

预期输出：

```
NAME     TYPE     CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
kubernetes  ClusterIP  172.XX.XX.1   <none>     443/TCP  30m
nginx     ClusterIP  172.XX.XX.48  <none>     80/TCP   12s
```

2. 执行以下命令，创建名称为busybox的Pod，访问名为nginx的Service。

```
kubectl run busybox --rm -ti --image=busybox /bin/sh
```

预期输出：

```
If you don't see a command prompt, try pressing enter.
/ #
```

```
wget nginx
```

预期输出：

```
Connecting to nginx (172.XX.XX.48:80)
saving to 'index.html'
index.html       100% |********************************************************************************************
************************************************************|  612  0:00:00 ETA
'index.html' saved
```

通过网络策略限制服务只能被带有特定标签的应用访问

通过控制台操作

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 网络策略**。
5. 在**网络策略**页面右上方，单击**创建**，然后在**创建面板**中配置网络策略。

参数	描述
名称	设置网络策略的名称，本文示例填写：access-nginx。
命名空间	设置放置网络策略的命名空间。

参数	描述
Pod选择器	设置使用网络策略的Pod。 本文示例设置如下： ○ 设置类型为：**无状态** ○ 设置工作负载为：**nginx** ○ 设置标签为：**app=nginx** **说明** 如果Pod选择器设置为空，则对命名空间下所有Pod生效。
来源	每个网络策略可包含一个来源（ingress规则）的白名单列表。 每个规则都允许同时匹配来源规则和特定端口部分的流量。 ○ **规则**： ■ podSelector：此选择器将在与网络策略同命名空间中选择特定的 Pod，并将其允许作为入站流量来源。 ■ namespaceSelector：此选择器将选择特定的命名空间，并将所有Pod 用作其入站流量来源。 ■ ipBlock：此选择器将选择特定的IP CIDR范围以用作入站流量来源。 ○ **端口**：支持TCP和UDP协议。 **说明** 如果不添加任何规则，表示不允许访问任何Pod与选择的Pod进行访问。 本示例不添加任何来源规则。
去向	每个网络策略可包含一个egress规则的白名单列表。每个规则都允许匹配去向规则和特定端口部分的流量。 ○ **规则**： ■ podSelector：此选择器将在与网络策略同命名空间中选择特定的 Pod，并将其允许作为出站流量目的地。 ■ namespaceSelector：此选择器将选择特定的命名空间，并将所有Pod用作其出站流量目的地。 ■ ipBlock：此选择器将选择特定的IP CIDR范围以出站流量目的地。 ○ **端口**：支持TCP和UDP协议。 本示例不添加任何去向规则。

6. 在busybox命令行终端执行 `wget nginx` 命令，测试与nginx Service的通信。具体操作，请参见步骤3。
因在网络策略没有允许busybox访问，所以访问会超时。

```
/ # wget nginx
Connecting to nginx (172.16.49.27:80)
wget: can't connect to remote host (172.16.49.27): Connection timed out
/ #
```

7. 修改网络策略为允许busybox的应用访问。
   i. 在网络策略列表中，找到access-nginx网络策略并在其右侧，单击**编辑**。

ii. 添加来源规则。

在**来源**右侧，单击**+添加**，并继续以下操作：

- 在**规则**右侧，单击**+添加**。添加podSelector的访问，规则设置如下：

参数	示例值
选择器	podSelector
类型	无状态
工作负载	busybox

- 在**端口**右侧，单击**+添加**。端口设置如下：

参数	示例值
协议	TCP
端口	80

iii. 单击**下一步**，然后单击**确定**。

iv. 执行 `wget -O /dev/null nginx` 命令，测试修改网络策略后busybox与nginx的通信。
网络策略增加busybox应用的规则后，busybox可以正常访问nginx。

```
/ # wget -O /dev/null nginx
Connecting to nginx (172.16.49.27:80)
saving to '/dev/null'
null                 100% |********************************|   612  0:00:00 ETA
'/dev/null' saved
```

通过命令行操作

1. 使用以下YAML模板并执行命令 `vim policy.yaml` ，创建名为*policy.yaml*文件。

```
vim policy.yaml
```

以下为YAML文件内容。

```
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，根据上述的*policy.yaml*文件创建网络策略。

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

预期输出：

```
networkpolicy.networking.k8s.io/access-nginx created
```

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

```
kubectl run busybox --rm -ti --image=busybox /bin/sh
```

```
wget nginx
```

预期输出：

```
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 Service。

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

```
wget nginx
```

预期输出：

```
Connecting to nginx (172.21.x.114:80)
saving to 'index.html'
index.html       100% |********************************************************************************************
**************************************************************|  612  0:00:00 ETA
'index.html' saved
```

连接nginx的进度为100%时，说明请求成功，可以正常访问nginx服务。

通过网络策略限制可访问公网服务的来源IP网段

通过控制台操作

1. 为nginx Service设置网络策略，相关配置，请参见通过网络策略限制服务只能被带有特定标签的应用访问。
2. 在服务列表中的**外部端点**列下，查看创建可以被其他Pod正常访问的nginx测试应用中nginx Service的公网访问地址（47.xxx.xx.x），并在浏览器中访问该地址。



由于已为nginx添加网络策略，默认访问nginx失败。

3. 通过网络策略增加允许访问的网段来允许客户端访问。
   i. 通过浏览器访问myip.ipip.net获取本机的IP地址。

ii. 在网络策略列表中，找到access-nginx网络策略并在其右侧，单击**编辑**，然后在**编辑**面板中修改规则。

在**来源**右侧，单击**+添加**，并继续以下操作：

- 在**规则**右侧，单击**+添加**。规则中增加本机的IP地址的访问：

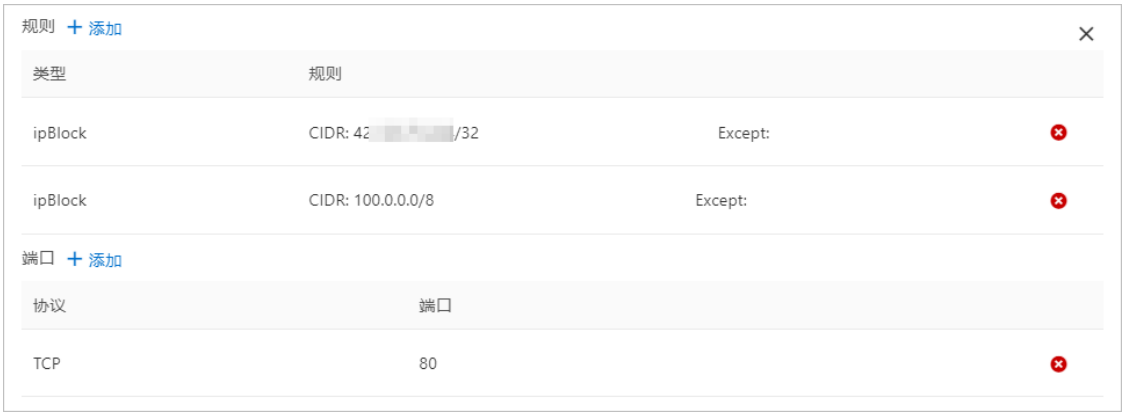
参数	示例值
选择器	ipBlock
cidr	<本机的IP地址>/32 例如：42.xx.xx.xx/32

- 在**规则**右侧，单击**+添加**。规则中增加阿里云SLB的健康检查网段的访问：

参数	示例值
选择器	ipBlock
cidr	100.0.0.0/8

- 在**端口**右侧，单击**+添加**。端口设置如下：

参数	示例值
协议	TCP
端口	80



iii. 单击**下一步**，然后单击**确定**。

iv. 访问nginx外部端点（47.xxx.xx.x）。

增加网络策略后客户端可以通过公网SLB访问nginx。

通过命令行操作

1. 执行以下命令，为上述名为nginx的应用创建阿里云SLB服务，指定`type=LoadBalancer`来向外网用户暴露nginx服务。

   *nginx-service.yaml*文件模板如下。

```
vim nginx-service.yaml
# 以下为YAML文件内容。
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
```

预期输出：

```
service/nginx-slb created
```

```
kubectl get service nginx-slb
```

预期输出：

```
NAME     TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)     AGE
nginx-slb  LoadBalancer  172.19.xx.xxx  47.110.xxx.xxx  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.xxx.xxx*，访问失败。

```
wget 47.110.xxx.xxx
```

预期输出：

```
--2018-11-21 11:46:05--  http://47.110.xx.xxx/
Connecting to 47.110.XX.XX:80... failed: Connection refused.
```

> **说明**
> 访问失败的原因是：
> - 配置的nginx Service只能被带有特定标签即 `access=true` 的应用访问。
> - 访问SLB的IP地址，是从外部访问Kubernetes，与通过网络策略限制服务只能被带有特定标签的应用访问不同。
>
> 解决方法：修改网络策略，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
```

预期输出：

```
当前IP：10.0.x.x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
# 以下为YAML文件内容。
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
```

预期输出：

```
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。

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

```
wget 47.110.XX.XX
```

预期输出：

```
Connecting to 47.110.XX.XX (47.110.XX.XX:80)
index.html       100% |*******************************************************|  612  0:00:00 ETA
```

连接进度显示100%时，说明成功访问Nginx服务。

通过网络策略限制一个Pod只能访问指定地址

通过控制台操作

本文以*www.aliyun.com*和*registry.aliyuncs.com*为例，说明通过设置网络策略一个Pod只能访问*registry.aliyuncs.com*。

1. 通过Public DNS网站查询*registry.aliyuncs.com*地址绑定的IP地址（120.55.105.209）。
2. 在网络策略中创建规则限制busybox应用只能访问*registry.aliyuncs.com*。
   i. 在**网络策略**页面右上角，单击**创建**，然后在**创建**面板中配置网络策略规则。
   关于具体的参数描述，请参见通过网络策略限制服务只能被带有特定标签的应用访问。示例配置如下。
   - 设置**名称**为busybox-policy。

- 设置Pod选择器。

参数	示例值
类型	无状态
工作负载	busybox
标签	app=busybox

- 在**去向**右侧，单击**+添加**，并继续以下操作。
  - 在**规则**右侧，单击**+添加**。添加ipBlock为前面获取的*registry.aliyuncs.com*的解析的IP地址（120.55.105.209）/32的规则。

参数	示例值
选择器	ipBlock
cidr	120.55.105.209/32

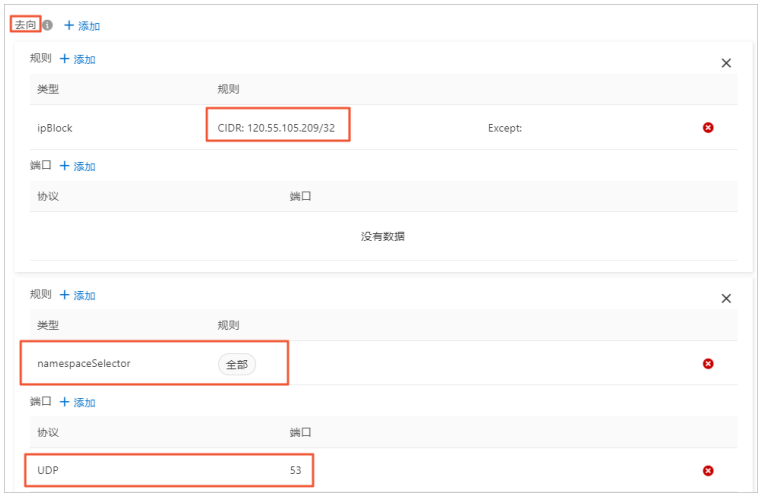
- 在**去向**右侧，单击**+添加**。添加命名空间选择器为全部命名空间的UDP 53端口的访问的规则，用来保障应用可以解析DNS。
  - 在**规则**右侧，单击**+添加**。添加命名空间选择器为全部的规则。

参数	示例值
选择器	namespace
命名空间	全部

  - 在**端口**右侧，单击**+添加**。添加UDP 53的规则用来保障应用可以解析DNS。端口设置如下：

参数	示例值
协议	UDP
端口	53

去向规则配置如下图。



ii. 单击**下一步**，然后单击**确定**。

iii. 在busybox终端执行以下命令分别访问*www.aliyun.com*和*registry.aliyuncs.com*。

```
nc -vz -w 1 www.aliyunc.com 443
```

```
nc -vz -w 1 registry.aliyuncs.com 443
```

增加网络策略后，busybox应用只能访问registry.aliyuncs.com而无法访问其他地址。

```
/ # nc -vz -w 1 www.aliyunc.com 443
nc: www.aliyunc.com (183.111.66.98:443): Connection timed out
/ # nc -vz -w 1 registry.aliyuncs.com 443
registry.aliyuncs.com (120.55.105.209:443) open
/ #
```

通过命令行操作

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
```

预期输出：

```
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.XX.XX.21
140.XX.XX.4
140.XX.XX.13
140.XX.XX.3
```

2. 执行以下命令，创建*busybox-policy*文件。

```
vim busybox-policy.yaml
# 以下为YAML文件内容。
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: busybox-policy
spec:
 podSelector:
  matchLabels:
   run: busybox
 egress:
 - to:
  - ipBlock:
     cidr: 106.XX.XX.21/32
  - ipBlock:
     cidr: 140.XX.XX.4/32
  - ipBlock:
     cidr: 140.XX.XX.13/32
  - ipBlock:
     cidr: 140.XX.XX.3/32
 - to:
  - ipBlock:
     cidr: 0.0.0.0/0
  - namespaceSelector: {}
  ports:
  - protocol: UDP
    port: 53
```

说明 在*busybox-policy*文件中，配置了去向规则（Egress），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*文件创建网络策略。

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

预期输出：

```
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.taobao.com*。

```
kubectl run busybox --rm -ti --image=busybox /bin/sh
```

```
wget www.taobao.com
```

预期输出：

```
Connecting to www.taobao.com (64.13.XX.XX:80)
wget: can't connect to remote host (64.13.XX.XX): Connection timed out
```

出现can't connect to remote host时，说明失败访问服务。

5. 执行以下命令，访问*www.aliyun.com*。

```
wget www.aliyun.com
```

预期输出：

```
Connecting to www.aliyun.com (140.205.XX.XX:80)
Connecting to www.aliyun.com (140.205.XX.XX:443)
wget: note: TLS certificate validation not implemented
index.html       100% |*******************************************************| 462k  0:00:00 ETA
```

进行显示100%，说明成功访问服务。

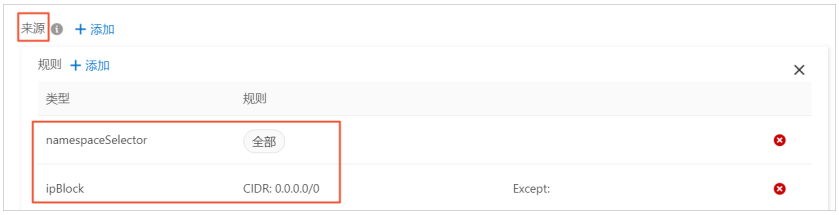
通过网络策略控制命名空间下Pod公网访问权限

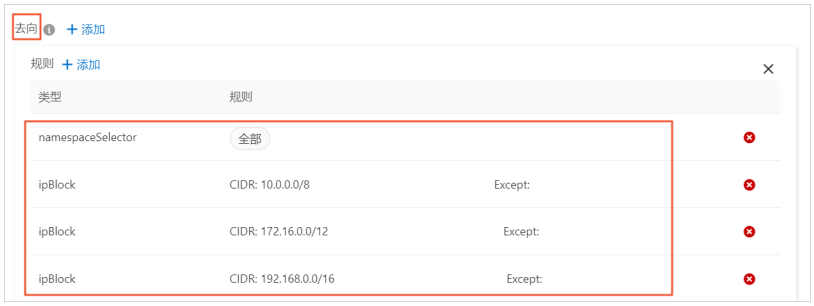
> **说明** 此操作有可能影响线上正在访问公网的服务，建议在一个空的命名空间中进行以下操作。

通过控制台操作

1. 在**网络策略**页面右上角，单击**创建**，然后在**创建**面板中配置网络策略规则。

   关于具体的参数描述及操作，请参见通过网络策略限制服务只能被带有特定标签的应用访问。示例配置如下。

参数	示例值
名称	deny-public-net
Pod选择器	设置类型为**全部**。
来源	添加以下两条规则： ○ 设置**namespaceSelector**的**全部**允许的规则。 ○ 设置**ipBlock**的**全部**允许的规则。 

参数	示例值
去向	添加只允许访问内网的规则： ○ 设置**namespaceSelector**的**全部**允许的规则，允许Pod访问内网的所有Pod。 ○ 设置**ipBlock**的三条内网网段的规则，包括以下三个内网网段： ■ 10.0.0.0/8 ■ 172.16.0.0/12 ■ 192.168.0.0/16 **说明** 不可同时在一条ipBlock规则中同时添加多个网段。 

2. 单击**下一步**，然后单击**确定**。
3. 通过上文创建可以被其他Pod正常访问的nginx测试应用中创建的busybox应用，执行以下命令测试Pod的公网访问权限。

```
nc -vz -w 1 www.aliyunc.com 443
```

```
nc -vz -w 1 192.168.0.120:10255
```

Pod只能访问内网的地址而无法访问公网的地址。

```
/ # nc -vz -w 1 www.aliyunc.com 443
nc: www.aliyunc.com (183.111.66.98:443): Connection timed out
/ # nc -vz -w 1 192.168.0.120:10255
192.168.0.120:10255 (192.168.0.120:10255) open
```

通过命令行操作

1. 执行以下命令，创建一个测试的命名空间来验证限制的能力。

   创建一个test-np的命名空间。

```
kubectl create ns test-np
```

   预期输出：

```
namespace/test-np created
```

2. 执行以下命令，为这个命名空间设置默认的只允许主动访问私网的网络策略规则。

```
vim default-deny.yaml
# 以下为YAML文件内容。
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 namespace: test-np
 name: deny-public-net
spec:
 podSelector: {}
 ingress:
 - from:
  - ipBlock:
    cidr: 0.0.0.0/0
 egress:
 - to:
  - ipBlock:
    cidr: 192.168.0.0/16
  - ipBlock:
    cidr: 172.16.0.0/12
  - ipBlock:
    cidr: 10.0.0.0/8
```

```
kubectl apply -f default-deny.yaml
```

预期输出：

```
networkpolicy.networking.k8s.io/deny-public-net created
```

```
kubectl get networkpolicy -n test-np
```

预期输出：

```
NAME              POD-SELECTOR     AGE
deny-public-net       <none>       1m
```

3. 执行以下命令，为特殊标签的Pod设置允许访问公网。

示例标签设置为 `public-network=true` 。

```
vim allow-specify-label.yaml
# 以下为YAML文件内容。
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: allow-public-network-for-labels
 namespace: test-np
spec:
 podSelector:
  matchLabels:
   public-network: "true"
 ingress:
 - from:
  - ipBlock:
     cidr: 0.0.0.0/0
 egress:
 - to:
  - ipBlock:
     cidr: 0.0.0.0/0
  - namespaceSelector:
     matchLabels:
      ns: kube-system
```

```
kubectl apply -f allow-specify-label.yaml
```

预期输出：

```
networkpolicy.networking.k8s.io/allow-public-network-for-labels created
```

```
kubectl get networkpolicy -n test-np
```

预期输出：

```
NAME                    POD-SELECTOR      AGE
allow-public-network-for-labels  public-network=true   1m
deny-public-net            <none>          3m
```

4. 执行以下命令验证无特殊标签的Pod不能访问公网。

```
kubectl run -it --namespace test-np --rm --image busybox  busybox-intranet
```

```
ping aliyun.com
```

预期输出：

```
PING aliyun.com (106.11.2xx.xxx): 56 data bytes
^C
--- aliyun.com ping statistics ---
9 packets transmitted, 0 packets received, 100% packet loss
```

显示0 packets received时，说明访问失败。

> **说明** 访问失败的原因是：由于通过deny-public-net的网络策略规则限制了test-np的这个命名空间下的Pod的默认的公网访问，所以它下面启动的默认标签的Pod无法访问公网。

5. 执行以下命令验证带有public-network=true的Pod可以访问服务。

```
kubectl run -it --namespace test-np --labels public-network=true --rm --image busybox  busybox-internet
```

```
ping aliyun.com
```

预期输出：

```
PING aliyun.com (106.11.1xx.xx): 56 data bytes
64 bytes from 106.11.1xx.xx: seq=0 ttl=47 time=4.235 ms
64 bytes from 106.11.1xx.xx: seq=1 ttl=47 time=4.200 ms
64 bytes from 106.11.1xx.xx: seq=2 ttl=47 time=4.182 ms
^C
--- aliyun.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 4.182/4.205/4.235 ms
```

显示0% packet loss时，说明成功访问服务。

> **说明** 可以正常访问的原因是：由于在allow-public-network-for-labels的网络策略的规则中允许了带有public-network=true的Pod的公网访问，所以在带有这个Label的busybox-internet的Pod可以访问公网。

7.3.5. 使用Host网络

Pod使用Host网络即让Pod使用Node节点的网络空间和网络资源。将Pod的网络配置成Host网络后，Pod可以直接访问到Node上Loopback设备，可以在Node节点上监听地址，也可以用于监控Node上别的Pod的网络流量。本文介绍如何使用Host网络。

前提条件

- 您已经成功创建一个Kubernetes集群，参见创建ACK Pro版集群。
- 您可以通过Kubectl连接到Kubernetes集群，参见通过kubectl工具连接集群。

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. 执行以下命令，创建一个Pod。

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

3. 执行以下命令，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>
```

7.3.6. 为Pod挂载独立公网EIP

Pod的IP地址一般是私网的IP地址，但是在ACK的Terway网络模式中，某些场景下的Pod需要独立的公网IP地址。本文介绍如何为Terway网络中的Pod挂载独立的公网EIP。

背景信息

一般情况下，Terway网络模式中的Pod访问公网的流量是通过主机SNAT（Source Network Address Translation）或者外部SNAT通过EIP而实现，对于Pod的公网入口流量一般是通过LoadBalancer类型的Service流入。在有的场景中Pod需要独立的公网地址，例如：

- Pod的对外映射端口是随机的，一般常见于UDP（User Datagram Protocol）的游戏服务器或电话会议等。例如RTSP协议，对不同的客户端使用不同的端口。
- Pod访问公网流量存在争抢，Pod需要独立的公网出口。

关于ACK集群Terway网络模式的更多信息，请参见使用Terway网络插件。

步骤一：升级Terway组件至最新版本

您需升级Terway组件到支持EIP功能的版本。关于升级Terway组件的具体步骤，请参见管理组件。

> **说明** 目前Terway组件 `v1.0.10.280-gdc2cb6c-aliyun` 及以上版本支持EIP功能。建议您升级Terway组件至最新版本。

步骤二：为Terway部署EIP相关的配置和权限

Terway为Pod配置EIP需要EIP相关的权限去申请和配置EIP，所以需要部署额外的配置和权限。

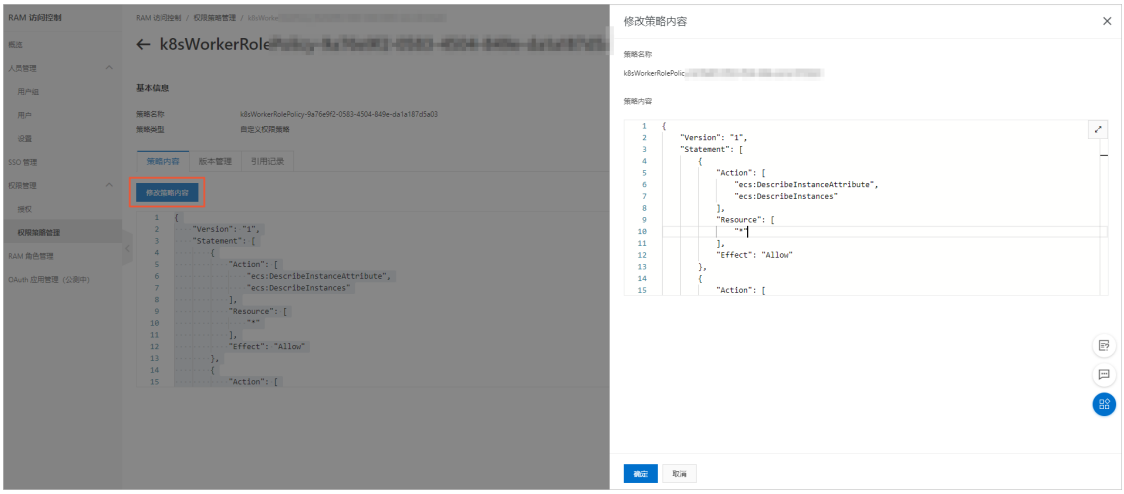
1. 为Terway模式的RAM角色配置EIP相关权限。

   ### 对于专有版集群或者2020年06月份之前创建的托管版集群，Terway的权限是通过Worker节点上的RAM角色进行配置。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 单击**集群资源**页签，然后单击**Worker RAM角色**链接。
   v. 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。



vi. 单击**修改策略内容**，从右侧滑出侧边栏**修改策略内容**面板，在**策略内容**的 Action 字段中补充以下策略，然后单击**确定**。




```
"vpc:DescribeVSwitches",
"vpc:AllocateEipAddress",
"vpc:DescribeEipAddresses",
"vpc:AssociateEipAddress",
"vpc:UnassociateEipAddress",
"vpc:ReleaseEipAddress"
```

> **说明** 在策略内容的任意一个 Action 字段的最后一行补充英文逗号（，）。

您需要在RAM控制台中为这个角色添加以下EIP相关权限。

```
"vpc:DescribeVSwitches",
"vpc:AllocateEipAddress",
"vpc:DescribeEipAddresses",
"vpc:AssociateEipAddress",
"vpc:UnassociateEipAddress",
"vpc:ReleaseEipAddress"
```

对于Pro托管版集群，或者是2020年6月份及之后创建的标准托管版集群，Terway的权限是通过RAM角色AliyunCSManagedNetworkRole进行配置。

i. 在RAM角色AliyunCSManagedNetworkRole页面的RAM角色列表中，单击目标授权策略名称右侧的**添加权限**。

ii. 在**添加权限**面板的**选择权限**区域，单击**+新建权限策略**创建自定义策略。

关于如何创建自定义策略，请参见创建自定义策略。

iii. 在您创建的自定义权限策略中添加以下权限。

关于如何修改自定义策略内容，请参见修改自定义策略内容。

```
"vpc:DescribeVSwitches",
"vpc:AllocateEipAddress",
"vpc:DescribeEipAddresses",
"vpc:AssociateEipAddress",
"vpc:UnassociateEipAddress",
"vpc:ReleaseEipAddress"
```

2. 部署Terway配置以支持EIP功能。

i. 执行以下命令修改Terway的配置ConfigMap。

```
kubectl edit cm eni-config -n kube-system
```

ii. 在eni_conf中增加以下内容。

```
"enable_eip_pool": "true"
```

> 注意 如果您希望在指定EIP的时候能够强制解绑之前的实例，还需要在eni_conf中增加参数"allow_eip_rob": "true"。

iii. 执行以下命令重建Terway实例。

```
kubectl delete pod -n kube-system -l app=terway-eniip
```

为Terway模式的RAM角色配置EIP相关权限

步骤三：通过注解为指定Pod创建和关联EIP

通过指定Pod中的annotations可以创建或者关联EIP到此Pod中。根据不同场景，配置注解方式如下：

- **自动分配EIP场景**
  - 添加以下注解，为Pod创建并绑定一个EIP：

    ```
    k8s.aliyun.com/pod-with-eip: "true"
    ```

    示例YAML如下。

    ```
    apiVersion: apps/v1
    kind: Deployment
    metadata:
     name: nginx-deployment-basic
     labels:
       app: nginx
    spec:
     replicas: 2
     selector:
       matchLabels:
         app: nginx
     template:
       metadata:
         annotations:
           k8s.aliyun.com/pod-with-eip: "true" # 为Nginx容器自动分配公网EIP地址。
         labels:
           app: nginx
       spec:
         containers:
         - name: nginx
           image: nginx
    ```

  - 添加以下注解，指定EIP的带宽，默认带宽5 Mbps（与EIP的默认值保持一致）：

    ```
    k8s.aliyun.com/eip-bandwidth: "5"
    ```

- **指定EIP场景**

  添加以下注解，为Pod指定EIP实例ID：

  ```
  k8s.aliyun.com/pod-eip-instanceid: "<youreipInstanceId>"
  ```

> 说明
> - 由于单个EIP不支持关联多个Pod，所以不适用于Deployment和StatefulSet等场景。
> - 默认情况下如果EIP已经绑定了实例，则会创建EIP失败。如果希望解绑之前的实例再绑定新的实例，需要在上述修改ConfigMap中配置"allow_eip_rob": "true"。
> - 指定EIP实例ID的场景只能用于单个副本实例的情况，例如，Deployment不能超过1个Replicas。

验证配置

当Pod变成Running状态之后，可以观察部署完后的Pod中annotations的k8s.aliyun.com/allocated-eipAddress值来查看到它分配到的关联EIP地址，通过这个EIP即可访问到Pod。

相关文档

- Kubernetes集群网络规划
- 使用Terway网络插件

7.3.7. 使用VPC的多路由表功能

ACK通过CCM（Cloud Controller Manager）组件在VPC路由表中添加合适的路由来连通集群中Pod之间的网络连接。您可以通过更新*cloud-config*配置文件让ACK专有版集群使用VPC多路由表功能。本文介绍如何让ACK专有版集群使用VPC支持的多路由表。

前提条件

- 创建一个ACK专有版集群。具体步骤，请参见创建Kubernetes专有版集群。
- ACK专有版集群的网络为Flannel模式。
- CCM版本需大于v1.9.3.86-g4454991-aliyun。关于如何查看CCM版本的步骤，请参见管理组件。

背景信息

VPC对多个路由表的支持能力，让您可以自行选择ECS关联路由表。由于CCM早期版本中仅提供了VPC单路由表的支持，因此您需要在容器服务控制台升级CCM至最新版本。关于如何升级CCM的版本，请参见管理组件。

> 说明 目前仅ACK专有版集群可通过自行配置使用VPC多路由表功能。如果需ACK托管版集群支持使用VPC的多路由表功能，请提交工单。

通过kubectl命令行方式配置支持VPC多路由表

使用kubectl命令方式配置支持VPC多路由表前，确保可以使用kubectl命令链接ACK专有版集群。具体操作，请参见通过kubectl工具连接集群。

1. 更新配置项*cloud-config*文件。

   使用命令`kubectl edit cm -n kube-system cloud-config`修改*cloud-config*配置项文件，将配置项文件中的`${ROUTE_TABLES_IDS}`替换为您的VPC路由表ID（必须包含系统路由表），多个路由表使用英文逗号（,）分隔（例如，vtb-t4n788888****,vtb-t4n7k6u3m0n840799****）。
   *cloud-config*配置项文件部分内容如下。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: cloud-config
 namespace: kube-system
data:
 cloud-config.conf: |-
  {
    "Global": {
      "routeTableIDs": "${ROUTE_TABLES_IDS}"
    }
  }
```

2. 执行以下命令重启CCM Pod。

```
kubectl -n kube-system delete po -lapp=cloud-controller-manager
```

重启完毕后，即可在路由表中看到对应的集群节点条目。

通过控制台方式配置支持VPC多路由表

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在kube-system命名空间下，单击配置项*cloud-config*右侧的**YAML编辑**。
6. 在**查看YAML**面板的routeTableIDs参数的值中，填写您的VPC路由表ID（必须包含系统路由表），多个路由表使用半角逗号（,）分隔（例如，vtb-t4n788888****,vtb-t4n7k6u3m0n840799****），然后单击**确定**。

```
1  apiVersion: v1
2  data:
3    cloud-config.conf: |-
4      {
5        "Global": {
6
7          "routeTableIDs": "",
8          "clusterID": "cb2eda21
9        }
10     }
```

7. 在集群管理页左侧导航栏中，选择**工作负载 > 守护进程集**。
8. 在kube-system命名空间下，找到应用cloud-controller-manager并在其右侧，选择**更多 > 重新部署**。
重新部署后，即可在路由表中看到对应的集群节点条目。

7.3.8. 为已有集群开启SNAT公网访问能力

SNAT可以为VPC内无公网IP的ECS实例或集群提供访问互联网的代理服务。如果您在创建集群时未开启SNAT功能，可以通过ACK控制台为已有集群开启SNAT功能。本文介绍如何在ACK控制台为未开启公网访问的集群开启SNAT，以实现集群的公网访问能力。

背景信息

ACK暂不支持通过API方式为已有集群开启SNAT功能。关于SNAT相关的说明，请参见什么是NAT网关。

为已有集群开启SNAT公网访问能力的流程图及对应操作步骤如下。



操作步骤

1. 创建NAT网关。
   i. 登录NAT网关管理控制台。
   ii. 在左侧导航栏，单击**NAT网关**。
   iii. 在**NAT网关**页面，单击**创建NAT网关**。
   创建NAT网关的参数说明，请参见创建公网NAT网关实例。

   > **说明** 创建NAT网关时，地域和VPC的配置需要和集群的保持一致。

   成功创建NAT网关后，系统会在VPC的路由表中自动添加一条目标网段为0.0.0.0/0，下一跳为NAT网关的路由条目，用于将流量路由到NAT网关。

   > **注意** 默认只有VPC的第一个NAT网关，才会自动创建路由条目。对于多个NAT网关，您需手动为其配置路由。具体操作，请参见添加和删除路由表中的路由条目。

2. 创建弹性公网IP。
   在左侧导航栏，选择**弹性公网IP > 弹性公网IP**，然后在**弹性公网IP**页面，单击**创建弹性公网IP**。
   如果您已经有弹性公网IP，可以跳过这一步。
3. 绑定弹性公网IP到对应的NAT网关。
   i. 在**NAT网关**页面，找到目标NAT网关，然后在其右侧选择 **> 绑定弹性公网IP**。
   ii. 在**绑定弹性公网IP**对话框中，在**所在资源组**下拉列表中设置资源组，然后在**从已有弹性公网IP中选择**下拉列表中，选择创建的弹性公网IP。
   iii. 单击**确定**。
4. 为NAT网关创建SNAT条目。
   i. 在**NAT网关**页面，找到目标NAT网关，然后在其右侧，单击**管理**。
   ii. 单击**SNAT管理**页签，然后单击**创建SNAT条目**。
   iii. 在**创建SNAT条目**页面，配置以下参数，然后单击**确定创建**。
   关于更多创建SNAT条目的参数配置说明，请参见创建SNAT条目。

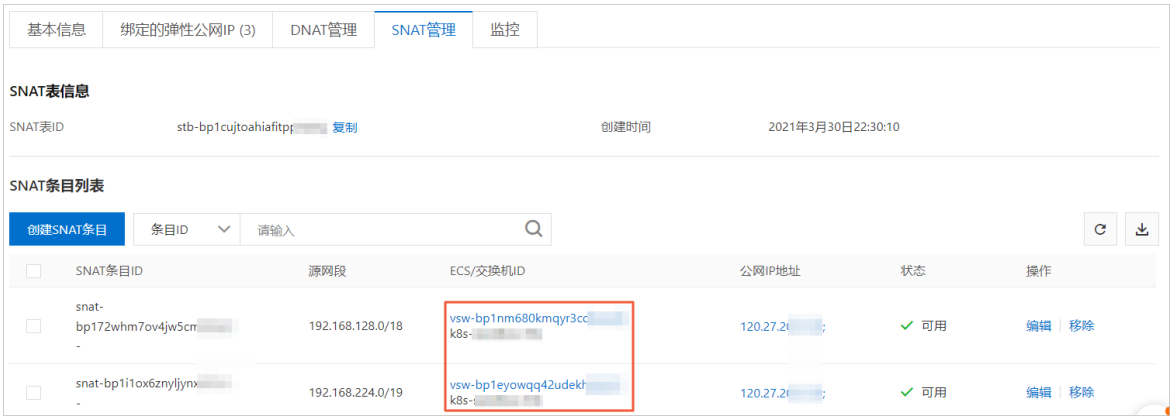
配置	说明

配置	说明
SNAT条目粒度	选中**交换机粒度**，然后选择相应的集群网络所使用的虚拟交换机。 ■ Terway网络：需要选中节点虚拟交换机和Pod虚拟交换机所对应ID。 ■ Flannel网络：只需选中节点虚拟交换机所对应的ID。 若网络插件为Terway或Flannel网络，请执行以下步骤查询集群网络所使用的节点虚拟交换机ID。 a. 登录容器服务管理控制台。 b. 在控制台左侧导航栏中，单击**集群**。 c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。 d. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。 e. 在目标节点池右侧**操作**列单击**详情**。 在**节点配置**区域查看节点虚拟交换机ID。  若网络插件为Terway，请执行以下步骤查询集群网络所使用的Pod虚拟交换机ID。 a. 登录容器服务管理控制台。 b. 在左侧导航栏，单击**集群**，然后在集群列表中单击目标集群名称或目标集群右侧**操作**列下的**详情**。 c. 在集群信息页面，单击**集群资源**页签，然后查看Pod虚拟交换机ID。 
选择公网IP地址	选择用来提供互联网访问的公网IP。

创建SNAT条目并配置SNAT规则之后，集群就开启了SNAT。您可以在NAT网关管理控制台看到NAT网关的详细信息，包括SNAT使用的弹性公网IP。下图为Terway网络模式下开启了SNAT公网访问的NAT网关实例。



单击NAT网关名称，在对应NAT网关实例的**SNAT管理**页签中，查看并确认已绑定了正确的集群所用虚拟交换机。下图为Terway网络模式下创建的SNAT条目信息。



执行结果

登录到集群的节点上，然后访问公网，可以查看网络正常，数据包无丢失。

```
ping taobao.com -c 2
PING taobao.com (140.[redacted]): 56 data bytes
64 bytes from 140.[redacted]: icmp_seq=0 ttl=43 time=7.267 ms
64 bytes from 140.[redacted]: icmp_seq=1 ttl=43 time=12.072 ms

--- taobao.com ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 7.267/9.669/12.072/2.403 ms
```

7.3.9. 容器网络FAQ

本文为您介绍容器网络中的常见问题。

- Terway网络模式下增加了虚拟交换机后，集群无法访问公网怎么办？
- 手动升级了Flannel镜像版本后，如何解决无法兼容1.16以上版本集群的问题？
- 如何解决Pod启动后存在时延的问题？
- 如何让Pod访问自己暴露的服务？
- 如何选择Kubernetes集群Terway和Flannel网络插件？
- 如何规划集群网络？
- ACK是否支持hostPort的端口映射？
- ACK集群是否支持VPC多路由表？
- 如何查看集群的网络类型及对应的虚拟交换机？
- 如何查看集群中使用的云资源？

- Kubernetes集群中Pod出现网络异常
- 如何确认Kubernetes集群内应用的出网地址
- 使用Terway网络的ENI模式出现网络异常
- Kubernetes管控集群链路异常的排查思路
- Terway网络场景中交换机的IP资源不足
- Kubernetes集群中访问LoadBalancer暴露出去的SLB地址不通
- 如何修改kube-proxy配置？
- 如何提升Linux连接跟踪Conntrack数量限制？
- 如何修改kube-proxy中IPVS负载均衡模式？
- 如何修改kube-proxy中IPVS UDP会话保持的超时时间？
- 如何解决IPv6双栈的部分常见问题？

Terway网络模式下增加了虚拟交换机后，集群无法访问公网怎么办？

问题现象：在Terway网络下，因Pod没有IP资源而手动增加虚拟交换机，在增加虚拟交换机后，发现集群不能正常访问公网。

问题原因：Pod IP所属的虚拟交换机不具备公网访问的能力。

解决方法：您可以通过NAT网关的SNAT功能，为Pod IP所属的虚拟交换机配置公网SNAT规则。更多信息，请参见为已有集群开启SNAT公网访问能力。

手动升级了Flannel镜像版本后，如何解决无法兼容1.16以上版本集群的问题？

问题现象：

集群版本升级到1.16之后，集群节点变成NotReady。

问题原因：

手动升级了Flannel版本，而并没有升级Flannel的配置，导致Kubelet无法识别。

解决方法：

1. 执行以下命令，编辑Flannel，增加 `cniVersion` 字段。

   ```
   kubectl edit cm kube-flannel-cfg -n kube-system
   ```

   返回结果中增加 `cniVersion` 字段。

   ```
   "name": "cb0",
   "cniVersion":"0.3.0",
   "type": "flannel",
   ```

2. 执行以下命令，重启Flannel。

   ```
   kubectl delete pod -n kube-system -l app=flannel
   ```

如何解决Pod启动后存在时延的问题？

问题现象：

Pod启动后网络需要延迟一会才能通信。

问题原因：

配置Network Policy会有一定的时延，关闭Network Policy后，就能解决该问题。

解决方法：

1. 执行以下命令，修改Terway的ConfigMap，增加禁用NetworkPolicy的配置。

```
kubectl edit cm -n kube-system eni-config
```

在返回结果中增加以下字段。

```
disable_network_policy: "true"
```

2. （可选）如果Terway版本不是最新的，请在控制台升级Terway版本。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页面选择**运维管理 > 组件管理**。
   v. 在**组件管理**页面单击**网络**页签，单击目标Terway组件区域的**升级**。
   vi. 在**提示**对话框中单击**确认**。
3. 执行以下命令，重启所有Terway的Pod。

```
kubectl delete pod -n kube-system -l app=terway-eniip
```

如何让Pod访问自己暴露的服务？

问题现象：

Pod无法访问自己暴露的服务，存在时好时坏或者调度到自己就出问题的现象。

问题原因：

Flannel目前默认不允许回环访问。

解决方法：

- 使用Headless Service暴露服务和访问，具体操作，请参见Headless Services。

  > **说明** 推荐使用此方法。

- 重建集群使用Terway的网络插件，具体操作，请参见使用Terway网络插件。
- 修改Flannel的配置，然后重建Flannel和Pod。

  > **说明** 不推荐此方法，可能会被后续升级覆盖。

  i. 执行以下命令编辑*cni-config.json*，增加 `hairpinMode: true` 。

  ```
  kubectl edit cm kube-flannel-cfg -n kube-system
  ```

  返回结果中增加以下字段。

  ```
  hairpinMode: true
  ```

  ii. 执行以下命令，重启Flannel。

  ```
  kubectl delete pod -n kube-system -l app=flannel
  ```

  iii. 删除并重新创建Pod。

如何选择Kubernetes集群Terway和Flannel网络插件？

下面为您详细介绍在ACK创建集群时使用的两种网络插件：Terway和Flannel。

在创建Kubernetes集群时，阿里云容器服务提供以下两种网络插件：

- 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。了解更多Terway网络插件的相关内容，请参见使用Terway网络插件。

如何规划集群网络？

在创建ACK集群时，需要指定专有网络VPC、虚拟交换机、 Pod网络CIDR（地址段）和Service CIDR（地址段）。建议您提前规划ECS地址、Kubernetes Pod地址和Service地址。详情请参见Kubernetes集群网络规划。

ACK是否支持hostPort的端口映射？

- 不支持hostPort。
- 容器服务ACK的Pod地址是可以直接被VPC中其他资源访问的，不需要额外的端口映射。
- 如果需要把服务暴露到外部，可以使用NodePort或者LoadBalancer类型的Service。

ACK集群是否支持VPC多路由表？

目前只有专有版集群支持配置VPC多路由表。具体操作，请参见使用VPC的多路由表功能。如果需ACK托管版集群支持使用VPC的多路由表功能，请提交工单。

如何查看集群的网络类型及对应的虚拟交换机？

ACK支持两种容器网络类型，分别是Flannel网络类型和Terway网络类型。

通过以下步骤查看您创建集群时所选择的网络类型

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在集群列表页面中，单击目标集群名称或者目标集群右侧操作列下的**详情**。
4. 单击**基本信息**页签，在**集群信息**区域查看集群的容器网络类型，即**网络插件**右侧的值。
   - 如果**网络插件**右侧显示**terway-eniip** ，则容器的网络类型为Terway网络。
   - 如果**网络插件**右侧显示**Flannel**，则容器的网络类型为Flannel网络。

通过以下步骤查看对应网络类型使用的节点虚拟交换机

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在目标节点池右侧**操作**列单击**详情**。
   在**节点配置**区域查看**节点虚拟交换机**ID。

通过以下步骤查询Terway网络类型使用的Pod虚拟交换机ID

> **说明** 只有Terway网络类型使用Pod虚拟交换机，Flannel网络类型无需使用Pod虚拟交换机。

1. 登录容器服务管理控制台。
2. 在左侧导航栏，单击**集群**，然后在集群列表中单击目标集群名称或目标集群右侧**操作**列下的**详情**。
3. 在集群信息页面，单击**集群资源**页签，然后查看**Pod虚拟交换机**ID。

如何查看集群中使用的云资源？

通过以下步骤查看集群中使用的云资源信息，包括虚拟机、虚拟专有网络VPC、Worker RAM角色等。

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，单击**集群**。
3. 在集群列表页面中，单击目标集群名称或者目标集群右侧操作列下的**详情**。
4. 单击**集群资源**页签，查看集群中使用的云资源信息。

如何修改kube-proxy配置？

ACK托管版默认部署kube-proxy-worker DaemonSet作为负载均衡，可通过其同名配置项kube-proxy-worker ConfigMap控制其参数。如果您使用的是ACK专有版，您集群中会额外部署kube-proxy-master DaemonSet和同名配置项，其会运行于Master节点上。

kube-proxy配置项均兼容社区KubeProxyConfiguration标准，您可以参考社区KubeProxyConfiguration标准进行自定义配置，更多信息，请参见kube-proxy Configuration。kube-proxy配置文件对格式要求严格，请勿漏填冒号和空格。修改kube-proxy配置操作如下：

- 如果您使用的是托管版集群，您需要修改kube-proxy-worker的配置。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
  v. 在顶部选择**kube-system**命名空间，然后单击配置项*kube-proxy-worker*右侧的**YAML编辑**。
  vi. 在**查看YAML**面板修改参数，然后单击**确定**。
  vii. 重建所有名为kube-proxy-worker的容器，使配置生效。

  > **注意** kube-proxy的重启不会导致已有运行业务中断，如果有并发业务发布，新启动的业务在kube-proxy的生效时间会略有延迟，请尽量于业务低峰期操作。

  a. 在集群管理页左侧导航栏中，选择**工作负载 > 守护进程集**。
  b. 在守护进程集列表中，找到并单击**kube-proxy-worker**。
  c. 在**kube-proxy-worker**页面中的**容器组**页签下对应容器组右侧，选择**更多 > 删除**，然后单击**确定**。重复操作删除所有容器组。删除容器组后，系统会自动重建所有容器。
- 如果您使用的是专有版集群，您需要修改kube-proxy-worker和kube-proxy-master的配置，然后删除kube-proxy-worker和kube-proxy-master Pod，该Pod自动重新创建后会使配置生效。具体操作，请参见上文。

如何提升Linux连接跟踪Conntrack数量限制？

内核日志（dmesg）中出现 `conntrack full` 的报错日志，说明Conntrack数量已经达到conntrack_max数量限制。您需要提升Linux连接跟踪Conntrack数量限制。

1. 执行 `conntrack -L` 命令，确认目前Conntrack中的各协议占用情况。
   - 如果出现大量TCP协议的占用，您需要确认具体业务，如果是短连接型应用可以考虑整改成长连接型。
   - 如果出现大量DNS的占用，您需要在ACK集群中使用NodeLocal DNSCache，提高DNS的性能，具体操作，请参见在ACK集群中使用NodeLocal DNSCache。
2. 如果Conntrack实际占用情况合理，或者您不希望对业务进行整改，您可以通过配置kube-proxy增加maxPerCore参数，调整连接跟踪数量限制。
   - 如果您使用的是托管版集群，您需要在kube-proxy-worker中增加maxPerCore参数，并设置其值为65536或更高的数值，然后删除kube-proxy-worker Pod，该Pod自动重新创建后会使配置生效。关于如何修改并删除kube-proxy-worker，请参见如何修改kube-proxy配置？。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: kube-proxy-worker
 namespace: kube-system
data:
 config.conf: |
  apiVersion: kubeproxy.config.k8s.io/v1alpha1
  kind: KubeProxyConfiguration
  featureGates:
   IPv6DualStack: true
  clusterCIDR: 172.20.0.0/16
  clientConnection:
   kubeconfig: /var/lib/kube-proxy/kubeconfig.conf
  conntrack:
   maxPerCore: 65536 // 需设置maxPerCore至合理值，此处65536为默认设置。
  mode: ipvs
# 其它略
```

- 如果您使用的是专有版集群，您需要在kube-proxy-worker和kube-proxy-master的中增加maxPerCore参数，并设置其值为65536或更高的数值，然后删除kube-proxy-worker和kube-proxy-master Pod，该Pod自动重新创建后会使配置生效。关于如何修改并删除kube-proxy-worker和kube-proxy-master，请参见如何修改kube-proxy配置？。

如何修改kube-proxy中IPVS负载均衡模式？

您可以通过修改kube-proxy中IPVS负载均衡模式来解决大量长连接的负载不均问题，具体操作如下：

1. 选择合适的调度算法。关于如何选择合适的调度算法，请参见parameter-changes。
2. 设置kube-proxy中的ipvs scheduler参数值为合理的调度算法。
   - 如果您使用的是托管版集群，您需要修改kube-proxy-worker的ipvs scheduler参数值为合理的调度算法，然后删除kube-proxy-worker Pod，该Pod自动重新创建后会使配置生效。关于如何修改并删除kube-proxy-worker，请参见如何修改kube-proxy配置？。

     ```
     apiVersion: v1
     kind: ConfigMap
     metadata:
      name: kube-proxy-worker
      namespace: kube-system
     data:
      config.conf: |
       apiVersion: kubeproxy.config.k8s.io/v1alpha1
       kind: KubeProxyConfiguration
       featureGates:
        IPv6DualStack: true
       clusterCIDR: 172.20.0.0/16
       clientConnection:
        kubeconfig: /var/lib/kube-proxy/kubeconfig.conf
       conntrack:
        maxPerCore: 65536
       mode: ipvs
       ipvs:
        scheduler: lc // 需设置scheduler成合理的调度算法。
     # 其它略
     ```

   - 如果您使用的是专有版集群，您需要修改kube-proxy-worker和kube-proxy-master中的ipvs scheduler参数值为合理的调度算法，然后删除kube-proxy-worker和kube-proxy-master Pod，该Pod自动重新创建后会使

配置生效。关于如何修改并删除kube-proxy-worker和kube-proxy-master，请参见如何修改kube-proxy配置？。

如何修改kube-proxy中IPVS UDP会话保持的超时时间？

如果您的ACK集群使用了kube-proxy IPVS模式，IPVS的默认会话保持策略会使UDP协议后端在摘除后五分钟内出现概率性丢包的问题。如果您业务依赖于CoreDNS，当CoreDNS组件升级或所在节点重启时，您可能会在五分钟内遇到业务接口延迟、请求超时等现象。

若您在ACK集群中的业务没有使用UDP协议，您可以通过降低IPVS UDP协议的会话保持的超时时间来减少解析延迟或失败的影响时间。具体操作如下：

> **说明** 若您的自有业务使用了UDP协议，请提交工单咨询。

- K8s 1.18及以上版本集群
  - 如果您使用的是托管版集群，您需要在kube-proxy-worker中修改udpTimeout参数值。然后删除kube-proxy-worker Pod，该Pod自动重新创建后会使配置生效。关于如何修改并删除kube-proxy-worker，请参见如何修改kube-proxy配置？。

    ```
    apiVersion: v1
    kind: ConfigMap
    metadata:
     name: kube-proxy-worker
     namespace: kube-system
    data:
     config.conf: |
      apiVersion: kubeproxy.config.k8s.io/v1alpha1
      kind: KubeProxyConfiguration
      // 其它不相关字段已省略。
      mode: ipvs
      // 如果ipvs键不存在，需要添加此键。
      ipvs:
       udpTimeout: 10s // 此处默认为300秒，调整成10秒可以将IPVS UDP类型后端摘除后丢包问题的影响时间缩短到10秒
    。
    ```

  - 如果您使用的是专有版集群，您需要在kube-proxy-worker和kube-proxy-master中修改udpTimeout参数值。然后删除kube-proxy-worker和kube-proxy-master Pod，该Pod自动重新创建后会使配置生效。关于如何修改并删除kube-proxy-worker，请参见如何修改kube-proxy配置？。
- K8s 1.16及以下版本集群
  此类版本集群的kube-proxy不支持udpTimeout参数，推荐使用运维编排服务OOS（Operation Orchestration Service）批量在所有集群机器上执行 `ipvsadm` 命令以调整UDP超时时间配置。命令如下：

  ```
  yum install -y ipvsadm
  ipvsadm -L --timeout > /tmp/ipvsadm_timeout_old
  ipvsadm --set 900 120 10
  ipvsadm -L --timeout > /tmp/ipvsadm_timeout_new
  diff /tmp/ipvsadm_timeout_old /tmp/ipvsadm_timeout_new
  ```

  关于OOS的批量操作实例介绍，请参见批量操作实例。

如何解决IPv6双栈的部分常见问题？

1. 问题现象：在kubectl中显示的Pod IP还是IPv4地址。
   解决方法：执行以下命令，展示Pod IPs字段，预期输出IPv6地址。

   ```
   kubectl get pods -A -o jsonpath='{range .items[*]}{@.metadata.namespace} {@.metadata.name} {@.status.podIPs[*].ip} {"\n"}{end}'
   ```

2. 问题现象：在kubectl中显示的Cluster IP还是IPv4地址。
   解决方法：
   i. 请确认spec.ipFamilyPolicy配置的不是SingleStack。
   ii. 执行以下命令，展示Cluster IPs字段，预期输出IPv6地址。

```
kubectl get svc -A -o jsonpath='{range .items[*]}{@.metadata.namespace}{@.metadata.name}{@.spec.ipFamilyPolicy}{@.spec.clusterIPs[*]}{"\n"}{end}'
```

3. 问题现象：无法通过IPv6地址访问Pod。
   问题原因：部分应用默认不监听IPv6地址，例如Nginx容器。
   解决方法：执行 `netstat -anp` 命令，确认Pod已监听IPv6地址。
   预期输出：

```
Proto Recv-Q Send-Q Local Address       Foreign Address      State     PID/Program name
tcp     0    0 127.0.XX.XX:10248      0.0.0.0:*          LISTEN    8196/kubelet
tcp     0    0 127.0.XX.XX:41935      0.0.0.0:*          LISTEN    8196/kubelet
tcp     0    0 0.0.XX.XX:111        0.0.0.0:*          LISTEN    598/rpcbind
tcp     0    0 0.0.XX.XX:22         0.0.0.0:*          LISTEN    3577/sshd
tcp6    0    0 :::30500           :::*               LISTEN    1916680/kube-proxy
tcp6    0    0 :::10250           :::*               LISTEN    8196/kubelet
tcp6    0    0 :::31183           :::*               LISTEN    1916680/kube-proxy
tcp6    0    0 :::10255           :::*               LISTEN    8196/kubelet
tcp6    0    0 :::111             :::*               LISTEN    598/rpcbind
tcp6    0    0 :::10256           :::*               LISTEN    1916680/kube-proxy
tcp6    0    0 :::31641           :::*               LISTEN    1916680/kube-proxy
udp     0    0 0.0.0.0:68         0.0.0.0:*                    4892/dhclient
udp     0    0 0.0.0.0:111        0.0.0.0:*                    598/rpcbind
udp     0    0 127.0.0.1:323      0.0.0.0:*                    6750/chronyd
udp     0    0 0.0.0.0:720        0.0.0.0:*                    598/rpcbind
udp6    0    0 :::111             :::*                         598/rpcbind
udp6    0    0 ::1:323            :::*                         6750/chronyd
udp6    0    0 fe80::216:XXXX:fe03:546 :::*                    6673/dhclient
udp6    0    0 :::720             :::*                         598/rpcbind
```

   `Proto` 显示为 `tcp` 即监听IPv4地址，显示为 `tcp6` 即监听IPv6地址。
4. 问题现象：通过IPv6地址可以在集群内访问Pod，但无法从公网访问。
   问题原因：该IPv6地址可能未配置公网带宽。
   解决方法：配置该IPv6地址的公网带宽。具体操作，请参见开通IPv6公网带宽。
5. 问题现象：无法通过IPv6 Cluster IP访问Pod。
   解决方法：
   i. 请确认spec.ipFamilyPolicy配置的不是SingleStack。

ii. 执行 `netstat -anp` 命令，确认Pod已监听IPv6地址。
预期输出：

```
Proto Recv-Q Send-Q Local Address       Foreign Address      State     PID/Program name
tcp     0    0 127.0.XX.XX:10248      0.0.0.0:*          LISTEN    8196/kubelet
tcp     0    0 127.0.XX.XX:41935      0.0.0.0:*          LISTEN    8196/kubelet
tcp     0    0 0.0.XX.XX:111          0.0.0.0:*        LISTEN    598/rpcbind
tcp     0    0 0.0.XX.XX:22           0.0.0.0:*        LISTEN    3577/sshd
tcp6    0    0 :::30500               :::*             LISTEN    1916680/kube-proxy
tcp6    0    0 :::10250               :::*             LISTEN    8196/kubelet
tcp6    0    0 :::31183               :::*             LISTEN    1916680/kube-proxy
tcp6    0    0 :::10255               :::*             LISTEN    8196/kubelet
tcp6    0    0 :::111               :::*             LISTEN    598/rpcbind
tcp6    0    0 :::10256               :::*             LISTEN    1916680/kube-proxy
tcp6    0    0 :::31641               :::*             LISTEN    1916680/kube-proxy
udp     0    0 0.0.0.0:68           0.0.0.0:*                  4892/dhclient
udp     0    0 0.0.0.0:111          0.0.0.0:*                  598/rpcbind
udp     0    0 127.0.0.1:323        0.0.0.0:*                  6750/chronyd
udp     0    0 0.0.0.0:720          0.0.0.0:*                  598/rpcbind
udp6    0    0 :::111               :::*                     598/rpcbind
udp6    0    0 ::1:323              :::*                      6750/chronyd
udp6    0    0 fe80::216:XXXX:fe03:546 :::*                       6673/dhclient
udp6    0    0 :::720               :::*                     598/rpcbind
```

`Proto` 显示为 `tcp` 即监听IPv4地址，显示为 `tcp6` 即监听IPv6地址。

7.4. Service管理

7.4.1. Service的负载均衡配置注意事项

当Service的类型设置为 `Type=LoadBalancer` 时，容器服务ACK的CCM（Cloud Controller Manager）组件会为该Service创建或配置阿里云负载均衡SLB（Server Load Balancer），包括含SLB、监听、后端服务器组等资源。本文介绍配置Service负载均衡的注意事项以及CCM的资源更新策略。

指定已有SLB时注意事项

- **哪些SLB可以被复用？**
  - 仅支持复用通过SLB控制台创建的SLB，不支持复用CCM自动创建的SLB。
  - 如果您需要在Kubernetes集群中复用私网类型的SLB，则该SLB需要和Kubernetes集群处于同一VPC下。
- CCM只为 `Type=LoadBalancer` 类型的Service配置SLB，对于非LoadBalancer类型的Service则不会为其配置负载均衡。

  > **注意** 当 `Type=LoadBalancer` 的Service变更为 `Type!=LoadBalancer` 时，CCM会删除为该SLB添加的配置，从而造成无法通过该SLB访问Service。

- CCM使用声明式API，会在一定条件下自动根据Service的配置刷新SLB配置。当 `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners:` 设置为 `true` 时，您自行在SLB控制台上修改的配置均存在被覆盖的风险。

  > **注意** 请勿在SLB控制台上手动修改Kubernetes创建并维护的SLB的任何配置，否则有配置丢失的风险，造成Service不可访问。

CCM管理SLB时注意事项

- CCM只为 `Type=LoadBalancer` 类型的Service创建SLB，对于非LoadBalancer类型的Service则不会为其创建或配置负载均衡。

  > **注意** 当 `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。

- CCM使用声明式API，会在一定条件下自动根据Service的配置更新SLB配置。您自行在SLB控制台上修改的配置均存在被覆盖的风险。

  > **注意** 请勿在SLB控制台上手动修改Kubernetes创建并维护的SLB的任何配置，否则有配置丢失的风险，造成Service不可访问。

SLB更新策略

ACK支持为Serivce指定一个已有的SLB，或者让CCM自动创建新的SLB。两种方式在SLB的资源更新策略方面存在一些差异，如下表所示。

资源对象	指定已有SLB	CCM管理SLB
SLB	设置annotation：`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id`。 • CCM会使用该SLB作为Service的负载均衡，并根据其他annotation配置SLB，自动为SLB创建多个虚拟服务器组。 • 当Service删除时，CCM不会删除您通过ID指定的已有SLB。	• CCM会根据Service的配置，自动创建和配置SLB、监听、虚拟服务器组等资源，所有资源由CCM管理。 • 当Service删除时，CCM会删除自动创建的SLB。
监听	设置annotation：`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners:`： • 如果设置为false，CCM不会为SLB管理任何监听配置。 • 如果设置为true，CCM会根据Service配置管理监听；如果监听已经存在，则CCM会覆盖已有监听。	CCM会根据Service的配置，自动创建和配置监听策略。

资源对象	指定已有SLB	CCM管理SLB

后端服务器组	当Service对应的后端Endpoint或者集群节点发生变化时，CCM会自动更新SLB的后端虚拟服务器组。
	• 根据Service模式的不同，后端服务器组的更新策略也有所不同。
	  ◦ Cluster模式（ spec.externalTrafficPolicy = Cluster ）：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）。
	  注意 SLB存在配额限制，限制了每个ECS能够使用的SLB个数，这种方式会快速消耗该配额。当配额耗尽后，会造成Service Reconcile失败。解决办法：使用Local模式的Service。
	  ◦ Local模式（ spec.externalTrafficPolicy = Local ）：CCM默认只会将Service对应的Pod所在节点加入到SLB后端。这样可以降低SLB配额的消耗速度，同时支持四层源IP保留。
	• 任何情况下CCM都不会将Master节点作为SLB的后端。
	• CCM默认不会从SLB后端移除被驱逐（ kubectl drain ）或停止调度（ kubectl 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 。
	  注意 如果是v1.9.3.164-g2105d2e-aliyun之前的版本，CCM默认会从SLB后端移除被驱逐或停止调度的节点。	

配额限制

VPC

- 集群中一个节点对应一条路由表项，VPC默认情况下仅支持48条路由表项，如果集群节点数目多于48个，请提交工单给VPC产品。

  说明 您可以在提交工单时，说明需要修改 vpc_quota_route_entrys_num 参数，用于提升单个路由表可创建的自定义路由条目的数量。

- 更多VPC使用限制，请参见限制与配额。
  查询VPC配额，请参见专有网络VPC配额管理。

SLB

- CCM会为 Type=LoadBalancer 类型的Service创建SLB。默认情况下一个用户可以保留60个SLB实例。如果需要创建的SLB数量大于60，请提交工单给SLB产品。

  说明 您可以在提交工单时，说明需要修改 slb_quota_instances_num 参数，用于提高您可保有的SLB实例个数。

- CCM会根据Service的配置将ECS挂载到SLB后端服务器组中。
  - 默认情况下一个ECS实例可挂载的后端服务器组的数量为50个，如果一台ECS需要挂载到更多的后端服务器组中，请提交工单给SLB产品。

    说明 您可以在提交工单时，说明需要修改 slb_quota_backendserver_attached_num 参数，用于提高同一台服务器可以重复添加为SLB后端服务器的次数。

- 默认情况下一个SLB实例可以挂载200个后端服务器，如果需要挂载更多的后端服务器，请提交工单给SLB产品。

  > **说明** 您可以在提交工单时，说明需要修改 slb_quota_backendservers_num 参数，提高每个SLB实例可以挂载的服务器数量。

- CCM会根据Service中定义的端口创建SLB监听。默认情况下一个SLB实例可以添加50个监听，如需添加更多监听，请提交工单给SLB产品。

  > **说明** 您可以在提交工单时，说明需要修改 slb_quota_listeners_num 参数，用于提高每个实例可以保有的监听数量。

- 更多SLB使用限制请参见使用限制。
  负载均衡SLB配额查询请参见负载均衡SLB配额管理。

7.4.2. 通过Annotation配置负载均衡

通过Service YAML文件中的Annotation（注解），可以实现丰富的负载均衡功能。本文从SLB、监听和后端服务器组三种资源维度介绍通过注解可以对SLB进行的常见配置操作。

注解使用说明

- 注解的内容是区分大小写。
- 自2019年09月11日起， annotations 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，您无需做任何修改。

SLB

SLB的典型操作

- 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
```

- 指定负载均衡规格
  负载均衡规格可参见CreateLoadBalancer。通过该参数可以创建指定规格的SLB，或者更新已有SLB的规格。

  > **注意** 如果您通过SLB控制台修改SLB规格，可能会被CCM修改回原规格，请谨慎操作。

```
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*true*。

    > **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: 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
```

- 创建按带宽付费的负载均衡
  - 仅支持公网类型的负载均衡实例。
    其他限制，请参见修改公网负载均衡实例的计费方式。
  - 以下两项Annotation必选。
    此处提供的是示例值，请根据实际业务需要自行设置。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-type: "paybybandwidth"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-bandwidth: "2"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 443
  protocol: TCP
  targetPort: 443
 selector:
  run: nginx
 type: LoadBalancer
```

> **说明** service.beta.kubernetes.io/alibaba-cloud-loadbalancer-bandwidth为带宽峰值。

- 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。
  - 虚拟交换机必须与Kubernetes集群属于同一个VPC。
  - 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
```

- 创建IPv6类型的负载均衡
  - 集群的kube-proxy代理模式需要是IPVS。
  - 生成的IPv6地址仅可在支持IPv6的环境中访问。
  - 创建后IP类型不可更改。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version: "ipv6"
 name: nginx
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

- 为负载均衡开启删除保护
  默认开启删除保护。

> **注意** 对于LoadBalancer类型的Service创建的负载均衡，如果手动在SLB控制台开启了删除保护，仍可通过 `kubectl delete svc {your-svc-name}` 的方式删除Service关联的负载均衡。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-delete-protection: "on"
 name: nginx
spec:
 externalTrafficPolicy: Local
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

- 为负载均衡开启配置修改保护
  默认开启配置修改保护。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-modification-protection: "ConsoleProtection"
 name: nginx
spec:
 externalTrafficPolicy: Local
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

- 指定负载均衡名称

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-name: "your-svc-name"
 name: nginx
spec:
 externalTrafficPolicy: Local
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

- 指定负载均衡所属的资源组
  在阿里云资源管理平台查询资源组ID，然后使用以下Annotation为负载均衡实例指定资源组。

> **说明** 资源组ID创建后不可被修改。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-resource-group-id: "rg-xxxx"
 name: nginx
spec:
 externalTrafficPolicy: Local
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

监听

监听的典型操作

- 为TCP类型的负载均衡配置会话保持时间
  - 参数 `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-persistence-timeout` 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-persistence-timeout: "1800"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 443
  protocol: TCP
  targetPort: 443
 selector:
  run: nginx
 type: LoadBalancer
```

- 为HTTP和HTTPS协议的监听配置会话保持（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
```

- 为负载均衡配置访问控制策略组
  - 需要先在阿里云负载均衡控制台上创建一个负载均衡访问控制策略组，然后记录该访问控制策略组ID（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
```

- 为负载均衡设置调度算法
  - rr（默认值）：轮询，按照访问顺序依次将外部请求依序分发到后端服务器。
  - wrr：加权轮询，权重值越高的后端服务器，被轮询到的次数（概率）也越高。
  - wlc：加权最小连接数，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。

```
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HTTP类型的监听

```
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类型的监听
  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。

  > **说明** HTTPS请求会在SLB层解密，然后以HTTP请求的形式发送给后端的Pod。

```
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: 80
 selector:
  run: nginx
 type: LoadBalancer
```

- 创建带健康检查的监听

- 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
```

- 为监听设置连接优雅中断。
  仅支持TCP和UDP协议。如需设置连接优雅中断，以下两项Annotation必选。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain: "on"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain-timeout: "30"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 type: LoadBalancer
```

- 为监听配置通过X-Forwarded-Proto头字段获取SLB的监听协议
  仅支持HTTP和HTTPS协议。

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

- 为监听设置连接空闲超时时间
  仅支持HTTP和HTTPS协议。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-port: "http:80"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-idle-timeout: "30"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 type: LoadBalancer
```

- 关闭监听的HTTP2特性

仅支持HTTPS协议。

```
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}"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-http2-enabled: "off"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 443
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 type: LoadBalancer
```

后端服务器组

后端服务器的典型操作

- 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关系。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-backend-label: "failure-domain.beta.kubernetes.io/zo
ne=ap-southeast-5a"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 443
  protocol: TCP
  targetPort: 443
 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。
  - Local模式需要设置调度策略为加权轮询wrr。

> **说明**
> 在v1.9.3.164-g2105d2e-aliyun及之后版本，外部流量策略设置为Local模式的服务会自动根据Node上的Pod数量为Node设置权重，权重计算规则请参见Local模式下如何自动设置Node权重？。

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

- 移除SLB后端unschedulable状态的节点
  - kubectl cordon 与 kubectl drain 命令会将节点置为unschedulable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unschedulable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unschedulable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:
 externalTrafficPolicy: Local
 ports:
 - name: http
  port: 30080
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

- 直接将Pod ENI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** 您也可以手动将 service.beta.kubernetes.io/backend-type："eni" 中的 eni 设置为 ecs 将ECS挂载到SLB后端。

- 复用已有虚拟服务器组
  支持通过annotation： service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vgroup-port 复用已有虚拟服务器组。该annotation仅在复用已有SLB场景下生效。具体的使用示例，请参见通过CCM跨集群部署服务。
- 设置Service接收流量的权重。
  在多个Service复用同一个SLB的场景下，支持通过annotation： service.beta.kubernetes.io/alibaba-cloud-loadbalancer-weight 设置当前Service接收流量的权重。该Annotation仅在复用已有虚拟服务器组场景下生效。具体的使用示例，请参见通过CCM跨集群部署服务。

常用注解

- **SLB常用注解**

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type	string	取值：*internet*或者*intranet*。 ○ *internet*：服务通过公网访问，此为默认值。对应SLB的**地址类型**必须为**公网**。 ○ *intranet*：服务通过私网访问。对应SLB的**地址类型**必须为**私网**。	*internet*	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-type	string	取值：*paybytraffic*或者*paybybandwidth*。	*paybytraffic*	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
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	v1.9.3.81-gca19cd4-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec	string	负载均衡实例的规格。可参见：CreateLoadBalancer。	无	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-master-zoneid	string	主后端服务器的可用区ID。	无	v1.9.3.10-gfb99107-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-slave-zoneid	string	备后端服务器的可用区ID。	无	v1.9.3.10-gfb99107-aliyun及以上版本
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	v1.9.3.81-gca19cd4-aliyun及以上版本
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*	v1.9.3.10-gfb99107-aliyun及以上版本
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	v1.9.3及以上版本
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	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version	string	负载均衡实例的IP版本，取值：ipv4或ipv6。	ipv4	v1.9.3.220-g24b1885-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-delete-protection	string	负载均衡删除保护，取值：on或off。	on	v1.9.3.313-g748f81e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-modification-protection	string	负载均衡配置修改保护，取值：ConsoleProtection或NonProtection。	ConsoleProtection	v1.9.3.313-g748f81e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-resource-group-id	string	负载均衡所属资源组ID。	无	v1.9.3.313-g748f81e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-name	string	负载均衡实例名称。	无	v1.9.3.313-g748f81e-aliyun及以上版本

- **监听的常用注解**

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-scheduler	string	调度算法。取值*wrr*、*wlc*或*rr*。 ○ *wrr*：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*：默认取值，按照访问顺序依次将外部请求依序分发到后端服务器。	*rr*	v1.9.3及以上版本
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	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
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：CreateLoadBalancerTCPListener。	*0*	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session	string	是否开启会话保持。取值：*on*或*off*。 **说明** 仅对HTTP和HTTPS协议的监听生效。 可参见：CreateLoadBalancerHTTPListener和CreateLoadBalancerHTTPSListener。	*off*	v1.9.3及以上版本
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：CreateLoadBalancerHTTPListener和CreateLoadBalancerHTTPSListener。	无	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie-timeout	string	Cookie超时时间。取值：*1s~86400s*。 **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：CreateLoadBalancerHTTPListener和CreateLoadBalancerHTTPSListener。	无	v1.9.3及以上版本
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：CreateLoadBalancerHTTPListener和CreateLoadBalancerHTTPSListener。	无	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id	string	阿里云上的证书ID。您需要在SLB控制台先上传证书。	无	v1.9.3.164-g2105d2e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-flag	string	取值：*on*或*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。	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-type	string	健康检查类型，取值：*tcp*或*http*。 可参见：CreateLoadBalancerTCPListener。	*tcp*	v1.9.3及以上版本
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：CreateLoadBalancerTCPListener。	无	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-connect-port	string	健康检查使用的端口。取值：1~65535。	无	v1.9.3及以上版本
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：CreateLoadBalancerTCPListener。	*3*	v1.9.3及以上版本
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：CreateLoadBalancerTCPListener。	*3*	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-interval	string	健康检查的时间间隔。 取值：*1s~50s* 可参见：CreateLoadBalancerTCPListener。	*2*	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
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：*1s~300s* **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：CreateLoadBalancerTCPListener。	*5*	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
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：*1s~300s* **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：CreateLoadBalancerTCPListener。	*5*	v1.9.3及以上版本
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	v1.9.3及以上版本
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*	v1.9.3及以上版本

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-status	string	是否开启访问控制功能。取值：*on*或*off*。	*off*	v1.9.3.164-g2105d2e-aliyun及以上版本
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	v1.9.3.164-g2105d2e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-type	string	访问控制类型。 取值：*white*或*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	v1.9.3.164-g2105d2e-aliyun及以上版本
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	v1.9.3.164-g2105d2e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain	string	是否开启连接优雅中断，取值：*on*或*off*。	无	v2.0.1及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain-timeout	string	设置连接优雅中断超时时间。单位：秒。取值：*10~900*。	无	v2.0.1及以上版本

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-xforwardedfor-proto	string	配置是否通过X-Forwarded-Proto头字段获取SLB的监听协议。取值：*on*或*off*。	off	v2.1.0及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-idle-timeout	string	为监听设置连接空闲超时时间。单位：秒。取值范围：*1~60*。	15	v2.1.0及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-http2-enabled	string	是否开启HTTP2特性。取值：*on*或*off*。	on	v2.1.0及以上版本

- **后端服务器组的常用注解**

注解	类型	描述	默认值	支持的CCM版本
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	v1.9.3及以上版本
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*	v1.9.3及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-remove-unscheduled-backend	string	从SLB后端移除SchedulingDisabled Node。取值：*on*或*off*。	off	v1.9.3.164-g2105d2e-aliyun及以上版本

注解	类型	描述	默认值	支持的CCM版本
service.beta.kubernetes.io/backend-type	string	SLB后端服务器类型。 取值： • eni：将Pod挂载到SLB后端，仅Terway网络模式下生效，可以提高网络转发性能。 • ecs：将ECS挂载到SLB后端。	Flannel网络模式：默认值为ecs。 Terway网络模式： • 2020年8月10日之前创建的Terway集群默认值为ecs。 • 2020年8月10日之后创建的Terway集群默认值为eni。	v1.9.3.164-g2105d2e-aliyun及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vgroup-port	string	复用已有虚拟服务器组，仅在复用已有SLB场景下生效。如果有多个端口及虚拟服务器组的组合，可以通过半角逗号（,）分隔。	无	v2.0.1及以上版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-weight	string	在多个Service复用同一个SLB的场景下，设置当前Service接收流量的权重。该Annotation仅在复用已有虚拟服务器组场景下生效。	无	v2.0.1及以上版本

相关文档

- Service注解不生效如何处理？
- Service FAQ

7.4.3. 通过使用已有SLB的服务公开应用

通过阿里云负载均衡SLB（Server Load Balancer）暴露的服务（Service），在集群外可通过SLB域名或 <IP:服务端口> 的方式访问服务，在集群内可通过 <服务名:服务端口> 的方式访问服务。本文以Nginx应用为例，介绍如何通过使用已有SLB的服务来公开应用。

前提条件

已存在通过SLB控制台创建的SLB实例，且该实例与Kubernetes集群处于同一地域。如果没有创建，请参见创建实例。

背景信息

如果您集群的Cloud Controller Manager（CCM）组件版本大于等于v1.9.3，对于指定已有SLB，CCM默认不再为该SLB处理监听。您可以通过设置 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners: "true" 参数来启用监听配置，或者手动配置该SLB的监听规则。

查看CCM版本的方法：

- **使用控制台**：在集群组件管理页面查看CCM组件版本。
  i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏，单击**集群**。

iii. 在**集群列表**页面，在目标集群右侧，选择**更多 > 系统组件管理**，进入**组件管理**页签查看CCM的版本信息。

- **使用kubectl命令行**（仅适用于专有版集群）：执行以下命令查看CCM组件版本。

```
kubectl get pod -n kube-system -o yaml|grep image:|grep cloud-con|uniq
```

注意事项

- 被复用的SLB需要满足以下限制条件：
  - 支持复用通过SLB控制台创建的SLB，不支持复用CCM自动创建的SLB。
  - 如果您需要在Kubernetes集群中复用私网类型的SLB，则该SLB需要和Kubernetes集群处于同一VPC下。
  - 复用SLB的地址类型必须与服务的访问类型一致。当服务为**公网访问**（即 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type: "internet" ）时，所用SLB的**地址类型**必须为**公网**；当服务为**内部访问**（即 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type: "intranet" ）时，所用SLB的**地址类型**必须为**私网**。
  - 多个服务不能同时使用一个SLB的同一个监听端口。
- CCM只为 Type=LoadBalancer 类型的服务配置SLB。对于非LoadBalancer类型的服务，则不会为其配置负载均衡。

  > **注意** 当 Type=LoadBalancer 的服务变更为其他类型时，CCM会删除为该SLB添加的配置，从而造成无法通过该SLB访问服务。

- CCM使用声明式API，会在一定条件下自动根据服务的配置刷新SLB配置。当 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners: 设置为 true 时，您自行在SLB控制台上修改的配置均存在被覆盖的风险。

  > **注意** 请勿在SLB控制台上手动修改Kubernetes创建并维护的SLB的任何配置，否则有配置丢失的风险，造成服务不可访问。

- 已创建的LoadBalancer类型Service不支持重新指定SLB。如果您需要更换SLB，请重新创建Service。

SLB配额限制

- CCM会为 Type=LoadBalancer 类型的Service创建SLB。默认情况下一个用户可以保留60个SLB实例。如果需要创建的SLB数量大于60，请提交工单给SLB产品。

  > **说明** 您可以在提交工单时，说明需要修改 slb_quota_instances_num 参数，用于提高您可保有的SLB实例个数。

- CCM会根据Service中定义的端口创建SLB监听。默认情况下一个SLB实例可以添加50个监听，如需添加更多监听，请提交工单给SLB产品。

  > **说明** 您可以在提交工单时，说明需要修改 slb_quota_listeners_num 参数，用于提高每个实例可以保有的监听数量。

- CCM会根据Service的配置将ECS挂载到SLB后端服务器组中。
  - 默认情况下一个ECS实例可挂载的后端服务器组的数量为50个，如果一台ECS需要挂载到更多的后端服务器组中，请提交工单给SLB产品。

    > **说明** 您可以在提交工单时，说明需要修改 slb_quota_backendserver_attached_num 参数，用于提高同一台服务器可以重复添加为SLB后端服务器的次数。

- 默认情况下一个SLB实例可以挂载200个后端服务器，如果需要挂载更多的后端服务器，请提交工单给SLB产品。

  > **说明** 您可以在提交工单时，说明需要修改 slb_quota_backendservers_num 参数，提高每个SLB实例可以挂载的服务器数量。

关于更多SLB使用限制，请参见使用限制。关于如何查询负载均衡SLB配额，请参见负载均衡SLB配额管理。

步骤一：部署示例应用

以下应用部署通过**kubectl**命令行方式进行。如果您需要通过控制台部署应用，请参见创建无状态工作负载Deployment。

1. 使用以下示例应用的YAML内容，创建名为*my-nginx.yaml*文件。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
 name: my-nginx   #示例应用的名称。
 labels:
  app: nginx
spec:
 replicas: 3    #设置副本数量。
 selector:
  matchLabels:
   app: nginx   #对应服务中Selector的值需要与其一致，才可以通过服务公开此应用。
 template:
  metadata:
   labels:
    app: nginx
  spec:
  #  nodeSelector:
  #   env: test-team
   containers:
   - name: nginx
    image: registry.aliyuncs.com/acs/netdia:latest   #替换为您实际的镜像地址，格式为：<image_name:tags>。
    ports:
    - containerPort: 80                  #需要在服务中暴露该端口。
```

2. 执行以下命令，部署示例应用my-nginx。

```
kubectl apply -f my-nginx.yaml
```

3. 执行以下命令，确认示例应用状态正常。

```
kubectl get deployment my-nginx
```

返回结果示例：

```
NAME    READY  UP-TO-DATE  AVAILABLE  AGE
my-nginx  3/3   3       3       50s
```

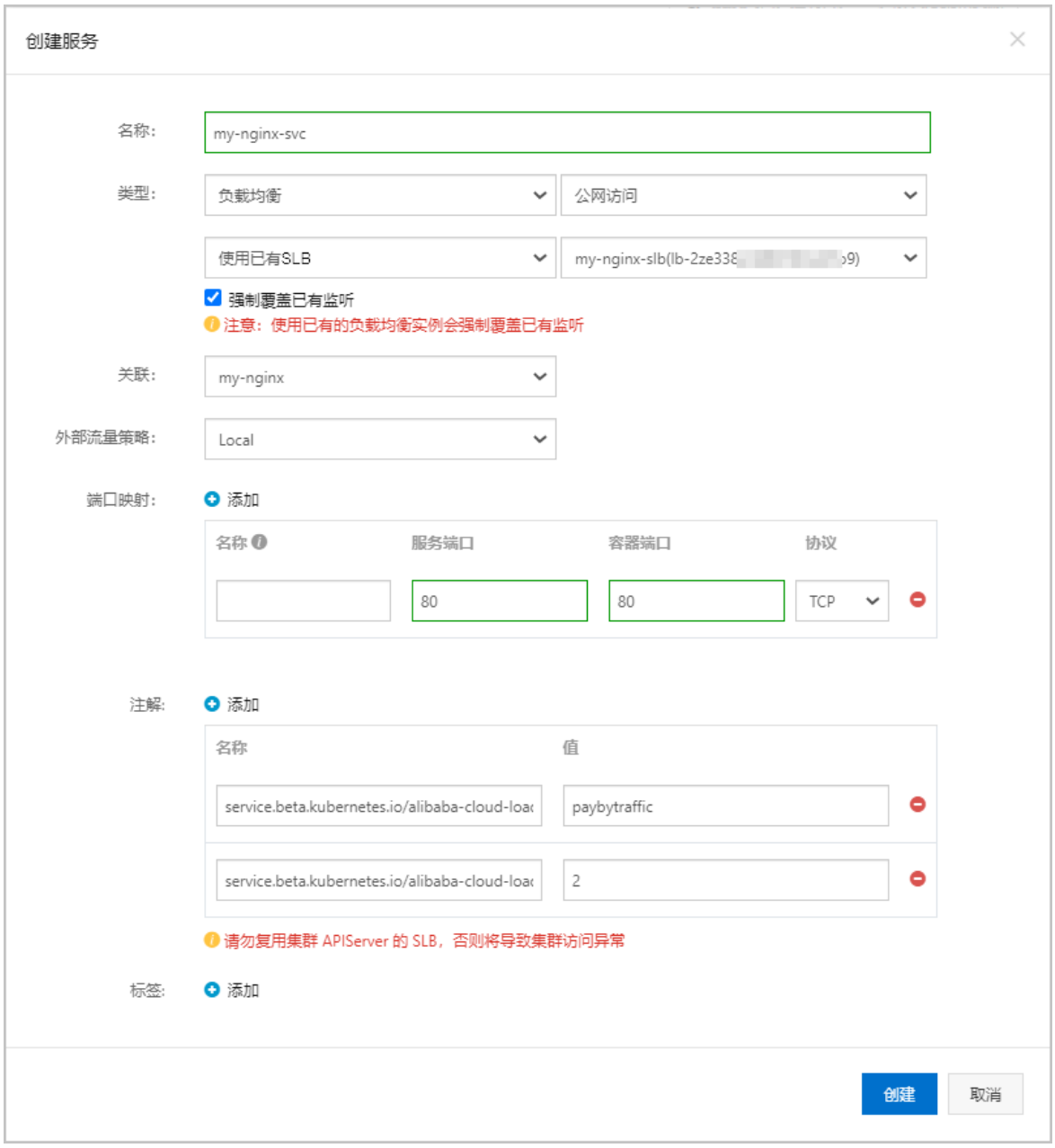
步骤二：通过使用已有SLB的服务公开应用

您可以通过控制台和**kubectl**两种方式来创建LoadBalancer类型的服务，并通过其公开应用。

控制台方式

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在**服务**页面，单击右上角的**创建**。
6. 在**创建服务**对话框中，设置服务的相关参数。



参数	描述
名称	输入服务的名称，本示例为my-nginx-svc。

参数	描述
类型	选择服务类型，即服务访问的方式。 ○ 依次选择：**负载均衡 > 公网访问 > 使用已有SLB** > 需要使用的SLB实例。 ○ **强制覆盖已有监听**：选择是否强制覆盖SLB上的已有监听或者为没有监听的SLB自动创建监听。本示例中的SLB实例为新创建，需要为其创建监听，因此勾选该选项。 **说明** ■ 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。
关联	选择关联该服务要绑定的后端应用，本示例为my-nginx。若不进行关联部署，则不会创建相关的Endpoints对象，您也可自己进行绑定，请参见services-without-selectors。
外部流量策略	设置外部流量策略，本示例为Local。 ○ Local：流量只发给本机的Pod。 ○ Cluster：流量可以转发到集群中其他节点上的Pod。 **说明** 您的服务类型为**节点端口**或**负载均衡**时，才能设置**外部流量策略**。
端口映射	添加服务端口（对应Service YAML文件中的 port ）和容器端口（对应Service YAML文件中的 targetPort ），容器端口需要与后端的Pod中暴露的容器端口一致。本示例皆为80。
注解	为该服务添加一个注解（Annotation），配置负载均衡的参数。您可以选择**自定义注解**或**阿里云注解**。本示例中，将该服务的收费方式设置为按带宽收费，带宽峰值设置为2 Mbit/s，从而控制服务的流量。更多注解请参见通过Annotation配置负载均衡。 ○ **类型**：阿里云注解 ○ **名称**： service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-type 、 service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth ○ **值**：paybybandwidth、2
标签	为该服务添加一个标签，标识该服务。

7. 单击**创建**。
   在**服务**页面，可以看到新创建的服务。



8. 单击该服务在**外部端点**列的**39.106.XX.XX:80**，访问示例应用。

Kubectl方式

1. 使用以下示例服务的YAML内容，创建名为*my-nginx-svc.yaml*的文件。

- 修改 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id ，请替换*${YOUR_LB_ID}*为您通过负载均衡管理控制台创建的SLB实例ID。
- 使用已有的SLB实例时，默认情况下不会为该SLB创建监听或覆盖已有监听。如有需要，请设置 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners 为 true 。本示例中，SLB实例为新创建，需要为其创建监听，因此设为 true 。更多注解请参见通过Annotation配置负载均衡。
- 将selector修改为*my-nginx.yaml*示例应用文件中matchLabels的值（即： app: nginx ），从而将该服务关联至后端应用。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: ${YOUR_LB_ID}
  service.beta.kubernetes.io/alicloud-loadbalancer-force-override-listeners: 'true'
 labels:
  app: nignx
 name: my-nginx-svc
 namespace: default
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: LoadBalancer
```

2. 执行以下命令创建名为my-nginx-svc的服务，并通过其公开应用。

```
kubectl apply -f my-nginx-svc.yaml
```

3. 执行以下命令确认LoadBalancer类型的服务创建成功。

```
kubectl get svc my-nginx-svc
```

返回结果示例：

```
NAME       TYPE       CLUSTER-IP   EXTERNAL-IP    PORT(S)     AGE
my-nginx-svc  LoadBalancer  172.21.5.82  39.106.XX.XX   80:30471/TCP  5m
```

4. 执行curl <YOUR-External-IP>命令访问示例应用，请将*YOUR-External-IP*替换为上面获取到的 EXTERNAL-IP 地址。

```
curl 39.106.XX.XX
```

返回结果示例：

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

相关文档

- 通过Annotation配置负载均衡
- 通过使用自动创建SLB的服务公开应用

7.4.4. 通过使用自动创建SLB的服务公开应用

当您没有可用的SLB时，Cloud Controller Manager（CCM）组件可以为LoadBalancer类型服务自动创建SLB，并对其进行管理。本文以Nginx应用为例，介绍如何通过使用自动创建SLB的服务来公开应用。

注意事项

- CCM只为 `Type=LoadBalancer` 类型的服务配置SLB。对于非LoadBalancer类型的服务，则不会为其配置负载均衡。

  > **注意** 当 `Type=LoadBalancer` 的服务变更为其他类型时，CCM会删除为该SLB添加的配置，从而造成无法通过该SLB访问服务。

- CCM使用声明式API，会在一定条件下自动根据服务的配置刷新SLB配置。您自行在SLB控制台上修改的配置均存在被覆盖的风险。

  > **注意** 请勿在SLB控制台上手动修改Kubernetes创建并维护的SLB的任何配置，否则有配置丢失的风险，造成服务不可访问。

- 已创建的LoadBalancer类型Service不支持重新指定SLB。如果您需要更换SLB，请重新创建Service。

SLB配额限制

- CCM会为 `Type=LoadBalancer` 类型的Service创建SLB。默认情况下一个用户可以保留60个SLB实例。如果需要创建的SLB数量大于60，请提交工单给SLB产品。

> **说明** 您可以在提交工单时，说明需要修改 slb_quota_instances_num 参数，用于提高您可保有的SLB实例个数。

- CCM会根据Service中定义的端口创建SLB监听。默认情况下一个SLB实例可以添加50个监听，如需添加更多监听，请提交工单给SLB产品。

  > **说明** 您可以在提交工单时，说明需要修改 slb_quota_listeners_num 参数，用于提高每个实例可以保有的监听数量。

- CCM会根据Service的配置将ECS挂载到SLB后端服务器组中。
  - 默认情况下一个ECS实例可挂载的后端服务器组的数量为50个，如果一台ECS需要挂载到更多的后端服务器组中，请提交工单给SLB产品。

    > **说明** 您可以在提交工单时，说明需要修改 slb_quota_backendserver_attached_num 参数，用于提高同一台服务器可以重复添加为SLB后端服务器的次数。

  - 默认情况下一个SLB实例可以挂载200个后端服务器，如果需要挂载更多的后端服务器，请提交工单给SLB产品。

    > **说明** 您可以在提交工单时，说明需要修改 slb_quota_backendservers_num 参数，提高每个SLB实例可以挂载的服务器数量。

关于更多SLB使用限制，请参见使用限制。关于如何查询负载均衡SLB配额，请参见负载均衡SLB配额管理。

步骤一：部署示例应用

以下应用部署通过**kubectl**命令行方式进行。如果您需要通过控制台部署应用，请参见创建无状态工作负载Deployment。

1. 使用以下示例应用的YAML内容，创建名为*my-nginx.yaml*文件。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
 name: my-nginx   #示例应用的名称。
 labels:
  app: nginx
spec:
 replicas: 3    #设置副本数量。
 selector:
  matchLabels:
   app: nginx   #对应服务中Selector的值需要与其一致，才可以通过服务公开此应用。
 template:
  metadata:
   labels:
    app: nginx
  spec:
  # nodeSelector:
  #   env: test-team
   containers:
   - name: nginx
    image: registry.aliyuncs.com/acs/netdia:latest   #替换为您实际的镜像地址，格式为：<image_name:tags>。
    ports:
    - containerPort: 80                  #需要在服务中暴露该端口。
```

2. 执行以下命令，部署示例应用my-nginx。

```
kubectl apply -f my-nginx.yaml
```

3. 执行以下命令，确认示例应用状态正常。

```
kubectl get deployment my-nginx
```

返回结果示例：

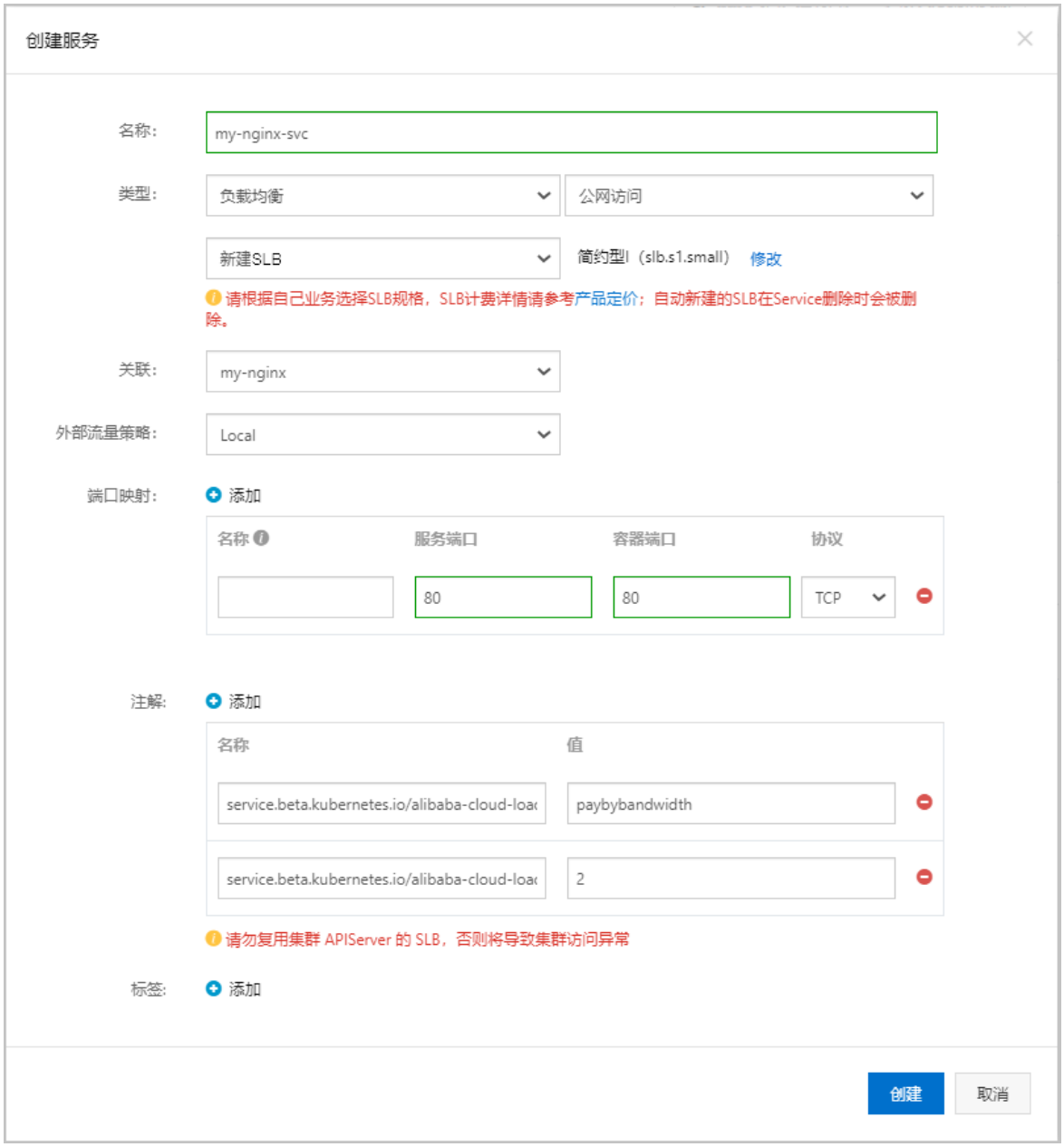
```
NAME    READY  UP-TO-DATE  AVAILABLE  AGE
my-nginx  3/3   3       3       50s
```

步骤二：通过使用自动创建SLB的服务公开应用

您可以通过控制台和**kubectl**两种方式来创建LoadBalancer类型的服务，并通过其公开应用。

控制台方式

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在**服务**页面，单击右上角的**创建**。
6. 在**创建服务**对话框中，设置服务的相关参数。



参数	描述
名称	输入服务的名称，本示例为my-nginx-svc。
类型	选择服务类型，即服务访问的方式。依次选择：**负载均衡** -> **公网访问** -> **新建SLB** -> 单击**修改**选择所需的SLB规格。本示例使用默认规格：简约型I（slb.s1.small）。
关联	选择关联该服务要绑定的后端应用，本示例为my-nginx。若不进行关联部署，则不会创建相关的Endpoints对象，您也可自己进行绑定，请参见services-without-selectors。

参数	描述
外部流量策略	设置外部流量策略，本示例为**Local**。 ○ **Local**：流量只发给本机的Pod。 ○ **Cluster**：流量可以转发到集群中其他节点上的Pod。 **说明** 您的服务类型为**节点端口**或**负载均衡**时，才能设置**外部流量策略**。
端口映射	添加服务端口（对应Service YAML文件中的 `port` ）和容器端口（对应Service YAML文件中的 `targetPort` ），容器端口需要与后端的Pod中暴露的容器端口一致。本示例皆为80。
注解	为该服务添加一个注解（Annotation），配置负载均衡的参数。您可以选择**自定义注解**或**阿里云注解**。本示例中，将该服务的收费方式设置为按带宽收费，带宽峰值设置为2 Mbit/s，从而控制服务的流量。更多注解请参见通过Annotation配置负载均衡。 ○ **类型**：阿里云注解 ○ **名称**：`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-type` 、`service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth` ○ **值**：paybybandwidth、2
标签	为该服务添加一个标签，标识该服务。

7. 单击**创建**。
   在**服务**页面，可以看到新创建的服务。



8. 单击该服务在**外部端点**列的**39.106.XX.XX:80**，访问示例应用。

Kubectl方式

1. 使用以下示例服务的YAML内容，创建名为*my-nginx-svc.yaml*的文件。

   将selector修改为*my-nginx.yaml*示例应用文件中matchLabels的值（本示例为 `app: nginx` ），从而将该服务关联至后端应用。

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

2. 执行以下命令创建名为my-nginx-svc的服务，并通过其公开应用。

```
kubectl apply -f my-nginx-svc.yaml
```

3. 执行以下命令确认LoadBalancer类型的服务创建成功。

```
kubectl get svc my-nginx-svc
```

返回结果示例：

```
NAME       TYPE       CLUSTER-IP   EXTERNAL-IP    PORT(S)     AGE
my-nginx-svc  LoadBalancer  172.21.5.82  39.106.XX.XX   80:30471/TCP  5m
```

4. 执行**curl <YOUR-External-IP>**命令访问示例应用，请将*YOUR-External-IP*替换为上面获取到的 EXTERNAL-IP 地址。

```
curl 39.106.XX.XX
```

返回结果示例：

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

相关文档

- 通过Annotation配置负载均衡
- 通过使用已有SLB的服务公开应用

7.4.5. 通过CCM跨集群部署服务

CCM（Cloud Controller Manager）是ACK的一个系统核心组件，支持在同一个CLB后端挂载集群内节点和集群外ECS，可以解决迁移过程中流量中断的难题。同时CCM还支持将业务流量转发至多个Kubernetes集群内，支撑备份、容灾等需求，保障业务高可用。本文通过两个典型场景介绍如何通过CCM跨集群部署服务。

前提条件

- 已通过传统型负载均衡CLB控制台创建CLB实例，且该实例与Kubernetes集群处于同一地域。具体操作，请参见创建实例。
- CCM需要v2.0.1及以上版本。

背景信息

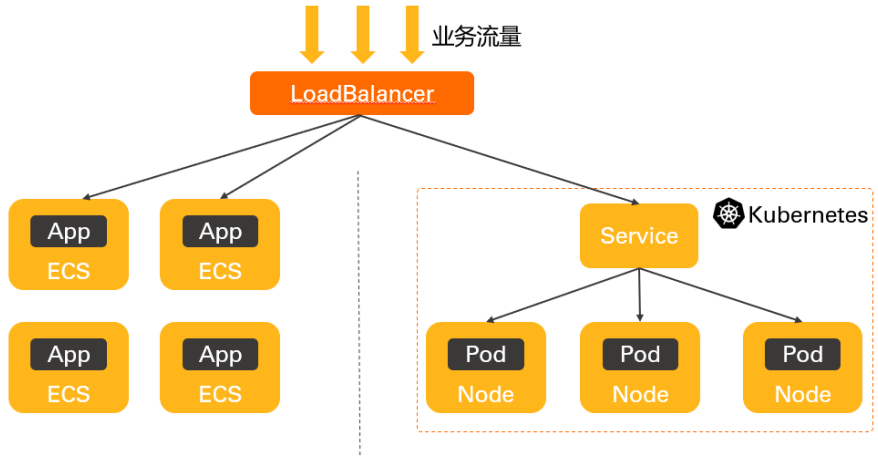
CCM提供Kubernetes与阿里云基础产品的对接能力，例如CLB、VPC等。目前CCM的功能包括管理负载均衡、跨节点通信等。

- 当Service的类型设置为 `Type=LoadBalancer` 时，CCM会为该Service创建或配置阿里云负载均衡CLB，包括CLB、监听、后端服务器组等资源。
- 当Service对应的后端Endpoint或者集群节点发生变化时，CCM会自动更新CLB的后端虚拟服务器组。

更多信息，请参见Cloud Controller Manager。

场景一：将流量转发至集群内节点和集群外ECS

CCM支持在同一个CLB后端挂载集群内节点和集群外ECS，将流量分别转发至集群内部和外部。



1. 使用以下示例服务的YAML内容，创建名为my-nginx-svc.yaml的文件。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: ${YOUR_LB_ID}        //${YOUR_LB_ID}替换为您通过负载均衡管理控制台创建的CLB实例ID。
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners: "false" //设置为false，CCM不会为CLB管理任何监听配置。
 labels:
  app: nignx
 name: my-nginx-svc
 namespace: default
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:  //将selector修改为my-nginx.yaml示例应用文件中matchLabels的值（即：app: nginx），从而将该服务关联至后端应用。
  app: nginx
 type: LoadBalancer
```

> **说明**
>
> - CCM会使用该CLB作为Service的负载均衡，并根据其他注解配置CLB，自动为CLB创建多个虚拟服务器组。
> - CCM还提供了许多阿里云注解，支持丰富的负载均衡能力。更多信息，请参见通过Annotation配置负载均衡。

2. 执行以下命令创建名为my-nginx-svc服务。

```
kubectl apply -f my-nginx-svc.yaml
```

3. 登录传统型负载均衡CLB控制台，为CLB实例创建监听并关联CCM自动创建的虚拟服务器组。

   具体操作，请参见监听概述。

> **注意** CLB的监听端口需要与Service Port一致。

4. 登录传统型负载均衡CLB控制台，在CCM自动创建的虚拟服务器组中添加集群外ECS并设置权重。

   具体操作，请参见创建虚拟服务器组。

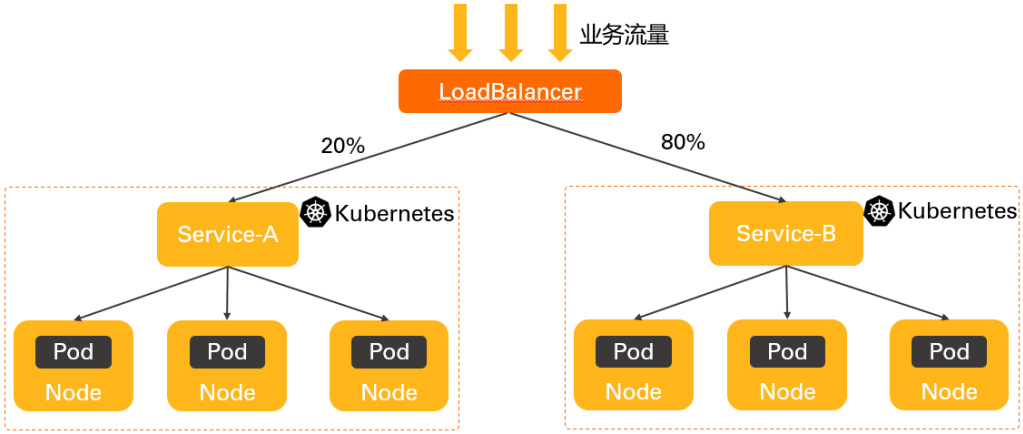
   预期结果

   配置完成后，在传统型负载均衡CLB控制台的**编辑虚拟服务组**页面，既可以看到集群内的节点（图示中①），也可以看到集群外的ECS（图示中②）。集群内应用进行扩缩容时，集群外的ECS节点不受影响。



场景二：将流量转发至多个Kubernetes集群内

通常企业会采取多种措施以保障应用的高可用性，例如创建多个集群进行备份容灾。这要求业务流量可以通过一个CLB接入多个Kubernetes集群，并且为Kubernetes集群设置不同的权重，示意图如下。



> **注意**
> - 不能跨CLB复用虚拟服务器组。
> - 一个虚拟服务器组只能与一个端口关联。

1. 登录传统型负载均衡CLB控制台，创建服务器为空的虚拟服务器组。

   具体操作，请参见创建虚拟服务器组。

2. 登录传统型负载均衡CLB控制台，为CLB实例创建监听并关联CCM自动创建的虚拟服务器组。

   具体操作，请参见监听概述。

   > **注意** CLB的监听端口需要与Service Port一致。

3. 使用以下示例应用的YAML内容，在集群A中创建名为service-a.yaml的文件。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: "${YOUR_LB_ID}"            //${YOUR_LB_ID}替换为您通过负载均衡管理控制台创建的CLB实例ID。
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vgroup-port: "${YOUR_VGROUP_ID}:80"  //${YOUR_VGROUP_ID}替换为虚拟服务器组ID。
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-weight: "20"                  //设置Service流量权重，该参数可选。
 name: service-a    //service-a文件。
 namespace: default
spec:
 ports:
 - name: http
   port: 80
   protocol: TCP
   targetPort: 80
 selector:
   app: nginx
 sessionAffinity: None
 type: LoadBalancer
```

> **说明** 如果 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vgroup-port 有多个端口及虚拟服务器组的组合，可以通过半角逗号（,）分隔。例如"${YOUR_VGROUP_ID_1}:80,${YOUR_VGROUP_ID_2}:443"。

4. 执行以下命令创建名为service-a服务。

```
kubectl apply -f service-a.yaml
```

5. 使用以下示例应用的YAML内容，在集群B中创建名为service-b.yaml的文件。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: "${YOUR_LB_ID}"            //${YOUR_LB_ID}替换为您通过负载均衡管理控制台创建的CLB实例ID。
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vgroup-port: "${YOUR_VGROUP_ID}:80"  //${YOUR_VGROUP_ID}替换为虚拟服务器组ID。
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-weight: "80"                  //设置Service流量权重，该参数可选。
 name: service-b    //service-b文件。
 namespace: default
spec:
 ports:
 - name: http
   port: 80
   protocol: TCP
   targetPort: 80
 selector:
   app: nginx
 sessionAffinity: None
 type: LoadBalancer
```

> **说明** 如果 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vgroup-port 有多个端口及虚拟服务器组的组合，可以通过半角逗号（,）分隔。例如"${YOUR_VGROUP_ID_1}:80,${YOUR_VGROUP_ID_2}:443"。

6. 执行以下命令创建名为service-b服务。

```
kubectl apply -f service-b.yaml
```

预期结果

配置完成后，在传统型负载均衡CLB控制台的**编辑虚拟服务组**页面，既可以看到集群A的节点，也可以看到集群B的节点。集群节点的权重按照注解 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-weight 自动设置。集群内应用进行扩缩容时，CLB后端虚拟服务器组会自动更新。

7.4.6. 管理服务

在Kubernetes中，Pod虽然拥有独立的IP，但Pod会被快速地创建和删除，因此不能通过Pod直接对外提供服务。通过服务（Service）能够解耦前端和后端的关联，从而实现松耦合的微服务设计。本文介绍如何通过控制台和Kubectl方式创建、更新和删除服务。

控制台操作指导

创建服务

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 单击页面右上角的**创建**。
6. 在**创建服务**对话框中，设置相关参数。

参数	描述
名称	设置服务的名称。

参数	描述
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Service类型。 **说明** 您的服务类型为**虚拟集群IP**时，才能设置**实例间发现服务（Headless Service）**。您可以使用无头Service与其他服务发现机制进行接口，而不必与Kubernetes的实现捆绑在一起。 ○ 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），可选择公网访问或私网访问。阿里云负载均衡服务可以路由到NodePort服务和ClusterIP服务。 ■ 新建SLB：您可以通过单击**修改**，修改SLB规格。 ■ 使用已有SLB： 您可以在已有的列表中选择SLB规格。 **说明** 负载均衡类型支持新建SLB和使用已有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	选择服务要绑定的后端应用。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，请参见services-without-selectors。
外部流量策略	设置外部流量策略。 ○ **Local**：流量只发给本机的Pod。 ○ **Cluster**：流量可以转发到集群中其他节点上的Pod。 **说明** 您的服务类型为**节点端口**或**负载均衡**时，才能设置**外部流量策略**。
端口映射	添加服务端口（对应Service YAML文件中的`port`）和容器端口（对应Service YAML文件中的`targetPort`），容器端口需要与后端的Pod中暴露的容器端口一致。

参数	描述
注解	为该服务添加一个注解（Annotation），配置负载均衡的参数。您可以选择**自定义注解**或**阿里云注解**。例如设置 service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth:2 表示将该服务的带宽峰值设置为2Mbit/s，从而控制服务的流量。更多参数请参见通过Annotation配置负载均衡。
标签	您可为该服务添加一个标签，标识该服务。

7. 单击**创建**。

   在服务列表中可以看到新创建的服务。



更新服务

1. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
2. 在对应服务的**操作**列中，单击**更新**。
3. 在**更新服务**对话框中，设置相关参数，然后单击**确定**。

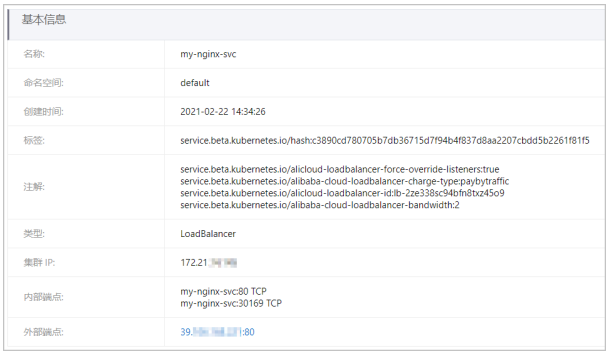


4. 在服务列表中，找到所需的服务，单击右侧的**详情**，查看服务变化的情况。

查看服务

1. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
2. 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地址和端口）。您可以单击该IP地址访问Nginx应用。



Kubectl操作指导

Service YAML示例

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

字段	说明
kind	定义资源对象为Service。
metadata	定义Service的名称、Label和命名空间等基本信息。
metadata.annotations	容器服务ACK支持丰富的负载均衡相关的注解（Annotations）。例如，在上面的YAML示例中，service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type 指定Service的访问方式为 intranet （私网访问）。更多注解，请参见通过Annotation配置负载均衡。
spec.selector	定义Service的选择器。Service会根据Selector和Pod Label的匹配关系，确定Service所要关联暴露的后端Pod。
spec.ports.port	定义Service暴露给ClusterIP的端口，集群内部客户端访问Service的入口，即 clusterIP:port 。
spec.ports.targetPort	定义后端Pod的端口。从 port 进来的流量，经由kube-proxy流入到后端Pod的 targetPort 上，最后进入容器。

字段	说明
spec.type	定义Service的被访问方式。 • LoadBalancer：使用阿里云负载均衡SLB公开服务。当Service没有指定已有SLB时，默认创建的SLB实例类型为公网。您可以通过设置注解service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type为intranet，来创建私网访问的Service以及相应的私网SLB。更多信息，请参见通过使用已有SLB的服务公开应用和通过使用自动创建SLB的服务公开应用。 • ClusterIP：在集群内部公开服务，可用于集群内部访问。 • NodePort：使用节点的端口映射到后端Service，集群外可以通过节点IP:NodePort访问。 • ExternalName：将服务映射到DNS。

创建Service

1. 参考以上YAML示例，创建Service YAML文件。

   下文以创建名为*my-nginx-svc.yaml*的Service YAML文件为例进行说明。

2. 通过Kubectl或CloudShell方式连接集群。详细操作，请参见通过kubectl工具连接集群和在CloudShell上通过kubectl管理Kubernetes集群。

3. 执行以下命令，创建Service。

   ```
   kubectl apply -f my-nginx-svc.yaml
   ```

4. 执行以下命令，验证服务是否创建成功。

   ```
   kubectl get svc my-nginx-svc
   ```

   返回结果示例：

   ```
   NAME        TYPE        CLUSTER-IP    EXTERNAL-IP     PORT(S)     AGE
   my-nginx-svc  LoadBalancer  172.21.XX.XX  192.168.XX.XX   80:31599/TCP  5m
   ```

更新Service

- 方式1：执行以下命令，更新Service。

  ```
  kubectl edit service my-nginx-svc
  ```

- 方式2：手动删除旧的Service，修改YAML文件后，重新创建Service。

  ```
  kubectl apply -f my-nginx-svc.yaml
  ```

查看Service

执行以下命令，查看Service。

```
kubectl get service my-nginx-svc
```

返回结果示例：

```
NAME        TYPE        CLUSTER-IP    EXTERNAL-IP     PORT(S)     AGE
my-nginx-svc  LoadBalancer  172.21.XX.XX  192.168.XX.XX   80:31599/TCP  5m
```

删除Service

执行以下命令，删除Service。

```
kubectl delete service my-nginx-svc
```

7.4.7. Service FAQ

本文汇总了使用容器服务ACK Service时的常见问题。

SLB相关

- 为什么看不到Service与LoadBalancer同步过程的事件Event信息？
- SLB创建一直处于Pending状态如何处理？
- SLB虚拟服务器组未更新如何处理？
- Service注解不生效如何处理？
- 为何SLB的配置被修改？
- 为什么集群内无法访问SLB IP？
- 误删除SLB怎么办？
- 删除Service是否会删除SLB？
- 旧版本CCM如何支持SLB重命名？
- Local模式下如何自动设置Node权重？
- ACK集群中SLB实例的具体用途

CCM升级相关

CCM升级失败如何处理？

使用已有SLB常见问题

- 为什么复用已有SLB没有生效？
- 为什么复用已有SLB时没有配置监听？

其他

Kubernetes的Service如何进行会话保持？

为什么看不到Service与LoadBalancer同步过程的事件Event信息？

如果执行命令 `kubectl -n {your-namespace} describe svc {your-svc-name}` 后看不到Event信息的话，请确认您的CCM组件版本是否大于或等于**v1.9.3.276-g372aa98-aliyun**。低于**v1.9.3.276-g372aa98-aliyun**版本的CCM无法看到Service与LoadBalancer同步过程的事件信息。关于如何查看及升级CCM版本的操作，请参见手动升级CCM组件。

如果您的CCM组件版本大于或等于**v1.9.3.276-g372aa98-aliyun**，请提交工单。

SLB创建一直处于Pending状态如何处理？

1. 执行命令 `kubectl -n {your-namespace} describe svc {your-svc-name}` 查看事件信息。
2. 处理事件中的报错信息。关于事件中不同报错信息的处理方式，请参见报错事件及对应处理方式。

   如果看不到报错信息，请参见为什么看不到Service与LoadBalancer同步过程的事件Event信息？。

SLB虚拟服务器组未更新如何处理？

1. 执行命令 `kubectl -n {your-namespace} describe svc {your-svc-name}` 查看事件信息。
2. 处理事件中的报错信息。关于事件中不同报错信息的处理方式，请参见报错事件及对应处理方式。

   如果看不到报错信息，请参见为什么看不到Service与LoadBalancer同步过程的事件Event信息？。

Service注解不生效如何处理？

1. 按照以下步骤查看报错信息。
   i. 执行命令 `kubectl -n {your-namespace} describe svc {your-svc-name}` 查看事件信息。
   ii. 处理事件中的报错信息。关于事件中不同报错信息的处理方式，请参见报错事件及对应处理方式。
2. 如果没有报错信息，分以下三种场景解决：
   - 确认您的CCM版本是否符合对应注解的版本要求。关于注解和CCM的版本对应关系，请参见常用注解。
   - 在服务Service列表中，单击目标服务右侧**操作**列下的**查看YAML**，确认服务Service中是否有对应的注解。如果没有对应注解，配置注解信息。
     关于如何配置注解，请参见通过Annotation配置负载均衡。
     关于如何查看服务列表，请参见管理服务。
   - 确认注解信息是否有误。

为何SLB的配置被修改？

CCM使用声明式API，会在一定条件下自动根据Service的配置更新SLB配置。您自行在SLB控制台上修改的配置均存在被覆盖的风险。建议您通过注解的方式配置SLB。关于注解配置方式的更多信息，请参见通过Annotation配置负载均衡。

> **注意** 请勿在SLB控制台上手动修改Kubernetes创建并维护的SLB的任何配置，否则有配置丢失的风险，造成Service不可访问。

为什么复用已有SLB没有生效？

- 排查CCM版本，低于v1.9.3.105-gfd4e547-aliyun的CCM版本不支持复用。关于如何查看及升级CCM版本的操作，请参见手动升级CCM组件。
- 确认复用的SLB是否为集群创建的，不支持复用集群创建的SLB。
- 确认SLB是否为API Server的SLB，不支持复用API Server的SLB。
- 如果是私网SLB，确认SLB是否和集群在同一个VPC中，不支持跨VPC复用SLB。

为什么复用已有SLB时没有配置监听？

确认是否在注解中配置了 `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners` 为 `true` 。如果没有配置，则不会自动创建监听。

> **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。

CCM升级失败如何处理？

关于Cloud Controller Manager（CCM）组件升级失败的解决方案，请参见Cloud Controller Manager（CCM）组件升级检查失败。

为什么集群内无法访问SLB IP？

关于为何集群内无法访问SLB IP地址的解决方法，请参见Kubernetes集群中访问LoadBalancer暴露出去的SLB地址不通。

删除Service是否会删除SLB？

如果是复用已有SLB（Service中有 `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: {your-slb-id}` 注解），删除Service时不会删除SLB；否则删除Service时会删除对应的SLB。

如果您更改Service类型（例如从LoadBalancer改为NodePort）也会删除对应的SLB。

误删除SLB怎么办？

- **场景一：误删除API Server SLB如何处理？**
  无法恢复，您需要重建集群。具体操作，请参见创建ACK Pro版集群。
- **场景二：误删除Ingress SLB如何处理？**
  按照以下步骤重建SLB。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 选择**网络 > 服务**。
  v. 在**服务**页面顶部的**命名空间**下拉列表中，单击**kube-system**，然后在服务列表中，找到**nginx-ingress-lb**，然后单击**操作**列下的**查看YAML**。
  如果没有**nginx-ingress-lb**，使用以下模板创建名为**nginx-ingress-lb**的Service。

```
apiVersion: v1
kind: Service
metadata:
 labels:
  app: nginx-ingress-lb
 name: nginx-ingress-lb
 namespace: kube-system
spec:
 externalTrafficPolicy: Local
 ports:
 - name: http
  port: 80
  protocol: TCP
  targetPort: 80
 - name: https
  port: 443
  protocol: TCP
  targetPort: 443
 selector:
  app: ingress-nginx
 type: LoadBalancer
```

  vi. 在**编辑YAML**对话框中，移除**status**内容，然后单击**更新**以让CCM重建SLB。
- **场景三：误删除业务相关的SLB如何处理？**
  - 如果不需要SLB对应的Service，删除Service。
  - 如果对应的Service还需要使用，解决方法如下。
    a. 登录容器服务管理控制台。
    b. 在控制台左侧导航栏中，单击**集群**。
    c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
    d. 在集群管理页面左侧导航栏选择**网络 > 服务**。
    e. 在**服务**页面顶部的**命名空间**下拉框中，单击**所有命名空间**，然后在服务列表中，找到业务Service，然后单击**操作**列下的**查看YAML**。
    f. 在**编辑YAML**对话框中，移除**status**内容，然后单击**更新**以让CCM重建SLB。

旧版本CCM如何支持SLB重命名？

Cloud Controller Manager组件V1.9.3.10后续版本创建的SLB支持自动打TAG从而可以重命名，而V1.9.3.10及之前的版本，您需要手动给该SLB打上一个特定的TAG从而支持SLB重命名。

> **说明**
> - 只有Cloud Controller Manager组件V1.9.3.10及之前版本创建的SLB才需要手动打TAG的方式来支持重命名。
> - Service类型为LoadBalancer。

1. 登录到Kubernetes集群Master节点，请参见通过kubectl工具连接集群。
2. 执行 `kubectl get svc -n ${namespace} ${service}` 命令，查看该Service类型及IP。

```
# kubectl get svc -n ${namespace} ${service}
nginx-local   LoadBalancer   172.19.[redacted]   47.111.[redacted]   8900:31598/TCP   33d
```

> **说明** 您需要将*namespace*与*service*替换为所选集群的命名空间及服务名称。

3. 执行以下命令，生成该SLB所需要的Tag。

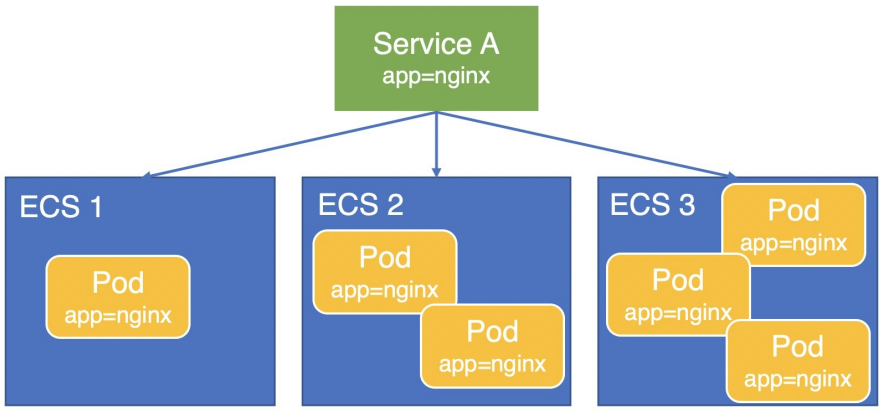
```
kubectl get svc -n ${namespace} ${service} -o jsonpath="{.metadata.uid}"|awk -F "-" '{print "kubernetes.do.not.delete: "substr("a"$1$2$3$4$5,1,32)}'
```

```
# kubectl get svc -n ${namespace} ${service} -o jsonpath="{.metadata.uid}"|awk -F "-" '{print "ku
kubernetes.do.not.delete: [redacted]
```

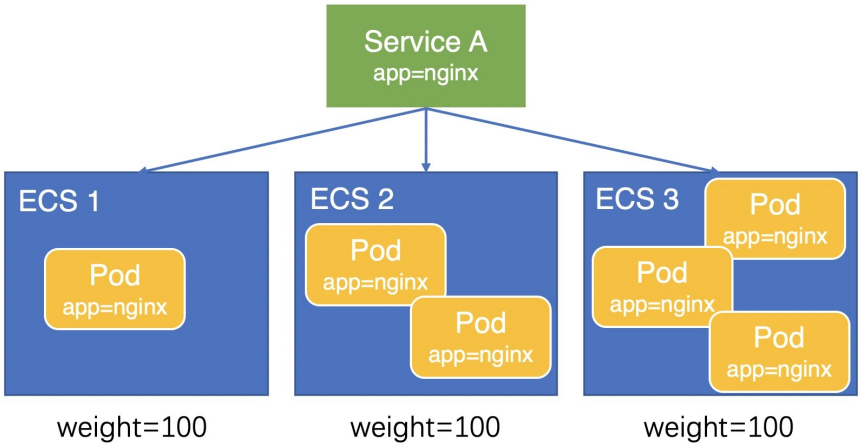
4. 登录负载均衡控制台根据步骤2中所获取的IP，在其所在的地域搜索到该SLB。
5. 根据步骤3生成的Key值和Value值（分别对应上图的1和2），为该SLB打上一个Tag。具体操作，请参见添加标签。

Local模式下如何自动设置Node权重？

本文以业务Pod（app=nginx）部署在三台ECS上，通过Service A对外提供服务为例，说明在Local模式下Node权重的计算方式。



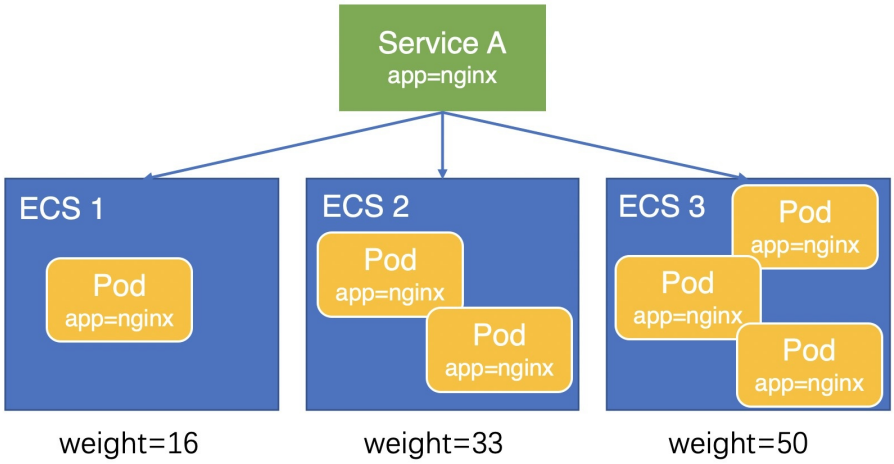
- **V1.9.3.164-g2105d2e-aliyun之前版本**
  如图2所示，在V1.9.3.164-g2105d2e-aliyun之前版本，Local模式的Service其所有后端权重均为100，即所有流量平均分配到这三台ECS上，造成ECS 1上Pod负载较重而ECS 3上的Pod负载较轻，导致Pod负载不均。



- **V1.9.3.164-g2105d2e-aliyun之后及V1.9.3.276-g372aa98-aliyun之前版本**
  如图所示，在V1.9.3.164-g2105d2e-aliyun之后及V1.9.3.276-g372aa98-aliyun之前版本，CCM会根据Node上部署的Pod数量计算Node权重。经计算三台ECS权重分别为16、33、50，因此流量将大致按照1：2：3的比例分配给三台ECS，Pod负载更加均衡。
  计算公式如下所示：

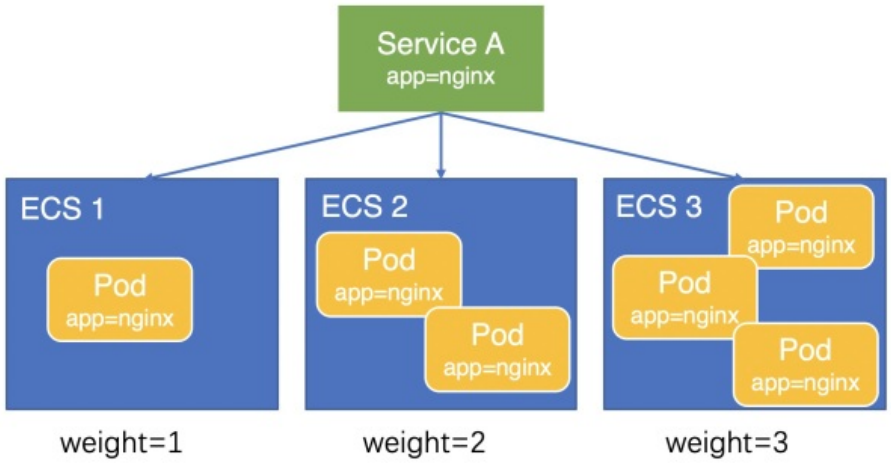
$$\text{weight}=\text{node\_pod\_num} \div \text{all\_pod\_num} \times 100$$

$$\text{node\_weight}=\begin{cases} int(weight) & weight > 1 \\ 1 & weight \leq 1 \end{cases}$$



- **V1.9.3.276-g372aa98-aliyun及之后版本**
  图3的权重计算方式因为计算精度问题，Pod间还会存在轻微的负载不均。在V1.9.3.276-g372aa98-aliyun及之后版本，CCM将Node上部署的Pod数量设置为Node权重，如图4所示，三台ECS的权重分别为1、2、3，流量会按照1：2：3的比例分配给三台ECS，Pod负载比图3的方式更加均衡。
  计算公式如下所示：

$$\text{node\_weight}=\begin{cases} \text{node\_pod\_num} & \text{node\_pod\_num} < 100 \\ 100 & \text{node\_pod\_num} \geq 100 \end{cases}$$



Service报错信息及对应处理方式

不同报错信息的解决方法如下表。

报错信息	说明及解决方法
The backend server number has reached to the quota limit of this load balancers	指SLB虚拟服务组配额不足。 建议您设置SLB外部流量策略为Local模式（设置 externalTrafficPolicy: Local ），因为Cluster模式会导致后端的配额被迅速消耗。如果您设置SLB的外部流量策略为Cluster模式的话，也可以通过标签来指定使用的虚拟服务器。关于如何在注解中添加标签关联后端的虚拟服务器的操作，请参见通过Annotation配置负载均衡。
The backend server number has reached to the quota limit of this load balancer	SLB后端的虚拟服务器超出额度。多个Service复用一个SLB后端服务器是累加的。建议您创建Service时，新建SLB。
The loadbalancer does not support backend servers of eni type	共享型SLB不支持ENI。如果SLB后端使用的是ENI，您需要选择性能保障型SLB实例。在Service中添加注解 annotation: service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: "slb.s1.small" 。 **注意** 添加注解需要注意是否符合CCM的版本要求。关于注解和CCM的版本对应关系，请参见常用注解。
alicloud: not able to find loadbalancer named [%s] in openapi, but it's defined in service.loaderbalancer.ingress, this may happen when you removed loadbalancerid annotation	无法根据Service关联SLB。 • 确认该SLB是否还存在，如果不存在且Service无需再使用，则删除对应的Service即可。 • 如果SLB存在，执行以下步骤。 i. 确认是否是手动在SLB控制台创建的SLB，如果是，则添加复用已有SLB的注解。更多信息，请参见通过Annotation配置负载均衡。 ii. 如果是在ACK中自动创建的SLB，确认该SLB是否有标签kubernetes.do.not.delete。如果没有该标签，请参见旧版本CCM如何支持SLB重命名？。
ORDER.ARREARAGE Message: The account is arrearage.	服务欠费。

报错信息	说明及解决方法
Status Code: 400 Code: Throttlingxxx	调用太多，触发OpenAPI限流。
Status Code: 400 Code: RspoolVipExist Message: there are vips associating with this vServer group.	无法删除虚拟服务器组有关联的监听。 解决方法： 1. 确认Service中的注解是否带有SLB实例的ID（例如 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: {your-slb-id} ）。 如果注解带有SLB实例ID，说明是复用的SLB。 2. 在SLB控制台中删除Service中port对应的监听。关于如何删除SLB监听，请参见管理监听转发规则。
Status Code: 400 Code: NetworkConflict	复用内网SLB时，该SLB和集群不在同一个VPC内。请确保您的SLB和集群在同一个VPC内。
Status Code: 400 Code: VSwitchAvailableIpNotExist Message: The specified VSwitch has no available ip.	虚拟交换机不足，通过 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vswitch-id: "${YOUR_VSWITCH_ID}" 指定同VPC下另一个虚拟交换机。
PAY.INSUFFICIENT_BALANCE Message: Your account does not have enough balance.	账号余额少于100元。
The specified Port must be between 1 and 65535.	ENI模式不支持String类型的 targetPort ，将Service YAML中的 targetPort 字段改为INT类型。
Status Code: 400 Code: ShareSlbHaltSales Message: The share instance has been discontinued.	共享型SLB不再售卖了，而低版本CCM默认创建共享型SLB。 解决方案：升级CCM。

相关文档

- Service的负载均衡配置注意事项
- 通过Annotation配置负载均衡
- Cloud Controller Manager

7.5. Ingress管理

7.5.1. Ingress概述

在Kubernetes集群中，Ingress对集群服务（Service）中外部可访问的API对象进行管理，提供七层负载均衡能力。本文介绍Ingress基本概念、工作原理和使用说明。

Ingress基本概念

在Kubernetes集群中，Ingress作为集群内服务对外暴露的访问接入点，其几乎承载着集群内服务访问的所有流量。Ingress是Kubernetes中的一个资源对象，用来管理集群外部访问集群内部服务的方式。您可以通过Ingress资源来配置不同的转发规则，从而达到根据不同的规则设置访问集群内不同的Service后端Pod。

Ingress资源仅支持配置HTTP流量的规则，无法配置一些高级特性，例如负载均衡的算法、Sessions Affinity等，这些高级特性都需要在Ingress Controller中进行配置。

Nginx Ingress Controller

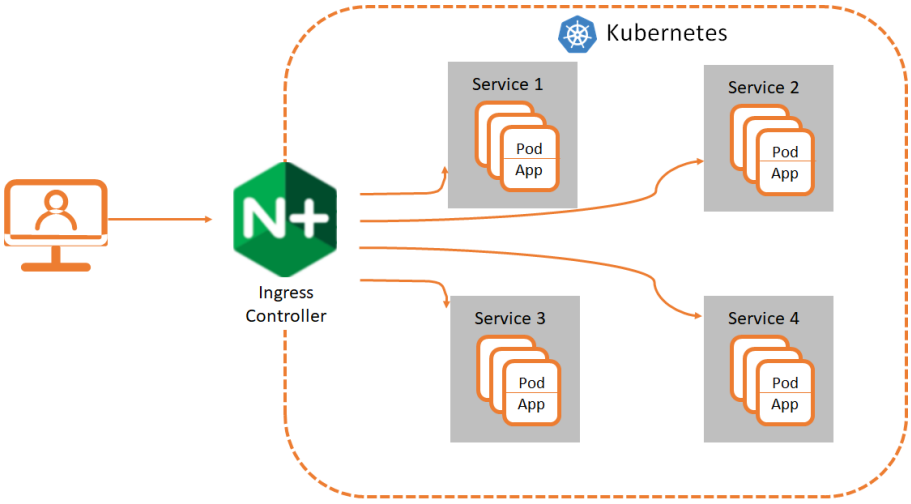
当前Kubernetes官方维护的是Nginx Ingress Controller，ACK基于社区版的Nginx Ingress Controller进行了优化。当在创建ACK集群时，您选择安装的Nginx Ingress Controller组件即为ACK定制版的Nginx Ingress Controller组件。

工作原理

为了使得Ingress资源正常工作，集群中必须要有个Ingress Controller来解析Ingress的转发规则。Ingress Controller收到请求，匹配Ingress转发规则转发到后端Service，而Service转发到Pod，最终由Pod处理请求。Kubernetes中Service、Ingress与Ingress Controller有着以下关系：

- Service是后端真实服务的抽象，一个Service可以代表多个相同的后端服务。
- Ingress是反向代理规则，用来规定HTTP/HTTPS请求应该被转发到哪个Service上。例如根据请求中不同的Host和URL路径，让请求落到不同的Service上。
- Ingress Controller是一个反向代理程序，负责解析Ingress的反向代理规则。如果Ingress有增删改的变动，Ingress Controller会及时更新自己相应的转发规则，当Ingress Controller收到请求后就会根据这些规则将请求转发到对应的Service。

Ingress Controller通过API Server获取Ingress资源的变化，动态地生成Load Balancer（例如Nginx）所需的配置文件（例如*nginx.conf*），然后重新加载Load Balancer（例如执行 `nginx -s load` 重新加载Nginx）来生成新的路由转发规则。



Ingress Controller可通过配置LoadBalancer类型的Service来创建SLB，因此可以从外部通过SLB访问到Kubernetes集群的内部服务。根据Ingress配置的不同规则来访问不同的服务。

相关文档

- Ingress高级用法
- 创建Ingress路由
- Ingress访问日志分析与监控
- 通过Ingress Controller来实现应用的流量复制

ALB Ingress Controller

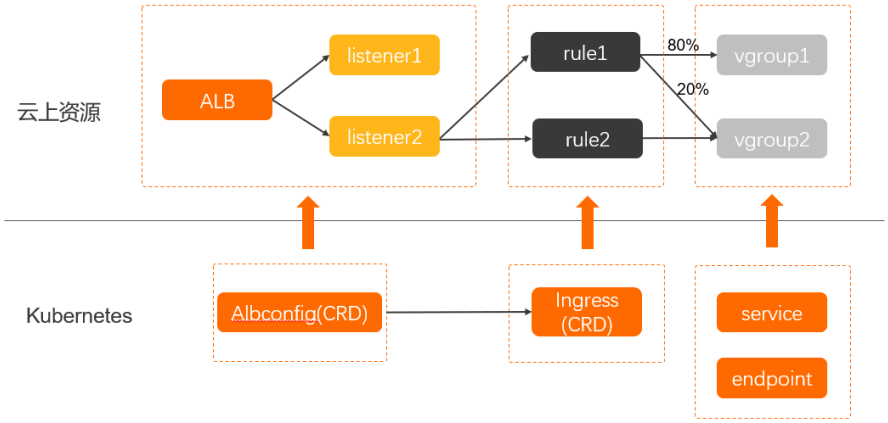
ALB Ingress基于阿里云应用型负载均衡ALB（Application Load Balancer）之上提供更为强大的Ingress流量管理方式，兼容Nginx Ingress，具备处理复杂业务路由和证书自动发现的能力，支持HTTP、HTTPS和QUIC协议，完全满足在云原生应用场景下对超强弹性和大规模七层流量处理能力的需求。

> 注意
>
> - 为Ingress服务的ALB是由Controller完全托管的，您不能自行在ALB控制台上面进行配置，否则可能造成Ingress服务的异常。
> - ALB Ingress目前处于公测阶段，若您无法使用该功能，请提交工单申请白名单。

工作原理

ALB Ingress Controller通过API Server获取Ingress资源的变化，动态地生成Albconfig，然后依次创建ALB实例、监听、路由转发规则以及后端服务器组。Kubernetes中Service、Ingress与Albconfig有着以下关系：

- Service是后端真实服务的抽象，一个Service可以代表多个相同的后端服务。
- Ingress是反向代理规则，用来规定HTTP/HTTPS请求应该被转发到哪个Service上。例如：根据请求中不同的Host和URL路径，让请求转发到不同的Service上。
- Albconfig是在ALB Ingress Controller提供的CRD资源，使用ALBConfig CRD来配置ALB实例和监听。一个Albconfig对应一个ALB实例。



相关文档

- 通过ALB Ingress访问服务
- ALB Ingress服务高级用法
- 为ACK专有版集群中的ALB Ingress Controller授权

7.5.2. 创建Ingress路由

Ingress是Kubernetes中的一个资源对象，用来管理集群外部访问集群内部服务的方式。您可以通过Ingress资源来配置不同的转发规则，从而根据转发规则访问集群内Pod。本文介绍如何通过控制台和Kubectl方式创建、查看、更新和删除Ingress。

前提条件

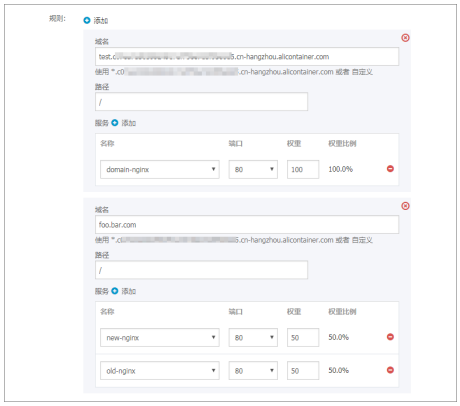
已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

控制台操作指导

创建Ingress

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
5. 在**路由**页面单击**创建**，在**创建**对话框配置路由名称，本例中为nginx-ingress。
6. 配置路由规则。

   路由规则是指授权入站到达集群服务的规则，支持HTTP或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。
   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.XX.XX 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new-nginx和old-nginx。
    - 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%。

7. 配置TLS。

   选中**开启**TLS，配置安全的路由服务。具体请参见Ingress支持。

   - 您可选择使用已有密钥。
     a. 登录Master节点，执行以下命令创建*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，执行以下命令创建*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。



8. 配置灰度发布策略。

   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。

   i. 基于Request Header的流量切分。
   ii. 基于Cookie的流量切分。
   iii. 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**：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。

> **说明** 灰度发布策略只支持2个Service配置。

9. 配置注解。

   单击**添加**，在**类型**下拉框中，您可以：

   - 选择**自定义注解**：输入注解名称和值，即Ingress的Annotation键值对，Ingress的注解请参见Annotations。
   - 选择**Ingress-Nginx**：根据名称选择或搜索要配置的注解。
     您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*/*。

     > **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。

10. 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。



11. 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。

查看Ingress

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
5. 在**路由**页面单击目标路由**操作**列的**详情**。

   在路由详情页面查看路由详细信息。

更新Ingress

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
5. 在**路由**页面单击目标路由**操作**列的**变更**。
6. 在**更新**对话框中修改路由参数，然后单击**更新**。

删除Ingress

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
5. 在**路由**页面选择目标路由**操作**列的**更多 > 删除**。
6. 在**提示**对话框中单击**确定**。

Kubectl操作指导

创建Ingress

1. 创建Deployment和Service。

   在创建Ingress资源之前，必须创建外部访问Kubernetes集群中的服务。

   i. 使用以下内容，创建*test-deployment-service.yaml*。

   以下YAML文件中包含了一个名为test-web1的Deployment和一个名为web1-service的Service。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: test-web1
 labels:
  app: test-web1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: test-web1
 template:
  metadata:
   labels:
    app: test-web1
  spec:
   containers:
   - name: test-web1
    imagePullPolicy: IfNotPresent
    image: registry.cn-hangzhou.aliyuncs.com/yilong/ingress-test:web1
    ports:
    - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
 name: web1-service
spec:
 type: ClusterIP
 selector:
  app: test-web1
 ports:
  - port: 8080
   targetPort: 8080
```

   ii. 执行以下命令，创建Deployment和Service。

```
kubectl apply -f test-deployment-service.yaml
```

2. 创建Ingress。

i. 使用以下内容，创建*test-ingress.yaml*。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: test-ingress
 namespace: default
spec:
 rules:
 - host: test-ingress.com
  http:
   paths:
   - path: /foo
    backend:
     service:
      name: web1-service
      port:
       number: 8080
    pathType: ImplementationSpecific
   - path: /bar
    backend:
     service:
      name: web1-service
      port:
       number: 8080
    pathType: ImplementationSpecific
```

- `name`：Ingress的名称，本例为test-ingress。
- `host`：指定服务访问域名。
- `path`：指定访问的URL路径。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`：由服务名称和服务端口组成。
  - 服务名称：Ingress转发的`backend`服务名称。
  - 服务端口：服务暴露的端口。

ii. 执行以下命令，创建Ingress。

```
kubectl apply -f test-ingress.yaml
```

查看Ingress

执行以下命令，查看Ingress。

```
kubectl get ingress
```

更新Ingress

执行以下命令，更新Ingress。

```
kubectl edit ingress <ingress名称>
```

删除Ingress

执行以下命令，删除Ingress。

```
kubectl delete ingress <ingress名称>
```

7.5.3. Nginx Ingress管理

7.5.3.1. Ingress高级用法

在Kubernetes集群中，Ingress对集群服务（Service）中外部可访问的API对象进行管理，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、Rewrite配置、HTTPS服务、以及灰度发布功能等。本文介绍如何配置安全的路由服务、HTTPS双向认证、域名支持正则化及泛化，申请免费的HTTPS证书等功能。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 集群中的Ingress Controller运行正常。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。
- 已创建示例Deployment和Service。具体操作，请参见Kubectl操作指导。

配置说明

针对Nginx Ingress Controller，阿里云容器服务团队采用与社区完全兼容的配置方式。关于所有的配置说明，请参见NGINX Configuration。

目前其主要支持三种配置方式：

- 基于Annotation的方式：在每个Ingress YAML的Annotation里配置，只对本Ingress生效。更多信息，请参见Annotations。
- 基于ConfigMap的方式：通过kube-system/nginx-configuration configmap的配置，是一个全局的配置，对所有的Ingress生效。更多信息，请参见ConfigMaps。
- 自定义NGINX Template模板的方式：对Ingress Controller内部的NGINX Template有特殊配置要求，且当前通过Annotation和ConfigMap方式都无法满足诉求的情况下采用该方式。更多信息，请参见Custom NGINX template。

配置URL重定向的路由服务

通过以下命令创建一个简单的Ingress，所有对*/svc*路径的访问都会重新定向到后端服务能够识别的/路径上面。

1. 部署以下模板，创建Ingress。

```
cat <<-EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: rewrite-test-ingress
 namespace: default
 annotations:
  # URL重定向
  nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
 - host: rewrite-test-ingress.com
  http:
   paths:
   - path: /svc/(.*)
    backend:
     service:
      name: web1-service
      port:
       number: 80
    pathType: ImplementationSpecific
EOF
```

2. 执行以下命令，访问Nginx服务。

   替换IP_ADDRESS为Ingress对应的IP，可通过 `kubectl get ing` 获取。

   ```
   curl -k -H "Host: rewrite-test-ingress.com"  http://<IP_ADDRESS>/svc/foo
   ```

   预期输出：

   ```
   web1: /foo
   ```

Rewrite配置

使用 `inginx.ingress.kubernetes.io/rewrite-target` 注解支持基本的Rewrite配置，对于一些复杂高级的Rewrite需求，可以通过如下注解来实现：

- `nginx.ingress.kubernetes.io/server-snippet` ：扩展配置到Server章节。
- `nginx.ingress.kubernetes.io/configuration-snippet` ：扩展配置到Location章节。

配置示例：

```
annotations:
   nginx.ingress.kubernetes.io/server-snippet: |
     rewrite ^/v4/(.*)/card/query http://m.maizuo.com/v5/#!/card/query permanent;
   nginx.ingress.kubernetes.io/configuration-snippet: |
     rewrite ^/v6/(.*)/card/query http://m.maizuo.com/v7/#!/card/query permanent;
```

示例配置生成的*nginx.conf*如下所示。

```
## start server foo.bar.com
  server {
    server_name foo.bar.com ;
    listen 80;
    listen [::]:80;
    set $proxy_upstream_name "-";
  ### server-snippet配置。
    rewrite ^/v4/(.*)/card/query http://m.maizuo.com/v5/#!/card/query permanent;
    ...
  ### configuration-snippet配置。
   rewrite ^/v6/(.*)/card/query http://m.maizuo.com/v7/#!/card/query permanent;
   ...
  }
  ## end server foo.bar.com
```

同时， `snippet` 也支持一些全局配置。详细信息，请参见server-snippet。

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** 域名需要与您的Ingress配置保持一致。

   i. 执行以下命令，生成一个证书文件*tls.crt*和一个私钥文件*tls.key*。

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

   ii. 执行以下命令，创建密钥。

   通过该证书和私钥创建一个名为tls-test-ingress的Kubernetes Secret。创建Ingress时需要引用这个Secret。

   ```
   kubectl create secret tls tls-test-ingress --key tls.key --cert tls.crt
   ```

2. 执行以下命令，创建一个安全的Ingress服务。

```
cat <<EOF | kubectl create -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: test-test-ingress
spec:
 #安全路由
 tls:
 - hosts:
  - tls-test-ingress.com
  secretName: tls-test-ingress
 rules:
 - host: tls-test-ingress.com
  http:
   paths:
   - path: /foo
    backend:
     service:
      name: web1-svc
      port:
       number: 80
    pathType: ImplementationSpecific
EOF
```

3. 执行以下命令，查询Ingress信息。

```
kubectl get ingress
```

预期输出：

```
NAME            HOSTS                ADDRESS      PORTS   AGE
test-test-ingress    ****.nginx.ingress.com      101.37.XX.XX   80     11s
```

4. 配置 `hosts` 文件或者设置域名来访问该TLS服务。

   您可以通过 `http://tls-test-ingress.com/foo` 访问到 `web1-svc` 服务。

配置HTTPS双向认证

某些业务场景需要启用HTTPS双向验证，Ingress-Nginx支持该特性，配置步骤参考以下示例。

1. 执行以下命令，创建自签的CA证书。

```
openssl req -x509 -sha256 -newkey rsa:4096 -keyout ca.key -out ca.crt -days 356 -nodes -subj '/CN=Fern Cert Au
thority'
```

预期输出：

```
Generating a 4096 bit RSA private key
.....................................................................................++
.....................................................................++
writing new private key to 'ca.key'
```

2. 执行以下命令，创建Server端证书。

i. 执行以下命令，生成Server端证书的请求文件。

```
openssl req -new -newkey rsa:4096 -keyout server.key -out server.csr -nodes -subj '/CN=test.nginx.ingress.com'
```

预期输出：

```
Generating a 4096 bit RSA private key
.....++
.....++
writing new private key to 'server.key'
```

ii. 执行以下命令，使用根证书签发Server端请求文件，生成Server端证书。

```
openssl x509 -req -sha256 -days 365 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
```

预期输出：

```
Signature ok
subject=/CN=test.nginx.ingress.com
Getting CA Private Key
```

3. 执行以下命令，创建Client端证书。

i. 生成Client端证书的请求文件。

```
openssl req -new -newkey rsa:4096 -keyout client.key -out client.csr -nodes -subj '/CN=Fern'
```

预期输出：

```
Generating a 4096 bit RSA private key
.....
....++
.....++
writing new private key to 'client.key'
-----
```

ii. 执行以下命令，使用根证书签发Client端请求文件，生成Client端证书。

```
openssl x509 -req -sha256 -days 365 -in client.csr -CA ca.crt -CAkey ca.key -set_serial 02 -out client.crt
```

预期输出：

```
Signature ok
subject=/CN=Fern
Getting CA Private Key
```

4. 执行以下命令，检查创建的证书。

```
ls
```

预期输出：

```
ca.crt  ca.key  client.crt  client.csr  client.key  server.crt  server.csr  server.key
```

5. 执行以下命令，创建CA证书的Secret。

```
kubectl create secret generic ca-secret --from-file=ca.crt=ca.crt
```

预期输出：

```
secret/ca-secret created
```

6. 执行以下命令，创建Server证书的Secret。

```
kubectl create secret generic tls-secret --from-file=tls.crt=server.crt --from-file=tls.key=server.key
```

预期输出：

```
secret/tls-secret created
```

7. 执行以下命令，创建测试用的Ingress用例。

```
cat <<-EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
  nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
  nginx.ingress.kubernetes.io/auth-tls-secret: "default/ca-secret"
  nginx.ingress.kubernetes.io/auth-tls-verify-depth: "1"
  nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream: "true"
 name: nginx-test
 namespace: default
spec:
 rules:
 - host: test.nginx.ingress.com
  http:
   paths:
   - backend:
      service:
       name: http-svc
       port:
        number: 80
     path: /
     pathType: ImplementationSpecific
 tls:
 - hosts:
  - test.nginx.ingrss.com
  secretName: tls-secret
EOF
```

预期输出：

```
ingress.networking.k8s.io/nginx-test configured
```

8. 执行以下命令，查看Ingress的IP地址。

```
kubectl get ing
```

预期输出如下，ADDRESS字段对应的IP地址即为Ingress的IP地址。

```
NAME     HOSTS          ADDRESS     PORTS   AGE
nginx-test  test.nginx.ingress.com  39.102.XX.XX  80, 443  4h42m
```

9. 执行以下命令，更新Hosts文件，替换下面的IP地址为真实获取的Ingress的IP地址。

```
sudo echo "39.102.XX.XX  test.nginx.ingress.com" >> /etc/hosts
```

结果验证：

- 客户端不传证书访问

```
curl --cacert ./ca.crt  https://test.nginx.ingress.com
```

预期输出：

```
<html>
<head><title>400 No required SSL certificate was sent</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<center>No required SSL certificate was sent</center>
<hr><center>nginx/1.19.0</center>
</body>
</html>
```

- 客户端传证书访问

```
curl --cacert ./ca.crt --cert ./client.crt --key ./client.key https://test.nginx.ingress.com
```

预期输出：

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

配置HTTPS服务转发到后端容器为HTTPS协议

当后端业务容器内是HTTPS服务时，可以通过使用注解 `nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"` 来使得Ingress Controller转发到后端容器时仍为HTTPS协议。

Ingress配置示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: backend-https
 annotations:
  # 注意这里：必须指定后端服务为HTTPS服务。
  nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
 tls:
 - hosts:
  - <your-host-name>
  secretName: <your-secret-cert-name>
 rules:
 - host: <your-host-name>
  http:
   paths:
   - path: /
    backend:
     service:
      name: <your-service-name>
      port:
       number: <your-service-port>
    pathType: ImplementationSpecific
```

配置域名支持正则化

在Kubernetes集群中，Ingress资源不支持对域名配置正则表达式，但是可以通过 nginx.ingress.kubernetes.io/server-alias 注解来实现。

1. 创建Ingress，以正则表达式 ~^www\.\d+\.example\.com 为例。

   ```
   cat <<-EOF | kubectl apply -f -
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
    name: ingress-regex
    namespace: default
    annotations:
     nginx.ingress.kubernetes.io/server-alias: '~^www\.\d+\.example\.com$, abc.example.com'
   spec:
    rules:
    - host: foo.bar.com
     http:
      paths:
      - path: /foo
       backend:
        service:
         name: http-svc1
         port:
          number: 80
       pathType: ImplementationSpecific
   EOF
   ```

2. 执行以下命令，查看对应Nginx Ingress Controller的配置。

i. 执行以下命令，查看部署Nginx Ingress Controller服务的Pod。

```
kubectl get pods -n kube-system | grep nginx-ingress-controller
```

预期输出：

```
nginx-ingress-controller-77cd987c4c-cp87g     1/1   Running  0      1h
nginx-ingress-controller-77cd987c4c-xvpt5     1/1   Running  0      1h
```

ii. 执行以下命令，查看对应Nginx Ingress Controller的配置，可以发现生效的配置（Server_Name字段）。

```
kubectl exec -n kube-system nginx-ingress-controller-77cd987c4c-cp87g cat /etc/nginx/nginx.conf | grep -C 3 "regex.ingress.test.com"
```

预期输出：

```
 ## start server foo.bar.com
 server {
--
 server {
  server_name foo.bar.com abc.example.com ~^www\.\d+\.example\.com$ ;
  listen 80  ;
  listen 443  ssl http2 ;
--
--
  }
 }
 ## end server foo.bar.com
```

3. 执行以下命令，获取Ingress对应的IP。

```
kubectl get ing
```

预期输出：

```
NAME      HOSTS     ADDRESS      PORTS   AGE
ingress-regex  foo.bar.com  101.37.XX.XX   80     11s
```

4. 执行以下命令，进行不同规则下的服务访问测试。

配置以下**IP_ADDRESS**为上一步获取的IP地址。

- 执行以下命令，通过 `Host: foo.bar.com` 访问服务。

```
curl -H "Host: foo.bar.com" <IP_ADDRESS>/foo
```

预期输出：

```
/foo
```

- 执行以下命令，通过 `Host: www.123.example.com` 访问服务。

```
curl -H "Host: www.123.example.com" <IP_ADDRESS>/foo
```

预期输出：

```
/foo
```

- 执行以下命令，通过 `Host: www.321.example.com` 访问服务。

```
curl -H "Host: www.321.example.com" <IP_ADDRESS>/foo
```

预期输出：

```
/foo
```

配置域名支持泛化

在Kubernetes集群中，Ingress资源支持对域名配置泛域名，例如，可配置 *.ingress-regex.com 泛域名。

1. 部署以下模板，创建Ingress。

```
cat <<-EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress-regex
 namespace: default
spec:
 rules:
 - host: '*.ingress-regex.com'
   http:
    paths:
    - path: /foo
      backend:
       service:
         name: http-svc1
         port:
          number: 80
     pathType: ImplementationSpecific
EOF
```

2. 执行以下命令，查看对应Nginx Ingress Controller的配置，可以发现生效的配置（Server_Name字段）。

```
kubectl exec -n kube-system <ningx-ingress-pod-name> cat /etc/nginx/nginx.conf | grep -C3 "ingress-regex.com
"
```

> **说明** 替换*ningx-ingress-pod-name*为实际环境的nginx-ingress pod。

预期输出：

```
## start server *.bar.com
 server {
  server_name *.ingress-regex.com ;
  listen 80;
  listen [::]:80;
...
 }
 ## end server *.bar.com
```

3. 执行以下命令，获取Ingress对应的IP。

```
kubectl get ing
```

预期输出：

```
NAME      HOSTS     ADDRESS      PORTS   AGE
ingress-regex  *.bar.com   101.37.XX.XX    80     11s
```

4. 执行以下命令，进行不同规则下的服务访问测试。

   配置以下**IP_ADDRESS**为上一步获取的IP地址。

   - 执行以下命令，通过 Host: abc.ingress-regex.com 访问服务。

```
curl -H "Host: abc.ingress-regex.com" <IP_ADDRESS>/foo
```

预期输出：

```
/foo
```

- 执行以下命令，通过 `Host: 123.ingress-regex.com` 访问服务。

```
curl -H "Host: 123.ingress-regex.com" <IP_ADDRESS>/foo
```

预期输出：

```
/foo
```

- 执行以下命令，通过 `Host: a1b1.ingress-regex.com` 访问服务。

```
curl -H "Host: a1b1.ingress-regex.com" <IP_ADDRESS>/foo
```

预期输出：

```
/foo
```

通过注解实现灰度发布

灰度发布功能可以通过设置注解来实现，为了启用灰度发布功能，需要设置注解 `nginx.ingress.kubernetes.io/canary: "true"` ，通过不同注解可以实现不同的灰度发布功能：

- `nginx.ingress.kubernetes.io/canary-weight` ：设置请求到指定服务的百分比（值为0~100的整数）。
- `nginx.ingress.kubernetes.io/canary-by-header` ：基于Request Header的流量切分，当配置的 `hearder` 值为 `always` 时，请求流量会被分配到灰度服务入口；当 `hearder` 值为 `never` 时，请求流量不会分配到灰度服务；将忽略其他 `hearder` 值，并通过灰度优先级将请求流量分配到其他规则设置的灰度服务。
- `nginx.ingress.kubernetes.io/canary-by-header-value` 和 `nginx.ingress.kubernetes.io/canary-by-header` ：当请求中的 `hearder` 和 `header-value` 与设置的值匹配时，请求流量会被分配到灰度服务入口；将忽略其他 `hearder` 值，并通过灰度优先级将请求流量分配到其他规则设置的灰度服务。
- `nginx.ingress.kubernetes.io/canary-by-cookie` ：基于Cookie的流量切分，当配置的 `cookie` 值为 `always` 时，请求流量将被分配到灰度服务入口；当配置的 `cookie` 值为 `never` 时，请求流量将不会分配到灰度服务入口。

不同注解配置示例如下：

- 基于权重灰度：配置灰度服务的权重为20%。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
   kubernetes.io/ingress.class: nginx
   nginx.ingress.kubernetes.io/canary: "true"
   nginx.ingress.kubernetes.io/canary-weight: "20"
```

- 基于Header灰度：请求Header为 `ack：always` 时将访问灰度服务；请求Header为 `ack：never` 时将不访问灰度服务；其它Header将根据灰度权重将流量分配给灰度服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
   kubernetes.io/ingress.class: nginx
   nginx.ingress.kubernetes.io/canary: "true"
   nginx.ingress.kubernetes.io/canary-weight: "50"
   nginx.ingress.kubernetes.io/canary-by-header: "ack"
```

- 基于Header灰度（自定义header值）：当请求Header为 `ack: alibaba` 时将访问灰度服务；其它Header将根据灰度权重将流量分配给灰度服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/canary: "true"
  nginx.ingress.kubernetes.io/canary-weight: "20"
  nginx.ingress.kubernetes.io/canary-by-header: "ack"
  nginx.ingress.kubernetes.io/canary-by-header-value: "alibaba"
```

- 基于Cookie灰度：当Header不匹配时，请求的Cookie为 `hangzhou_region=always` 时将访问灰度服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/canary: "true"
  nginx.ingress.kubernetes.io/canary-weight: "20"
  nginx.ingress.kubernetes.io/canary-by-header: "ack"
  nginx.ingress.kubernetes.io/canary-by-header-value: "alibaba"
  nginx.ingress.kubernetes.io/canary-by-cookie: "hangzhou_region"
```

> 说明
> - 基于Cookie的灰度不支持设置自定义，只有 `always` 和 `never` 。
> - 灰度优先级顺序：基于Header > 基于Cookie > 基于权重（从高到低）。

使用cert-manager申请免费的HTTPS证书

cert-manager是一个云原生证书管理开源工具，用于在Kubernetes集群中提供HTTPS证书并自动续期。以下示例介绍了如何使用cert-manager申请免费证书并自动续期。

1. 执行以下命令，部署cert-manager。

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.1.0/cert-manager.yaml
```

> **说明** 上述命令中的YAML示例仅适用于在ACK集群中部署cert-manager。关于如何在ASK集群中部署cert-manager，请参见使用cert-manager申请免费的HTTPS证书。

2. 执行以下命令，查看Pod状态。

```
kubectl get pods -n cert-manager
```

预期输出：

```
NAME                       READY  STATUS   RESTARTS  AGE
cert-manager-1      1/1   Running  0      2m11s
cert-manager-cainjector 1/1   Running  0      2m11s
cert-manager-webhook   1/1   Running  0      2m10s
```

3. 执行以下命令，创建ClusterIssuer。

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
 name: letsencrypt-prod-http01
spec:
 acme:
  server: https://acme-v02.api.letsencrypt.org/directory
  email: <your_email_name@gmail.com> #替换为您的邮箱名。
  privateKeySecretRef:
   name: letsencrypt-http01
  solvers:
  - http01:
     ingress:
      class: nginx
EOF
```

4. 执行以下命令，查看ClusterIssuer。

```
kubectl get clusterissuer
```

预期输出：

```
NAME                READY  AGE
letsencrypt-prod-http01    True   17s
```

5. 执行以下命令，创建Ingress资源对象。

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress-tls
 annotations:
  kubernetes.io/ingress.class: "nginx"
  cert-manager.io/cluster-issuer: "letsencrypt-prod-http01"
spec:
 tls:
 - hosts:
  - <your_domain_name>     # 替换为您的域名。
  secretName: ingress-tls
 rules:
 - host: <your_domain_name>   # 替换为您的域名。
  http:
   paths:
   - path: /
     backend:
      service:
       name: <your_service_name> # 替换为您的后端服务名。
       port:
        number: <your_service_port> # 替换为您的服务端口。
    pathType: ImplementationSpecific
EOF
```

> **说明** 替换的域名your_domain_name必须符合以下条件：
> - 域名不能超过64个字符。
> - 不支持泛域名。
> - 在公网以HTTP协议可正常访问。

6. 执行以下命令，查看证书。

```
kubectl get cert
```

预期输出：

```
NAME          READY  SECRET       AGE
ingress-tls   True   ingress-tls  52m
```

> **说明** 如果READY状态不为True，可通过 `kubectl describe cert ingress-tls` 查看证书处理过程。

7. 执行以下命令，查看Secret。

```
kubectl get secret  ingress-tls
```

预期输出：

```
NAME          TYPE                DATA  AGE
ingress-tls   kubernetes.io/tls   2     2m
```

8. 通过Web浏览器输入 `https:[网站域名]` 访问设置的域名。

7.5.3.2. 通过Ingress实现灰度发布和蓝绿发布

当对服务进行版本更新升级时，需要使用到滚动升级、分批暂停发布、蓝绿发布以及灰度发布等发布方式。本文将介绍在ACK集群中如何通过Ingress Controller来实现应用服务的灰度发布。

体验实验室

为了帮助您快速上手本教程，ACK为您提供了可以动手操作的交互式实验环境。如您需要体验，请访问通过Ingress进行灰度发布。

前提条件

您已创建一个ACK集群。具体操作，请参见创建Kubernetes托管版集群。

背景信息

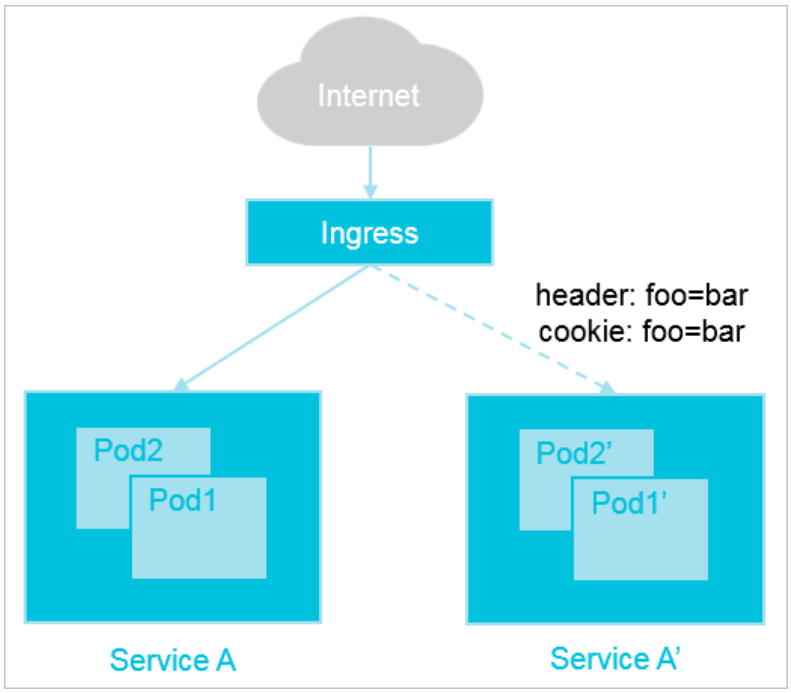
灰度及蓝绿发布是为新版本创建一个与老版本完全一致的生产环境，在不影响老版本的前提下，按照一定的规则把部分流量切换到新版本，当新版本试运行一段时间没有问题后，将用户的全量流量从老版本迁移至新版本。

其中AB测试就是一种灰度发布方式，一部分用户继续使用老版本的服务，将一部分用户的流量切换到新版本，如果新版本运行稳定，则逐步将所有用户迁移到新版本。

应用场景

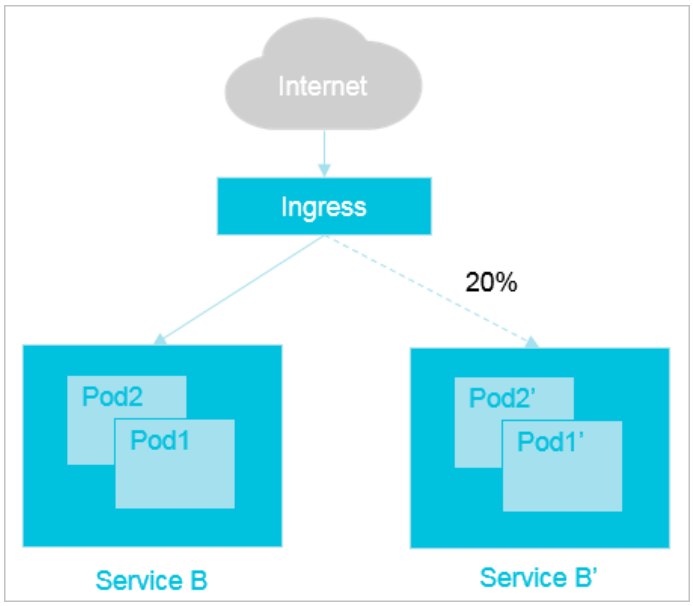
基于客户端请求的流量切分场景

假设当前线上环境，您已经有一套服务Service A对外提供7层服务，此时上线了一些新的特性，需要发布上线一个新的版本Service A'。但又不想直接替换Service A服务，而是希望将请求头中包含 `foo=bar` 或者cookie中包含 `foo=bar` 的客户端请求转发到Service A'服务中。待运行一段时间稳定后，可将所有的流量从Service A切换到Service A'服务中，再平滑地将Service A服务下线。



基于服务权重的流量切分场景

假设当前线上环境，您已经有一套服务Service B对外提供7层服务，此时修复了一些问题，需要发布上线一个新的版本Service B'。但又不想将所有客户端流量切换到新版本Service B'中，而是希望将20%的流量切换到新版本Service B'中。待运行一段时间稳定后，再将所有的流量从Service B切换到Service B'服务中，再平滑地将Service B服务下线。



针对以上多种不同的应用发布需求，阿里云容器服务Ingress Controller支持了多种流量切分方式：

- 基于Request Header的流量切分，适用于灰度发布以及AB测试场景。
- 基于Cookie的流量切分，适用于灰度发布以及AB测试场景。
- 基于Query Param的流量切分，适用于灰度发布以及AB测试场景。

- 基于服务权重的流量切分，适用于蓝绿发布场景。

注解说明

Ingress Controller通过下列Annotation来支持应用服务的灰度发布机制。

- nginx.ingress.kubernetes.io/service-match
  该注解用来配置新版本服务的路由匹配规则。

```
nginx.ingress.kubernetes.io/service-match: |
  <service-name>: <match-rule>
# 参数说明：
# service-name：服务名称，满足match-rule的请求会被路由到该服务中。
# match-rule：路由匹配规则
#
# 路由匹配规则：
# 1. 支持的匹配类型
# - header：基于请求头，支持正则匹配和完整匹配。
# - cookie：基于cookie，支持正则匹配和完整匹配。
# - query：基于请求参数，支持正则匹配和完整匹配。
#
# 2. 匹配方式
# - 正则匹配格式：/{regular expression}/，//表明采用正则方式匹配。
# - 完整匹配格式："{exact expression}"，""表明采用完整方式匹配。
```

  路由匹配规则配置示例：

```
# 请求头中满足foo正则匹配^bar$的请求被转发到新版本服务new-nginx中。
new-nginx: header("foo", /^bar$/)
# 请求头中满足foo完整匹配bar的请求被转发到新版本服务new-nginx中。
new-nginx: header("foo", "bar")
# cookie中满足foo正则匹配^sticky-.+$的请求被转发到新版本服务new-nginx中。
new-nginx: cookie("foo", /^sticky-.+$/)
# query param中满足foo完整匹配bar的请求被转发到新版本服务new-nginx中。
new-nginx: query("foo", "bar")
```

- nginx.ingress.kubernetes.io/service-weight
  该注解用来配置新旧版本服务的流量权重。

```
nginx.ingress.kubernetes.io/service-weight: |
  <new-svc-name>:<new-svc-weight>, <old-svc-name>:<old-svc-weight>
参数说明：
new-svc-name：新版本服务名称
new-svc-weight：新版本服务权重
old-svc-name：旧版本服务名称
old-svc-weight：旧版本服务权重
```

  服务权重配置示例：

```
nginx.ingress.kubernetes.io/service-weight: |
  new-nginx: 20, old-nginx: 60
```

步骤一：部署服务

部署Nginx服务并通过Ingress Controller对外提供7层域名访问。

1. 创建Deployment和Service。

i. 创建*nginx.yaml*。

```
apiVersion: apps/v1
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/acs-sample/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
```

ii. 执行以下命令，创建Deployment和Service。

```
kubectl apply -f nginx.yaml
```

2. 部署Ingress。

i. 创建*ingress.yaml*。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

ii. 执行以下命令，部署Ingress。

```
kubectl apply -f ingress.yaml
```

3. 测试访问情况。

i. 执行以下命令，获取外部IP。

```
kubectl get ingress
```

ii. 执行以下命令，查看路由访问情况。

```
curl -H "Host: www.example.com"  http://<EXTERNAL_IP>
```

预期输出：

```
old
```

步骤二：灰度发布新版本服务

发布一个新版本的Nginx服务并配置路由规则。

1. 部署新版本的Deployment和Service。

i. 创建*nginx1.yaml*

```
apiVersion: apps/v1
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/acs-sample/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
```

ii. 部署新版本的Deployment和Service。

```
kubectl apply -f nginx1.yaml
```

2. 设置访问新版本服务的路由规则。

   ACK支持设置以下三种路由规则，您可以根据实际情况选择路由规则。

   - 设置满足特定规则的客户端才能访问新版本服务。以下示例仅请求头中满足`foo=bar`的客户端请求才能路由到新版本服务。

a. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
 annotations:
  # 请求头中满足正则匹配foo=bar的请求才会被路由到新版本服务new-nginx中
  nginx.ingress.kubernetes.io/service-match: |
   new-nginx: header("foo", /^bar$/)
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

b. 查看路由访问情况。

- 执行以下命令，访问服务。

  ```
  curl -H "Host: www.example.com" http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  old
  ```

- 执行以下命令，请求头中满足 `foo=bar` 的客户端请求访问服务。

  ```
  curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  new
  ```

重复执行以上命令，可以看到，仅请求头中满足 `foo=bar` 的客户端请求才能路由到新版本服务。

- 在满足特定规则的基础上设置一定比例的请求被路由到新版本服务中。以下示例要求请求头中满足 `foo=bar` 的客户端请求，且仅允许其中50%的流量被路由到新版本服务中。

a. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
 annotations:
  # 请求头中满足正则匹配foo=bar的请求才会被路由到新版本服务new-nginx中
  nginx.ingress.kubernetes.io/service-match: |
    new-nginx: header("foo", /^bar$/)
  # 在满足上述匹配规则的基础上仅允许50%的流量会被路由到新版本服务new-nginx中
  nginx.ingress.kubernetes.io/service-weight: |
    new-nginx: 50, old-nginx: 50
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

b. 查看路由访问情况。

- 执行以下命令，访问服务。

  ```
  curl -H "Host: www.example.com"  http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  old
  ```

- 执行以下命令，请求头中满足 `foo=bar` 的客户端请求访问服务。

  ```
  curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  new
  ```

  重复执行以上命令。可以看到，仅请求头中满足 `foo=bar` 的客户端请求，且只有50%的流量才能路由到新版本服务。

- 设置一定比例的请求被路由到新版本服务中，以下示例中仅50%的流量被路由到新版本服务中。

a. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
 annotations:
   # 允许50%的流量被路由到新版本服务new-nginx中
   nginx.ingress.kubernetes.io/service-weight: |
     new-nginx: 50, old-nginx: 50
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

b. 执行以下命令，查看路由访问情况。

```
curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

重复执行以上命令，可以看到仅50%的流量路由到新版本服务。

步骤三：删除老版本服务

系统运行一段时间后，当新版本服务已经稳定并且符合预期后，需要下线老版本的服务，仅保留新版本服务在线上运行。

1. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

2. 执行以下命令，查看路由访问情况。

```
curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

预期输出：

```
new
```

重复执行以上命令，可以看到请求全部被路由到了新版本的服务。

3. 删除旧版本的Deployment和Service。
   i. 执行以下命令，删除旧版本的Deployment。

   ```
   kubectl delete deploy <Deployment名称>
   ```

   ii. 执行以下命令，删除旧版本的Service。

   ```
   kubectl delete svc <Service名称>
   ```

7.5.3.3. 使用Ingress-sentinel实现流控

ACK提供了Ingress-sentinel流控功能，通过Ingress-sentinel流控功能可以快速接入到AHAS中。当有请求流量时，您可以在AHAS控制台中查看Nginx网关请求的实时QPS和RT等数据。本文介绍如何使用Ingress-sentinel功能实现流控。

前提条件

- 开通AHAS
- 创建Kubernetes托管版集群
- 请确保ACK集群中的Ingress Controller运行正常，且不低于0.44.0版本。

背景信息

应用高可用服务AHAS（Application High Availability Service）提供了应用架构探测感知，故障注入式高可用能力评测和流控降级高可用防护三大核心能力，其中针对Nginx的流控降级高可用防护功能已集成到ACK提供的Ingress-Nginx-Controller中，您可用根据实际需求开启该功能，查看流控数据。

使用限制

开启Ingress-sentinel流控功能，会消耗一定的系统资源。您使用的机器需要符合一定的配置，以下为配置说明：

- 最低配置：CPU为1000m Core，内存为2048 MiB。

- 建议配置：CPU为4000m Core及以上，内存为8192 MiB及以上。

开启Ingress-sentinel流控功能

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在配置项页面名称搜索框中搜索nginx-configuration，找到nginx-configuration，然后单击nginx-configuration**操作**列的**编辑**。
6. 自定义应用名称。

   在**编辑**面板单击**添加**，设置**名称**为sentinel-params，**值**为--app=demo-k8s-2020。

   > **说明** --app=demo-k8s-2020字段中demo-k8s-2020为应用名称，您可以根据实际需求替换应用名称。

7. （可选）如果您的集群所在地域不属于深圳、北京、上海、张家口、杭州，则还需要配置证书。

   单击**添加**，设置**名称**为sentinel-params，**值**为--app=demo-k8s --license=xxx。

   > **说明** 关于获取License的更多信息，请参见查看License。

8. （可选）如果您希望接收到超过流控配置上限的请求时，系统能够返回HTTP 429错误码（表示请求数过多）。则您可以配置流控HTTP返回码。

   单击**添加**，设置**名称**为sentinel-block-action，**值**为=429。

   > **说明** 如果您想要自定义HTTP错误码，可以根据实际需求修改429。

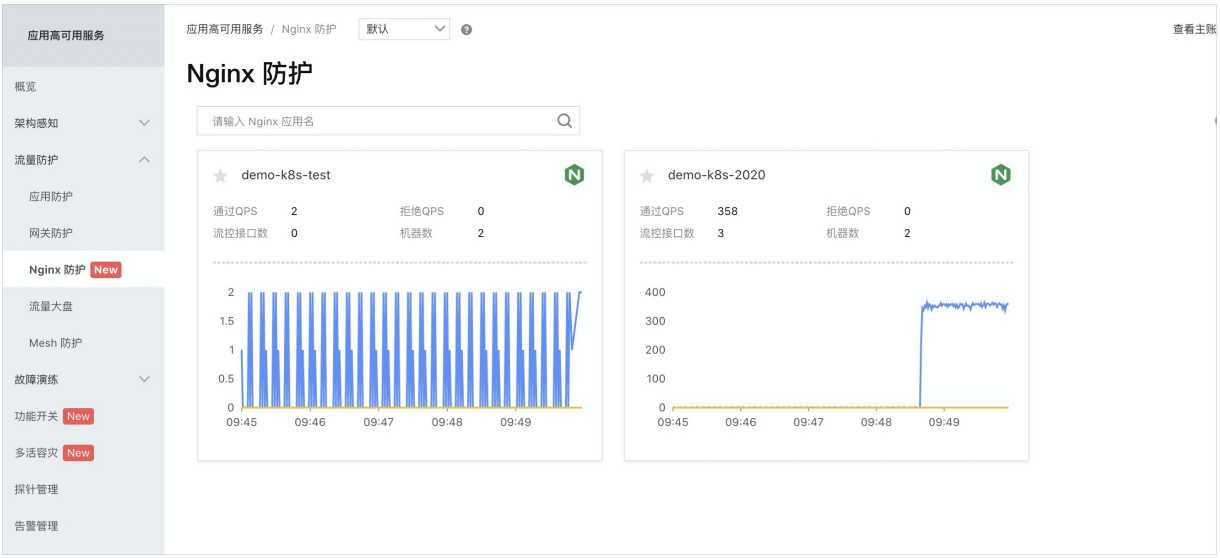
9. 开启Ingress-sentinel流控功能。

   > **说明** 接入Ingress-sentinel流控功能后默认不会触发流控，只有配置开启流控规则，并且请求流量超过流控规则限制后才会触发流控。

   单击**添加**，设置**名称**为use-sentinel，**值**为true。然后单击**确定**。

配置流控规则

登录AHAS控制台，在控制台左侧导航栏选择**流量防护 > Nginx防护**，单击目标应用卡片，可以看到应用名称对应的流控监控窗口。具体操作请参见配置流控规则、配置隔离规则或配置熔断规则。



如果您需要配置更详细的流控规则，请参见管理API和配置流控规则。规则配置完成后，您可以在**应用概览**页面实时查看流控指标。



7.5.3.4. 使用Ingress-tracing实现链路追踪

ACK提供了Ingress-tracing链路追踪功能，通过Ingress-tracing链路追踪功能可以接入到链路追踪控制台中。您可以在链路追踪控制台查看调用链路、链路拓扑等。本文介绍如何使用Ingress-tracing实现链路追踪。

前提条件

- 开通相关服务并授权
- 创建Kubernetes托管版集群

背景信息

阿里云提供了链路追踪服务，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、链路拓扑等，能够帮助快速分析和诊断分布式应用架构下的性能瓶颈，提高微服务时代下的开发诊断效率。ACK提供的Ingress-Nginx-Controller支持集成链路追踪服务，您可以根据需求开启该功能，查看链路追踪数据。

步骤一：获取接入点信息

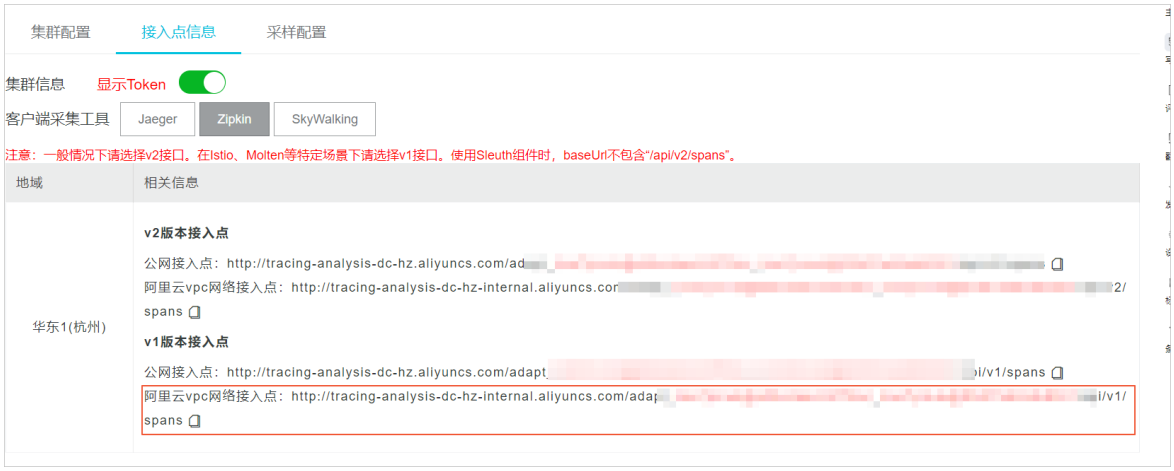
获取接入点信息，根据实际情况选择客户端采集工具，本文以Zipkin为例。

1. 登录链路追踪Tracing Analysis控制台。
2. 在控制台页面顶部选择地域。

   > **说明** 地域需要与步骤二：在Ingress中开启链路追踪服务的ACK集群保持一致。

3. 在左侧导航栏单击**集群配置**。
4. 在**集群配置**页面上单击**接入点信息**页签，在**集群信息**区域打开**显示Token**开关。
5. 在**客户端采集工具**区域单击**Zipkin**。
6. 在下方表格的相关信息列中，单击接入点信息末尾的复制图标。

   > **说明** 建议选择v1版本接入点。



步骤二：在Ingress中开启链路追踪服务

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在配置项页面上方设置命名空间为**kube-system**，然后在名称搜索框中搜索nginx-configuration，找到nginx-configuration，然后单击nginx-configuration**操作**列的**编辑**。
6. 配置链路追踪方式。

   根据使用的链路追踪方式，需要配置不同的字段，本文采用zipkin来提供链路追踪服务。关于更多链路追踪方式信息，请参见准备工作概述。
   在**编辑**面板单击**添加**，设置**名称**为*zipkin-collector-host*，**值**为步骤一：获取接入点信息获取的接入点信息。

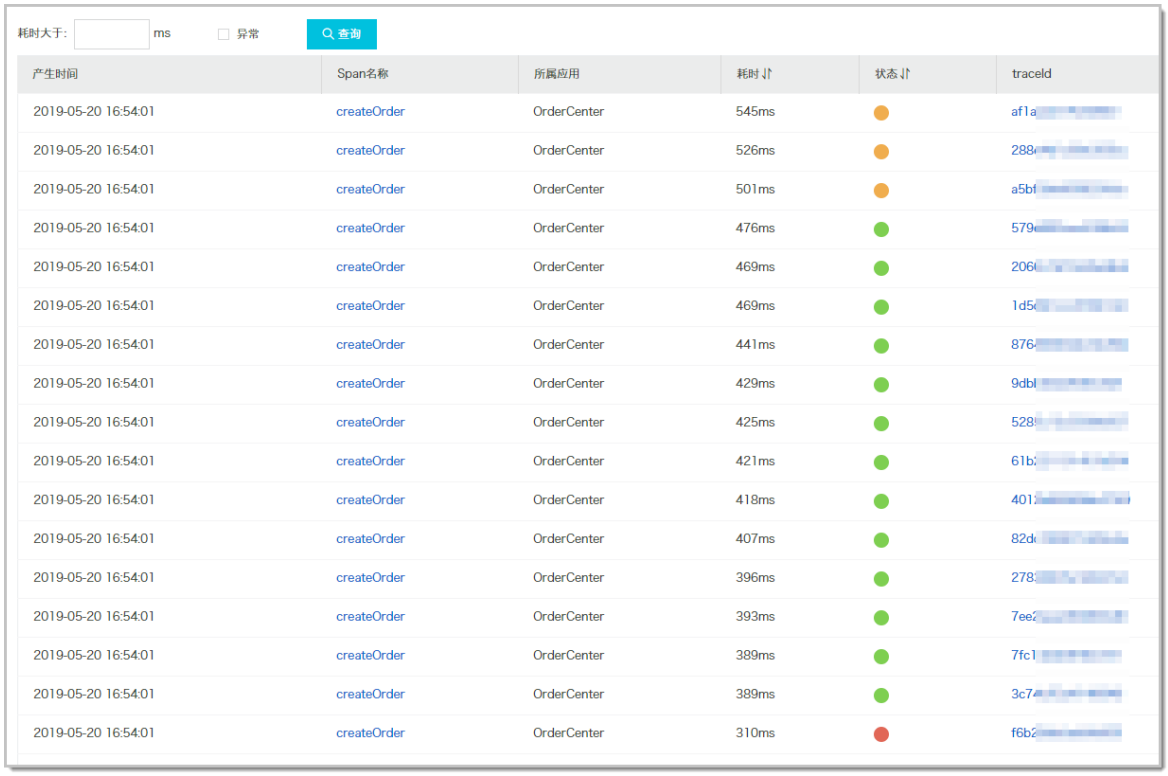
   > **说明** 该接入点信息需要去掉HTTP，并在末尾加上问号。例如接入点信息为 `http://tracing-analysis-dc-hz-internal.aliyuncs.com/adapt_******_******/api/v1/spans` ，则输入的值为 `tracing-analysis-dc-hz-internal.aliyuncs.com/adapt_******_******/api/v1/spans?` 。

7. 开启链路追踪服务。

   单击**添加**，设置**名称**为*enable-opentracing*，**值**为*true*，然后单击**确定**。

步骤三：查看调用链路数据

1. 登录链路追踪Tracing Analysis控制台。
2. 在控制台左侧导航栏中单击**应用列表**。
3. 在**应用列表**页面顶部选择地域，然后单击应用名称。
4. 在应用详情页面左侧导航栏中单击**接口调用**，在左侧的接口列表中单击调用链路页签，调用链路页签列出了该应用耗时最长的至多100个调用链路。关于更多链路数据，请参见查看接口调用情况。



7.5.3.5. Ingress访问日志分析与监控

阿里云Ingress组件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，还支持将所有您的HTTP请求日志记录到标准输出中。您可以在创建集群时启用日志服务，在阿里云日志服务SLS查看Ingress的访问日志分析报表和监控Ingress实时状态。本文介绍Ingress访问日志的部署及查看。

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，请参见通过日志服务采集Kubernetes容器日志进行手动安装。
2. 升级Log Controller。
   通过执行 `kubectl edit deployment alibaba-log-controller -n kube-system` 命令修改镜像版本进行升级。

部署Ingress采集配置

日志采集配置针对ACK进行了CRD扩展，您可以直接部署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。

> **注意**
> 部署AliyunLogConfig的CRD时需要注意以下事项：
>
> - 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。
> - 部署AliyunLogConfig的CRD只针对ACK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，请修改CRD配置中的正则表达式提取 `processor_regex` 部分，具体操作，请参见通过DaemonSet-CRD方式采集容器日志。
> - `IncludeLabel` 为Docker inspect中的Label信息。
> - Kubernetes中的命名空间和容器名会映射到Docker的Label中，分别为 `io.kubernetes.pod.namespace` 和 `io.kubernetes.container.name` 。例如，您创建的Pod所属命名空间为backend-prod，容器名为worker-server。
>   - 如果配置Label白名单为 `io.kubernetes.pod.namespace:backend-prod` ，则收集对应Pod中所有容器的日志。
>   - 如果配置Label白名单为 `io.kubernetes.container.name:worker-server` ，则收集对应容器的日志。
> - Kubernetes不建议使用除 `io.kubernetes.pod.namespace` 和 `io.kubernetes.container.name` 之外的其他Label。其他情况请使用IncludeEnv或ExcludeEnv，具体操作，请参见通过DaemonSet-控制台方式采集容器标准输出。
>
> 如果不需要Ingress日志采集，删掉AliyunLogConfig配置即可。

执行以下命令，部署AliyunLogConfig的CRD配置。

```
cat <<-EOF | kubectl apply -n kube-system -f -
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 #如果配置多个ingress controller请注意预防日志重复采集，详细请阅读注意事项中IncludeLabel释义
      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
        - proxy_alternative_upstream_name
        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*)\s*\[*([^]]*)\]*.*
        SourceKey: content
EOF
```

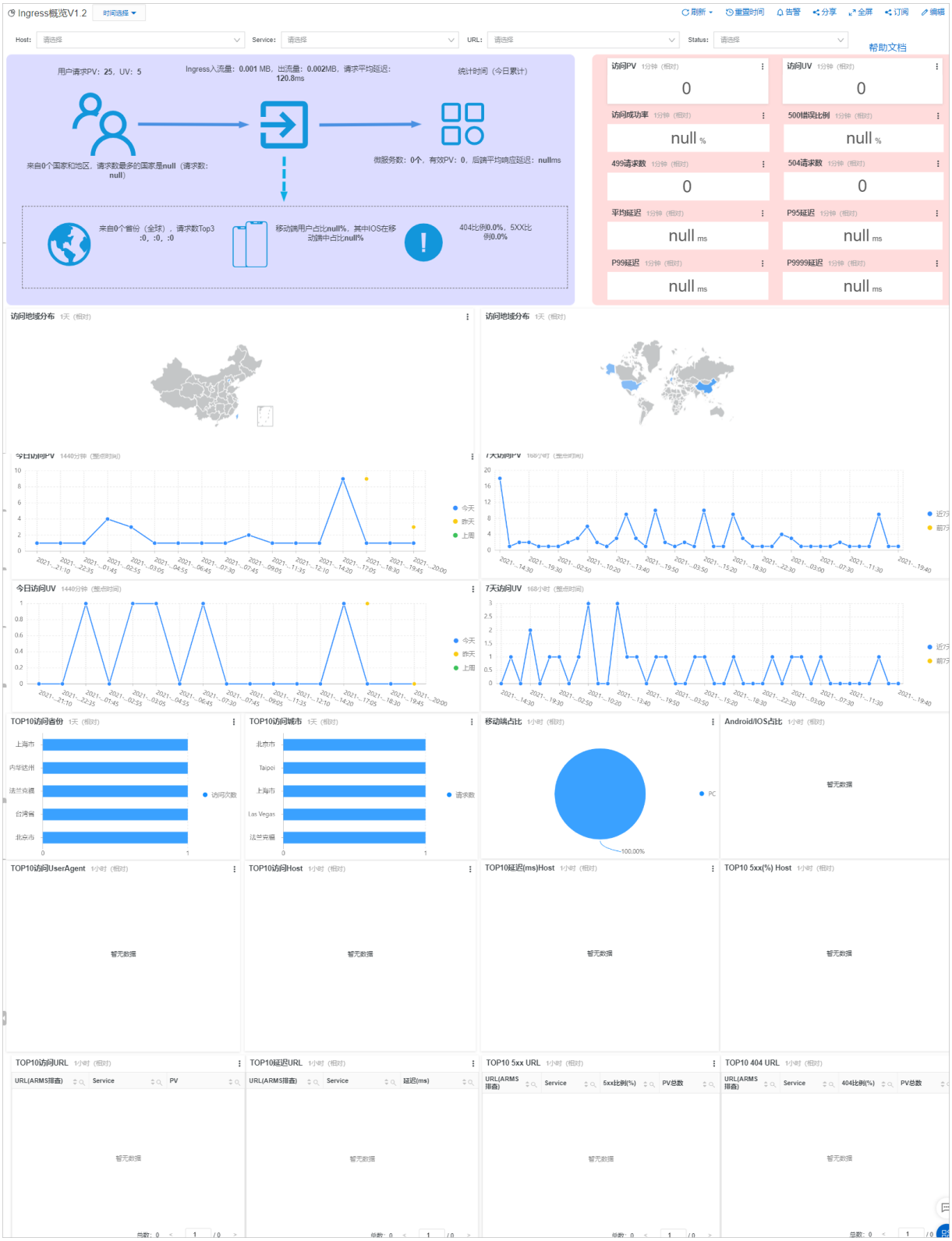
查看Ingress日志与报表

1. 登录日志服务控制台。
2. 在Project列表区域，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. 在日志库页面左侧导航栏选择**nginx-ingress > 可视化仪表盘**，查看所有Ingress的分析报表。

Ingress概览

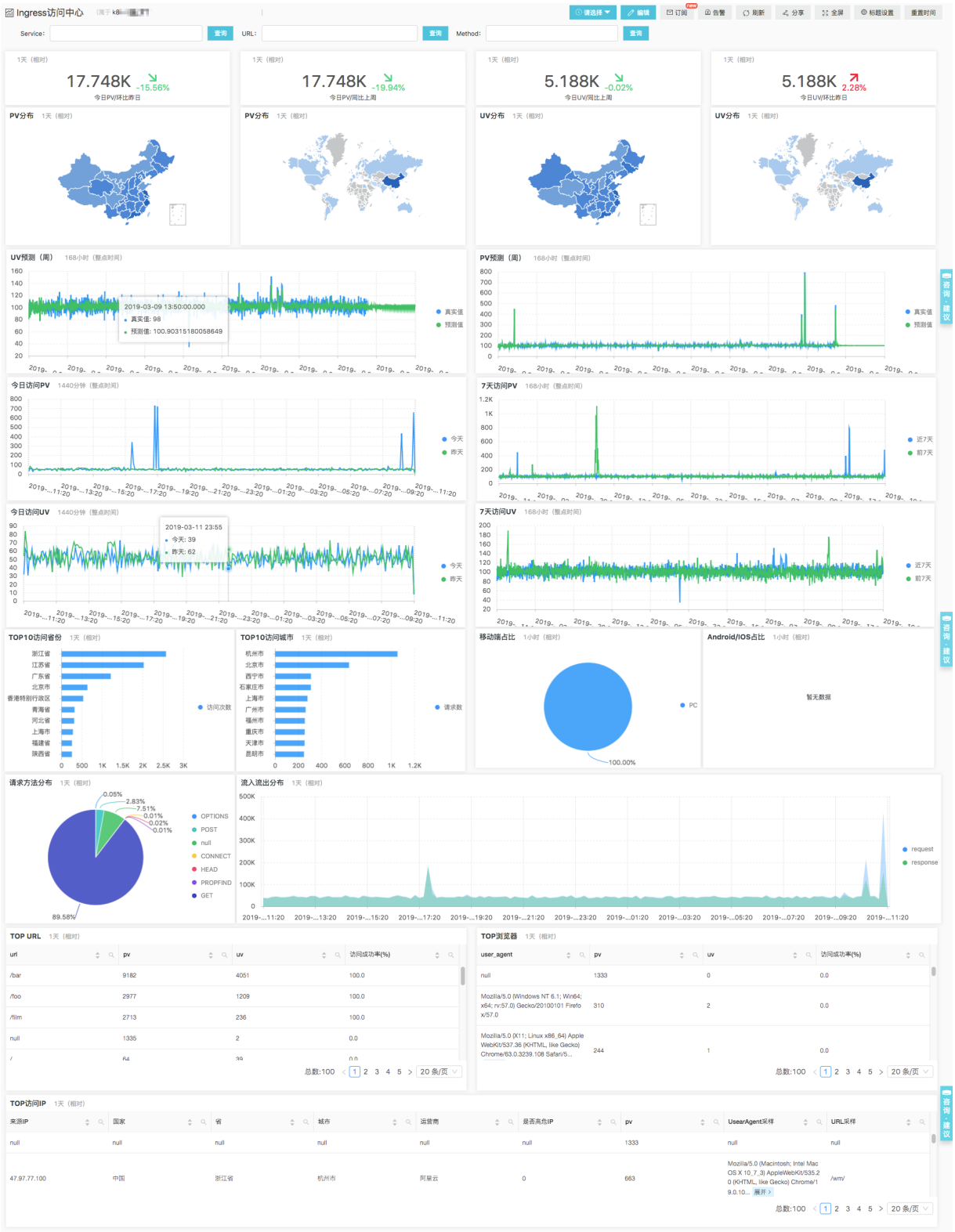
Ingress概览报表主要展示当前Ingress的整体状态，主要包括以下几类信息：

- 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），包括访问PV、访问UV、访问成功率、平均延迟、P95延迟和P99延迟等。
- 用户请求类信息（1天），包括1天访问PV对比、7天访问PV对比、地域分布、TOP访问省份、TOP访问城市、移动端占比和Android/IOS占比等。
- TOP URL统计（1小时），包括访问TOP10、延迟TOP10、5XX错误TOP10和404错误TOP10。



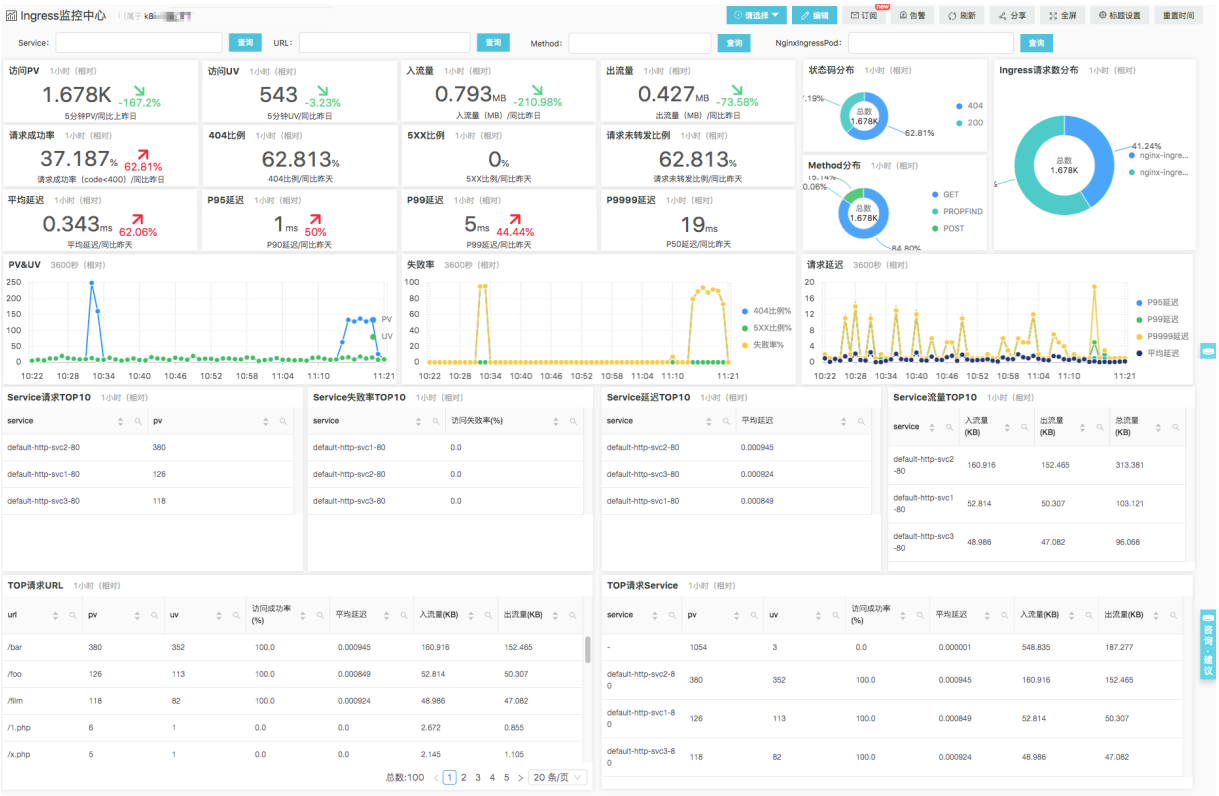
Ingress访问中心

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，包括今日UV、今日PV、UV分布、PV分布、TOP访问省份、TOP访问城市、TOP访问浏览器、TOP访问IP、移动端占比和Android/IOS占比等。



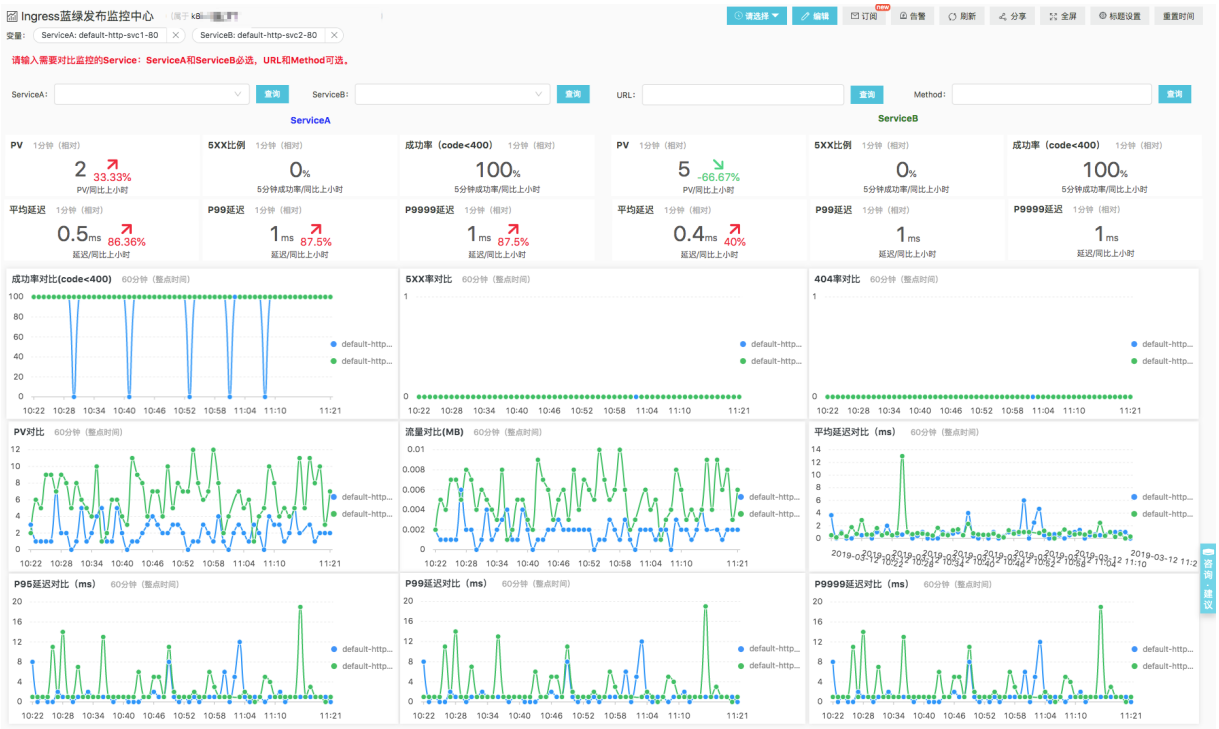
Ingress监控中心

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延迟、P99延迟、P999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。



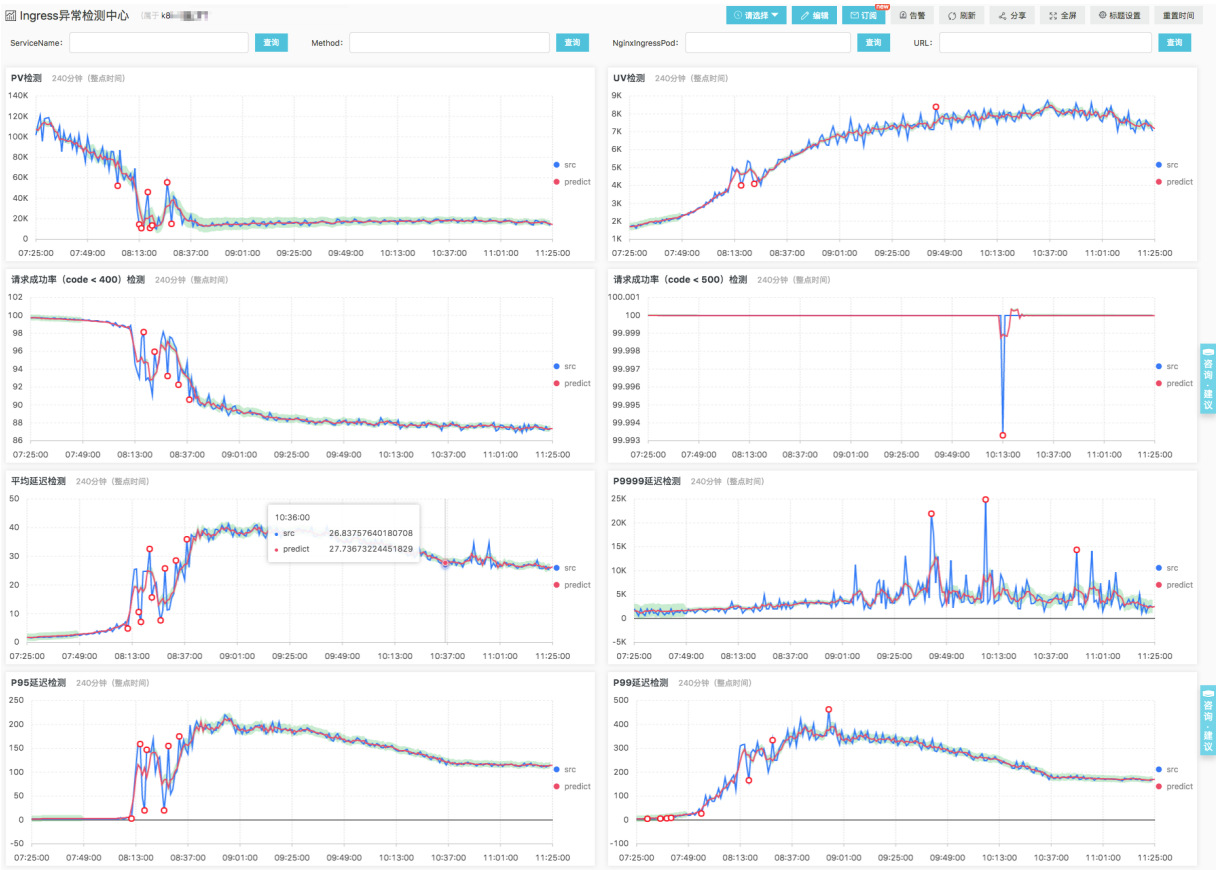
Ingress蓝绿发布监控中心

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延迟、P99延迟、P999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见设置告警。

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**右上角的⋮，在弹出的提示框中单击**创建告警**。



2. 在**告警配置**页签中，填入**告警名称**、**查询区间**、**频率**、**触发条件**，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，然后单击**提交**，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见订阅仪表盘。

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V1.2**报表，选择报表右上角的**订阅 > 创建**。
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，单击**下一步**。
3. 通知类型中选择**WebHook钉钉机器人**，填入钉钉机器人的WebHook地址（WebHook地址请参见获取Webhook地址），然后单击**提交**，即可完成订阅。

7.5.3.6. 部署高负载场景的Ingress Controller

在Kubernetes集群中，Ingress对集群服务（Service）中外部可访问的API对象进行管理，提供七层负载均衡能力。Ingress Controller负责实现Ingress的功能，根据规则配置负载均衡并提供访问入口。在高负载场景下，CPU资源和网络连接数的不足会影响应用的性能。本文介绍如何通过部署Ingress Controller来支撑高负载应用。

前提条件

- 已创建Kubernetes集群。具体操作步骤，请参见创建Kubernetes托管版集群。

- ACK集群中的Ingress Controller运行正常。
- 已安装kubectl命令行工具。

部署说明

部署高负载场景下的Ingress Controller需要考虑以下三个方面。

- 硬件选型
  在高并发场景下，Ingress对CPU资源和网络连接数占用都非常高，所以可选增强型ECS实例，例如：
  - 计算型实例：ecs.c6e.8xlarge（32 Core 64 GB，600万PPS）
  - 网络型实例：ecs.g6e.8xlarge（32 Core 128 GB，600万PPS）
- K8s配置
  - 设置Ingress Pod独占节点资源，添加污点和节点标签：

    ```
    kubectl label nodes $node_name ingress-pod="yes"
    kubectl taint nodes $node_name ingress-pod="yes":NoExecute
    ```

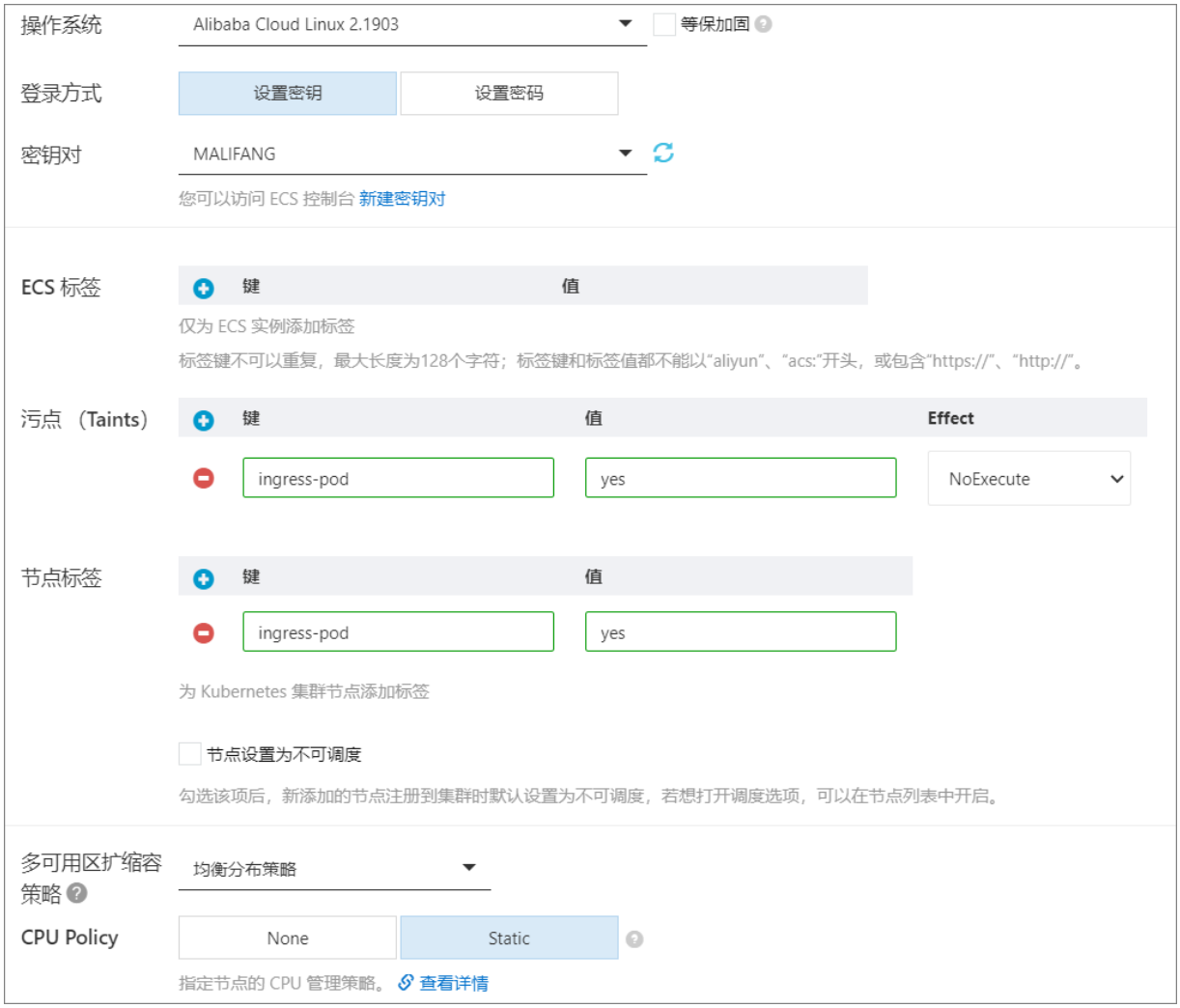
  - 设置CPU Policy为 `static` 。
  - 推荐调整ingress-controller service对应的SLB规格为超强型（slb.s3.large）。
  - 推荐集群使用Terway网络插件及配置独占ENI。
- Ingress配置
  - 设置Ingress Pod为Guaranteed类型。
    - 设置nginx-ingress-controller container的资源限制 `requests` 和 `limits` ：15 Core 20 GiB。
    - 设置init Container init-sysctl的资源限制 `requests` 和 `limits` ：100 m 70 MiB。
  - 删除Ingress Pod中的 `podAntiAffinity` ，使一个节点上可调度两个Pod。
  - 调整Deployment Replicas数为新增节点数的2倍。
  - 设置ConfigMap的 `worker-processes` 数为15（预留部分给系统使用）。
  - 调整ConfigMap的 `keepalive` 链接最大请求数。
  - 关闭日志访问记录。

步骤一：添加节点

选择指定集群，创建新的节点池并添加2台节点。

创建节点池需要配置的主要参数如下所示。具体操作，请参见管理节点池。

- 选择**操作系统**为Alibaba Cloud Linux 2.1903。
- 配置**节点标签**和**污点（Taints）**。
  - 设置**污点（Taints）**的**键**为*ingress-pod*，**值**为*yes*，**Effect**为NoExecute。
  - 设置**节点标签**的**键**为*ingress-pod*，**值**为*yes*。
- 选择**CPU Policy**为Static。



步骤二：配置Ingress Controller

执行 `kubectl edit deploy nginx-ingress-controller -n kube-system` 命令打开Ingress Controller的配置文件，根据以下内容更新Ingress Controller的配置：

- 删除Pod反亲和性配置。

```
podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
    matchExpressions:
    - key: app
    operator: In
    values:
    - ingress-nginx
  topologyKey: kubernetes.io/hostname
```

- 配置Init Container的 `requests` 和 `limits` 。

```
resources:
 limits:
  cpu: 100m
  memory: 70Mi
 requests:
  cpu: 100m
  memory: 70Mi
```

- 修改nginx-ingress-controller container的 `requests` 和 `limits` 都为15 Core 20 GiB。

```
resources:
 limits:
  cpu: "15"
  memory: 20Gi
 requests:
  cpu: "15"
  memory: 20Gi
```

- 设置节点亲和性和容忍性。

```
nodeSelector:
 ingress-pod: "yes"
tolerations:
- effect: NoExecute
 key: ingress-pod
 operator: Equal
 value: "yes"
```

- 调整Replicas数为等于新增节点数的2倍。
- 关闭Metrics采集，修改启动参数，添加 `--enable-metrics=false` 。

> **说明** 如果不需要获取Metrics信息，推荐关闭Metrics采集。

```
containers:
- args:
 - /nginx-ingress-controller
 - --configmap=$(POD_NAMESPACE)/nginx-configuration
 - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
 - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
 - --annotations-prefix=nginx.ingress.kubernetes.io
 - --publish-service=$(POD_NAMESPACE)/nginx-ingress-lb
 - --enable-metrics=false
 - --v=1
```

步骤三：配置Ingress ConfigMap

1. 使用 `kubectl edit cm nginx-ingress-controller -n kube-system` 更新ConfigMap，可参考以下代码示例：

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: nginx-configuration
 namespace: kube-system
data:
 allow-backend-server-header: "true"
 enable-underscores-in-headers: "true"
 generate-request-id: "true"
 ignore-invalid-headers: "true"
 log-format-upstream: $remote_addr - [$remote_addr] - $remote_user [$time_local] "$request" $status $bod
y_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time [$proxy_upstream_name]
$upstream_addr $upstream_response_length $upstream_response_time $upstream_status $req_id $host [$
proxy_alternative_upstream_name]
 max-worker-connections: "65536"
 proxy-body-size: 20m
 proxy-connect-timeout: "3"
 proxy-read-timeout: "5"
 proxy-send-timeout: "5"
 reuse-port: "true"
 server-tokens: "false"
 ssl-redirect: "false"
 upstream-keepalive-timeout: "900"
 worker-processes: 15
 worker-cpu-affinity: auto
 upstream-keepalive-connections: "300"
 upstream-keepalive-requests: "1000"
 keep-alive: "900"
 keep-alive-requests: "10000"
 disable-access-log: "true"
```

> **说明**
> - worker-processes：设置启动Nginx进程数。
> - keep-alive-requests：设置单个长连接最多可以发送多个。
> - disable-access-log：关闭日志访问记录。

2. 采用文件方式记录访问日志并对日志轮转。

   默认的日志访问记录输出到*/dev/stdout*，在高并发场景下，会占用大量CPU，故采用文件方式记录访问日志并对日志设置轮转。

   i. 以SSH方式登录部署ingress-controller Pod的ECS节点上。具体操作，请参见如何使用SSH登录Linux服务器？。

   ii. 在*/etc/crontab*文件结尾添加以下内容：

   ```
   */15 * * * * root /root/nginx-log-rotate.sh
   ```

   > **说明** 此处以每隔15分钟对日志轮转一次为例，可根据实际需求进行调整。

iii. 在*/root*目录下添加文件*nginx-log-rotate.sh*。

```
#!/bin/bash
# 最多保留日志文件个数，可根据需求进行调整。
keep_log_num=5
ingress_nginx_container_ids=$(docker ps | grep nginx-ingress-controller | grep -v pause | awk '{print $1}')
if [[ -z "$ingress_nginx_container_ids" ]]; then
  echo "error: failed to get ingress nginx container ids"
  exit 1
fi
#随机睡眠5~10秒
sleep $(( RANDOM % (10 - 5 + 1 ) + 5 ))
for id in $ingress_nginx_container_ids; do
  docker exec $id  bash -c "cd /var/log/nginx; if [[ \$(ls access.log-* | wc -l) -gt $keep_log_num ]]; then rm -f \
$(ls -t access.log-* | tail -1); fi ; mv access.log  access.log-\$(date +%F:%T) ; kill -USR1 \$(cat /tmp/nginx.pid)"
done
```

iv. 执行以下命令，对*nginx-log-rotate.sh*文件添加可执行权限。

```
chmod 755 /root/nginx-log-rotate.sh
```

7.5.3.7. 部署高可靠Ingress Controller

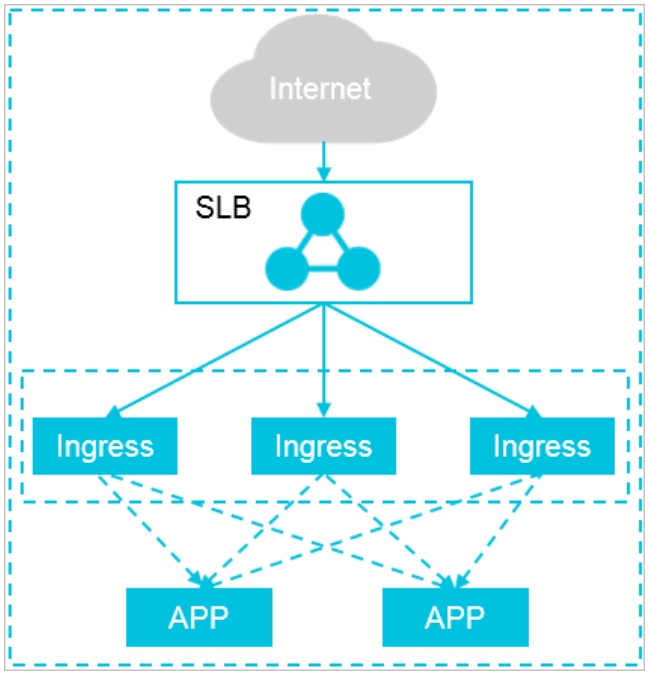
在Kubernetes集群中，Ingress是授权入站连接到达集群服务的规则集合，可以为您提供七层负载均衡能力，您可以通过Ingress配置提供外部可访问的URL、负载均衡、SSL、基于名称的虚拟主机等。作为集群流量接入层，Ingress的高可靠性显得尤为重要，本文介绍如何部署一套高性能、高可靠的Ingress接入层。

前提条件

- 创建Kubernetes托管版集群
- 通过kubectl工具连接集群

高可靠部署架构

高可靠性首先要解决的就是单点故障问题，通常系统采用多副本部署的方式，在Kubernetes集群中部署高可靠Ingress接入层同样采用多节点部署架构，同时由于Ingress作为集群流量接入口，建议系统采用一个Ingress服务独占一个Ingress节点的方式，以避免业务应用与Ingress服务发生资源抢占。



如上述部署架构图，由多个独占Ingress实例组成统一接入层来承载集群入口流量，同时可依据后端业务流量水平扩缩容Ingress节点。若您前期的集群规模并不大，也可以采用将Ingress服务与业务应用混部的方式，但注意进行资源限制和隔离。

查看集群默认部署的Pod副本及公网地址

当您成功创建一个集群后，默认情况下，集群内部已经部署了一套拥有2个Pod副本的Nginx Ingress Controller服务，其前端挂载在一个公网SLB实例上。

1. 执行以下命令查看部署Nginx Ingress Controller服务的Pod。

   ```
   kubectl -n kube-system get pod | grep nginx-ingress-controller
   ```

   预期输出：

   ```
   nginx-ingress-controller-8648ddc696-2bshk          1/1   Running  0     3h
   nginx-ingress-controller-8648ddc696-jvbs9          1/1   Running  0     3h
   ```

2. 执行以下命令查看nginx-ingress-lb服务对应的公网SLB地址。

   ```
   kubectl -n kube-system get svc nginx-ingress-lb
   ```

   预期输出：

   ```
   NAME         TYPE      CLUSTER-IP   EXTERNAL-IP    PORT(S)           AGE
   nginx-ingress-lb  LoadBalancer  172.XX.XX.XX  118.XX.XX.XX  80:32XXX/TCP,443:31XXX/TCP  21d
   ```

部署高可靠的Ingress接入层

随着集群业务规模的逐渐扩大，系统需要扩容Ingress接入层，以保证集群接入层的高性能高可用，可通过以下两种方法完成。

- **方法一：增加Pod副本数量**

  通过调整Nginx Ingress Controller Deployment的Pod副本数量，可以快速增加Ingress接入层的规模。

  i. 执行以下命令，将Pod副本数量扩容到3个。

  ```
  kubectl -n kube-system scale --replicas=3 deployment/nginx-ingress-controller
  ```

  预期输出：

  ```
  deployment.extensions/nginx-ingress-controller scaled
  ```

  ii. 执行以下命令，查看部署Nginx Ingress Controller服务的Pod。

  ```
  kubectl -n kube-system get pod | grep nginx-ingress-controller
  ```

  预期输出：

  ```
  nginx-ingress-controller-8648ddc696-2bshk          1/1   Running  0     3h
  nginx-ingress-controller-8648ddc696-jvbs9          1/1   Running  0     3h
  nginx-ingress-controller-8648ddc696-xqmfn          1/1   Running  0     33s
  ```

- **方法二：在指定的高配置节点上部署Ingress服务**

  如果您需要Nginx Ingress Controller仅运行在指定的一些高配置节点上，可以通过给节点添加标签来完成。

  i. 执行以下命令查看当前集群的节点情况。

  ```
  kubectl get node
  ```

  预期输出：

```
NAME                STATUS  ROLES   AGE  VERSION
cn-hangzhou.i-bp11bcmsna8d4bp****  Ready   master  21d  v1.11.5
cn-hangzhou.i-bp12h6biv9bg24l****  Ready   <none>  21d  v1.11.5
cn-hangzhou.i-bp12h6biv9bg24l****  Ready   <none>  21d  v1.11.5
cn-hangzhou.i-bp12h6biv9bg24l****  Ready   <none>  21d  v1.11.5
cn-hangzhou.i-bp181pofzyyksie****  Ready   master  21d  v1.11.5
cn-hangzhou.i-bp1cbsg6rf3580z****  Ready   master  21d  v1.11.5
```

ii. 执行以下命令，给Ingress Node节点 `cn-hangzhou.i-bp12h6biv9bg24lmdc2o` 和 `cn-hangzhou.i-bp12h6biv9bg24lmdc2p` 添加标签 `node-role.kubernetes.io/ingress="true"` 。

```
kubectl label nodes cn-hangzhou.i-bp12h6biv9bg24lmdc2o node-role.kubernetes.io/ingress="true"
```

预期输出：

```
node/cn-hangzhou.i-bp12h6biv9bg24lmdc2o labeled
```

```
kubectl label nodes cn-hangzhou.i-bp12h6biv9bg24lmdc2p node-role.kubernetes.io/ingress="true"
```

预期输出：

```
node/cn-hangzhou.i-bp12h6biv9bg24lmdc2p labeled
```

> **说明**
>
> - 添加标签的节点数量要大于等于集群Pod副本数，从而避免多个Pod运行在同一个节点上。
> - 查询结果中，若ROLES的值显示为*none*，则表示为Worker节点。
> - 建议您不要将Ingress服务部署到Master节点上，尽量选择Worker节点添加标签。

iii. 执行以下命令，更新Deployment，增加nodeSelector配置。

```
kubectl -n kube-system patch deployment nginx-ingress-controller -p '{"spec":{"template":{"spec":{"nodeSelector":{"node-role.kubernetes.io/ingress":"true"}}}}}'
```

预期输出：

```
deployment.extensions/nginx-ingress-controller patched
```

iv. 执行以下命令，查看Ingress Pod已部署在添加了标签 `node-role.kubernetes.io/ingress="true"` 的集群节点上。

```
kubectl -n kube-system get pod -o wide|grep nginx-ingress-controller
```

预期输出：

```
nginx-ingress-controller-8648ddc696-2bshk   1/1   Running   0   3h   172.16.XX.XX   cn-hangzhou.i-bp****   <none>
nginx-ingress-controller-8648ddc696-jvbs9   1/1   Running   0   3h   172.16.XX.XX   cn-hangzhou.i-bp****   <none>
```

7.5.3.8. 部署Ingress Controller使用私网SLB

容器服务集群可以允许公网访问，也可以被同一个VPC下的其他服务直接访问。本文介绍如何配置阿里云容器服务Kubernetes Ingress Controller使用私网负载均衡SLB（Server Load Balancer）。

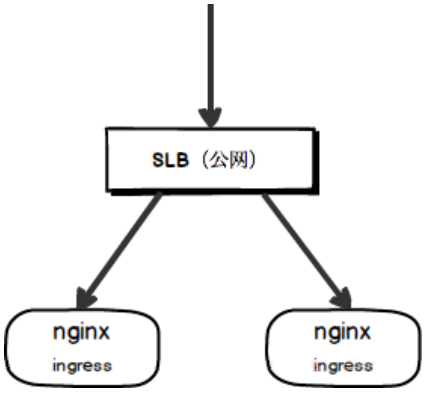
前提条件

- 您已成功创建一个容器服务Kubernetes集群，请参见创建Kubernetes托管版集群。

- SSH连接到Master节点，请参见通过SSH连接ACK专有版集群的Master节点。

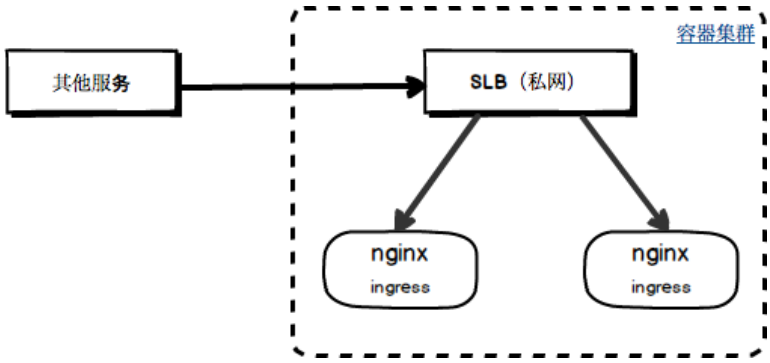
背景信息

通过阿里云容器服务管理控制台创建的Kubernetes集群在初始化时会自动部署一套Nginx Ingress Controller，默认其挂载在公网SLB实例上。



配置私网SLB

您可以调整Nginx Ingress Controller服务的配置，使容器集群内的服务仅仅只对同一个VPC内其他服务调用访问。



1. 手动申请私网SLB实例，请参见创建实例。

   > **说明** 在对应的VPC网络下申请一个期望规格的SLB实例。

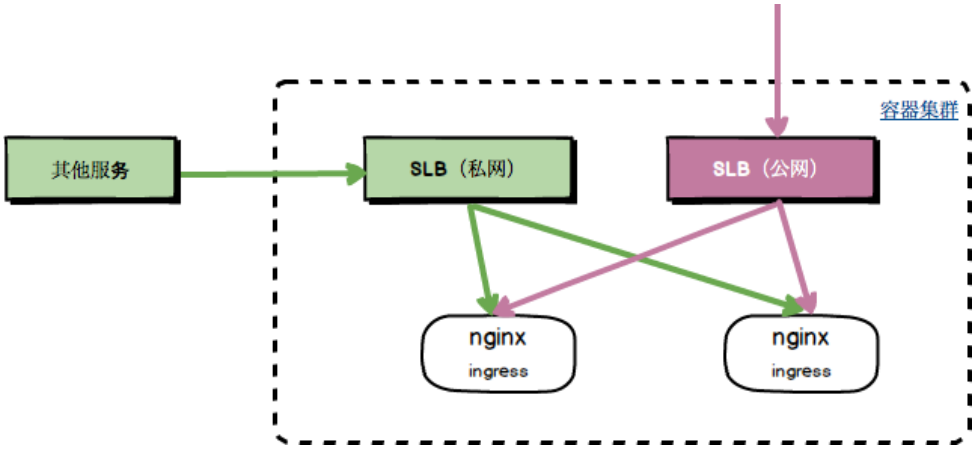
2. 配置Nginx Ingress Controller服务。
   当您成功创建一个私网SLB实例后，您可以通过以下示例注解来配置Nginx Ingress Controller使用该SLB实例，请参见通过Annotation配置负载均衡。

```
# nginx ingress slb service
apiVersion: v1
kind: Service
metadata:
 name: nginx-ingress-lb
 namespace: kube-system
 labels:
  app: nginx-ingress-lb
 annotations:
  # 指明SLB实例地址类型为私网类型。
  service.beta.kubernetes.io/alicloud-loadbalancer-address-type: intranet
  # 修改为您的私网SLB实例ID。
  service.beta.kubernetes.io/alicloud-loadbalancer-id: <YOUR_INTRANET_SLB_ID>
  # 是否自动创建SLB端口监听（会覆写已有端口监听），也可手动创建端口监听。
  #service.beta.kubernetes.io/alicloud-loadbalancer-force-override-listeners: 'true'
spec:
 type: LoadBalancer
 # route traffic to other nodes
 externalTrafficPolicy: "Cluster"
 ports:
 - port: 80
  name: http
  targetPort: 80
 - port: 443
  name: https
  targetPort: 443
 selector:
  # select app=ingress-nginx pods
  app: ingress-nginx
```

通过以上注解修改（apply）容器集群kube-system/nginx-ingress-lb服务即可配置使用指定的私网SLB实例。

同时使用私网SLB和公网SLB

对于一些特殊场景，您期望容器集群内的服务既能允许公网访问，同时又希望能被同一个VPC下的其他服务直接访问（不经过公网）。您只需额外部署一个kube-system/nginx-ingress-lb-intranet服务。



> **说明** 默认集群初始化时已经创建了一个kube-system/nginx-ingress-lb服务，其生成的是公网SLB实例。

1. 手动申请私网SLB实例，请参见创建实例。

> **说明** 在对应的VPC网络下申请一个期望规格的SLB实例。

2. 新建Intranet Nginx Ingress Controller服务。
   当您成功申请一个私网SLB实例后，您可以通过以下YAML新建一个kube-system/nginx-ingress-lb-intranet服务。

```
# intranet nginx ingress slb service
apiVersion: v1
kind: Service
metadata:
 # 这里服务取名为nginx-ingress-lb-intranet。
 name: nginx-ingress-lb-intranet
 namespace: kube-system
 labels:
  app: nginx-ingress-lb-intranet
 annotations:
  # 指明SLB实例地址类型为私网类型。
  service.beta.kubernetes.io/alicloud-loadbalancer-address-type: intranet
  # 修改为您的私网SLB实例ID。
  service.beta.kubernetes.io/alicloud-loadbalancer-id: <YOUR_INTRANET_SLB_ID>
  # 是否自动创建SLB端口监听（会覆写已有端口监听），也可手动创建端口监听。
  #service.beta.kubernetes.io/alicloud-loadbalancer-force-override-listeners: 'true'
spec:
 type: LoadBalancer
 # route traffic to other nodes
 externalTrafficPolicy: "Cluster"
 ports:
 - port: 80
  name: http
  targetPort: 80
 - port: 443
  name: https
  targetPort: 443
 selector:
  # select app=ingress-nginx pods
  app: ingress-nginx
```

成功创建kube-system/nginx-ingress-lb-intranet服务后，您可以通过 `kubectl -n kube-system get svc | grep nginx-ingress-lb` 命令看到以下两个Nginx Ingress Controller服务：一个配置的是公网SLB实例，一个是VPC私网SLB实例。

```
kubectl -n kube-system get svc | grep nginx-ingress-lb
```

```
nginx-ingress-lb        LoadBalancer  172.1*.*.**   47.96.2**.**  80:31456/TCP,443:30016/TCP  5h
nginx-ingress-lb-intranet  LoadBalancer  172.19.*.***  192.16*.*.**  80:32394/TCP,443:31000/TCP  7m
```

此时当您通过Ingress对外暴露服务时，即可以通过公网SLB来访问该服务，同一个VPC下的其他服务也可以直接通过私网SLB来访问该服务。

7.5.3.9. 部署多个Ingress Controller

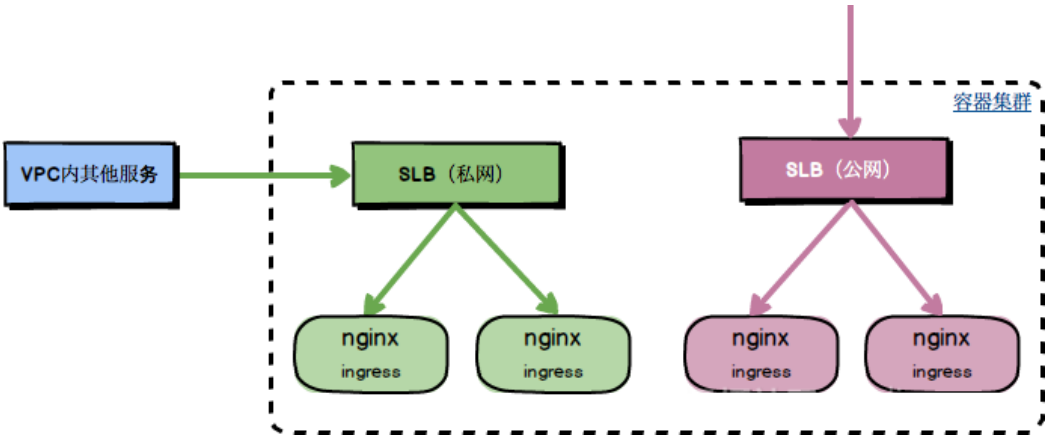
本文介绍如何在阿里云容器服务Kubernetes集群中同时部署多套独立的Nginx Ingress Controller对外提供不同的服务访问。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已通过kubectl连接kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

您可以通过调整阿里云容器服务Kubernetes集群中默认的Nginx Ingress Controller配置，使用私网SLB实例。更多信息，请参见部署Ingress Controller使用私网SLB，文中提到的两种模式可以满足大部分需求场景。但对于一些特殊场景，例如，集群内有部分公网服务需要通过公网Ingress方式来对外暴露提供访问，但是又有部分私网服务仅仅只希望对同VPC内非Kubernetes集群内的服务提供访问，而又不允许能被公网访问到，您可以通过部署两套独立的Nginx Ingress Controller服务，其前端绑定不同网络类型的SLB实例来满足这类需求场景。



部署新的Nginx Ingress Controller服务

当您成功创建一个ACK集群后，默认情况下，集群内部已经部署了一套拥有2个Pod副本的Nginx Ingress Controller服务，其前端挂载在一个公网SLB实例上。

您可以通过以下步骤在ACK集群中再部署一套完全独立的Nginx Ingress Controller服务。

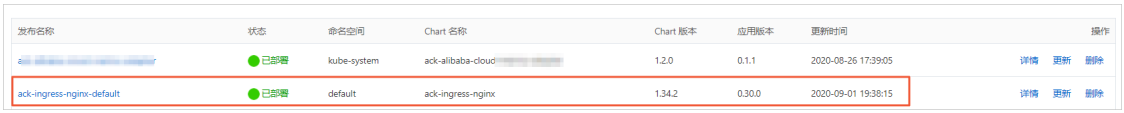
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**阿里云应用**页签，选中并单击**ack-ingress-nginx**。
4. 在**应用目录-ack-ingress-nginx**页面，单击**参数**页签并在右侧**创建**面板中设置目标集群、命名空间及发布名称，然后单击**创建**。

   主要参数设置如下表。

参数	描述
controller.image.repository	ingress-nginx镜像地址。对于国外地域，建议更改为对应的Region ID。
controller.image.tag	ingress-nginx镜像版本，请参见Nginx Ingress Controller。
controller.ingressClass	设置该Ingress Controller可监听的目标Ingress Class标识。 **注意** 同一个集群中不同套Ingress Controller监听的Ingress Class标识必须唯一，且不能设置为nginx关键字（nginx是集群默认Ingress Controller的监听标识）。
controller.replicaCount	设置该Ingress Controller Pod副本数。
controller.service.enabled	是否开启公网SLB访问，不需要开启则设置为false。
controller.service.internal.enabled	是否开启私网SLB访问，需要开启则设置为true。
controller.kind	设置IngressController部署形态，可选值：Deployment和DaemonSet。

5. 查看部署的Nginx Ingress Controller服务。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**应用 > Helm**。

   在**Helm**页面可以看到新的Nginx Ingress Controller服务已经成功部署。



访问测试

本文部署一个测试应用，并配置通过新部署的Nginx Ingress Controller来对外暴露提供服务访问。

1. 部署一个Nginx测试应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   run: nginx
 template:
  metadata:
   labels:
    run: nginx
  spec:
   containers:
   - image: nginx
    imagePullPolicy: Always
    name: nginx
    ports:
    - containerPort: 80
     protocol: TCP
   restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
 name: nginx
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 sessionAffinity: None
 type: NodePort
```

2. 通过Ingress来对外暴露提供服务访问。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: nginx
 annotations:
  #注意这里要设置为您前面配置的INGRESS_CLASS。
  kubernetes.io/ingress.class: "<YOUR_INGRESS_CLASS>"
spec:
 rules:
 - host: foo.bar.com
  http:
   paths:
   - path: /
    backend:
     service:
      name: nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

说明 您需要配置注释 kubernetes.io/ingress.class 。

在完成部署应用后，执行以下操作，查看该Ingress资源对应的端点IP地址与新部署的Nginx Ingress Controller服务的IP地址。

- 执行以下命令查看集群默认部署的nginx-ingress-lb服务对应的公网SLB地址。

  ```
  kubectl -n kube-system get svc nginx-ingress-lb
  ```

  预期输出：

  ```
  NAME         TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)           AGE
  nginx-ingress-lb  LoadBalancer  172.19.7.30  47.95.97.115  80:31429/TCP,443:32553/TCP  2d
  ```

- 执行以下命令查看新部署的nginx-ingress-lb服务对应的公网SLB地址。

  ```
  kubectl -n <YOUR_NAMESPACE> get svc nginx-ingress-lb
  ```

  预期输出：

  ```
  NAME         TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)           AGE
  nginx-ingress-lb  LoadBalancer  172.19.6.227  39.105.252.62  80:30969/TCP,443:31325/TCP  39m
  ```

- 执行以下命令查看该应用的Ingress配置。

  ```
  kubectl get ing
  ```

  预期输出：

  ```
  NAME    HOSTS     ADDRESS      PORTS   AGE
  nginx   foo.bar.com  39.105.252.62  80     5m
  ```

  通过以上预期输出可得，该Ingress资源对应的端点IP地址与新部署的Nginx Ingress Controller服务的一致。

3. 分别通过集群默认的Nginx Ingress Controller服务和新部署的Nginx Ingress Controller服务来访问该应用。

```
# 通过集群默认的Nginx Ingress Controller服务访问该应用（预期返回404）。
 curl -H "Host: foo.bar.com" http://47.95.97.115
default backend - 404
 # 通过新部署的Nginx Ingress Controller服务访问该应用（预期返回nginx页面）。curl -H "Host: foo.bar.com" http:/
/39.105.252.62
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

从上文测试访问情况可以看到，通过不同的Nginx Ingress Controller暴露的服务彼此完全是独立的，这特别适用于同一集群内部分服务需要提供公网访问能力，但又有部分服务仅仅只希望为同VPC内非Kubernetes集群的其他服务提供访问的场景。

相关文档

- 部署高可靠Ingress Controller
- 部署Ingress Controller使用私网SLB

7.5.3.10. 通过Ingress Controller来实现应用的流量复制

ACK支持在不同集群内进行流量复制，用于系统的仿真测试和问题定位。本文介绍如何通过Ingress Controller来实现应用的流量复制。

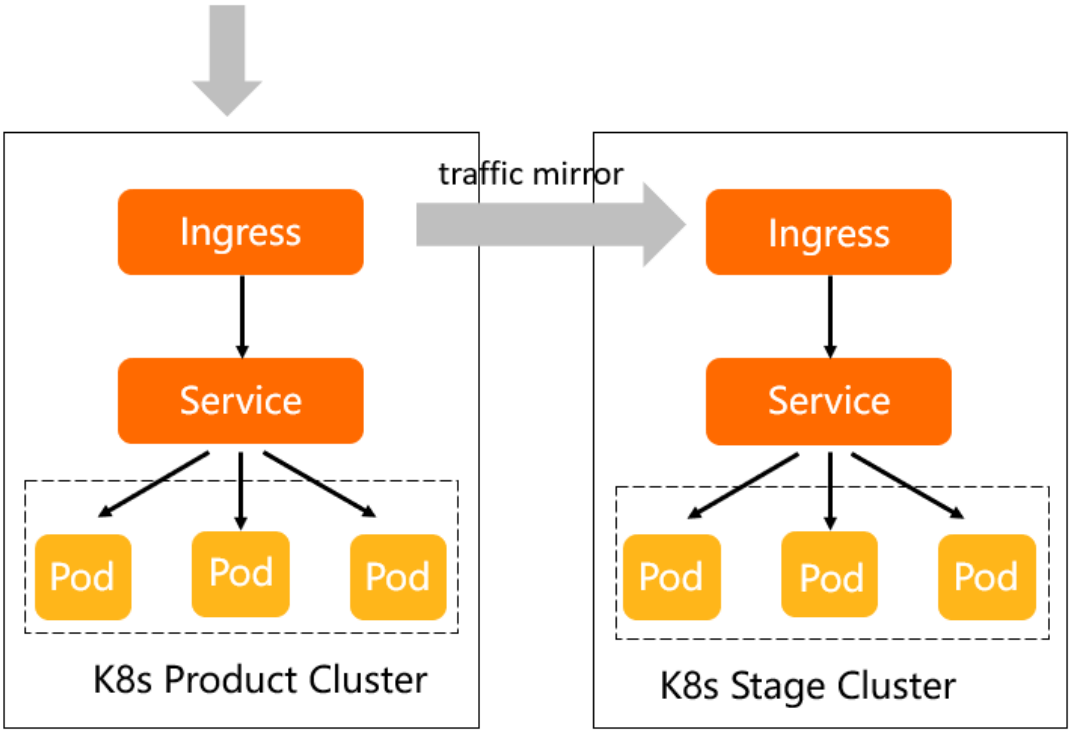
前提条件

已创建两个ACK集群，具体操作，请参见创建Kubernetes托管版集群。本例中ACK集群一个名为K8s Product Cluster（生产环境集群），另一个名为K8s Stage Cluster（测试环境集群）。

应用场景

流量复制（Traffic Mirror）功能应用于以下两个场景：

- 在系统进行重大重构或者发布新功能时，往往需要对系统进行压测以事先评估新系统的承载能力。传统情况下，在线下环境模拟线上各种测试数据来对新系统进行测试，但是这种方式往往并不能有效地模拟出线上真实的访问量，尤其是模拟正常流量中混杂着各色的异常流量。针对这种情况，可以通过将线上的应用流量复制到指定的线下环境来对新系统进行仿真测试。
- 线上系统遇到性能瓶颈，但是又不能快速地定位出问题时，这时可以采用流量复制的方式来将应用的真实流量引导到线下环境来进行问题定位。



步骤一：部署基础应用

1. 在K8s Product Cluster集群部署应用，并测试访问。

i. 在K8s Product Cluster集群中部署一个应用，并通过Ingress来对外暴露服务访问。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   # 当前配置为老版本镜像。
   - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
     imagePullPolicy: Always
     name: nginx
     ports:
     - containerPort: 80
       protocol: TCP
   restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: NodePort
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: nginx-ingress
spec:
 rules:
 # 这里配置使用了集群默认域名（您也可以使用自定义域名并作DNS解析）。
 - host: nginx.c37bf6b77bded43669ba2fb67448b****.cn-hangzhou.alicontainer.com
  http:
   paths:
   - path: /
     backend:
      service:
       name: nginx-service
       port:
        number: 80
     pathType: ImplementationSpecific
```

ii. 执行以下命令，查看该应用的Ingress配置。

```
kubectl get ing nginx-ingress
```

预期输出：

```
NAME      HOSTS                                    ADDRESS     PORTS   AGE
nginx-ingress  nginx.c37bf6b77bded43669ba2fb67448b****.cn-hangzhou.alicontainer.com  47.110.1**.**
80     8m
```

iii. 执行以下命令，测试访问应用域名。

```
curl http://nginx.c37bf6b77bded43669ba2fb67448b****.cn-hangzhou.alicontainer.com
```

2. 在K8s Stage Cluster部署一个相同应用，并测试访问。

i. 在K8s Stage Cluster中部署一个相同应用，同样通过Ingress来对外暴露服务访问。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   # 当前配置为新版本镜像。
   - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
     imagePullPolicy: Always
     name: nginx
     ports:
     - containerPort: 80
       protocol: TCP
   restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  app: nginx
 type: NodePort
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: nginx-ingress
spec:
 rules:
 # 这里配置使用了集群默认域名（您也可以使用自定义域名并作DNS解析）。
 - host: nginx.c41eb6ca34a3e49f7aea63b8bc9e8****.cn-beijing.alicontainer.com
  http:
   paths:
   - path: /
     backend:
      service:
       name: nginx-service
       port:
        number: 80
     pathType: ImplementationSpecific
```

ii. 执行以下命令，查看该应用的Ingress配置。

```
kubectl get ing nginx-ingress
```

预期输出：

```
NAME      HOSTS                                ADDRESS    PORTS   AGE
nginx-ingress  nginx.c41eb6ca34a3e49f7aea63b8bc9e8****.cn-beijing.alicontainer.com  39.106.***.*  80
1m
```

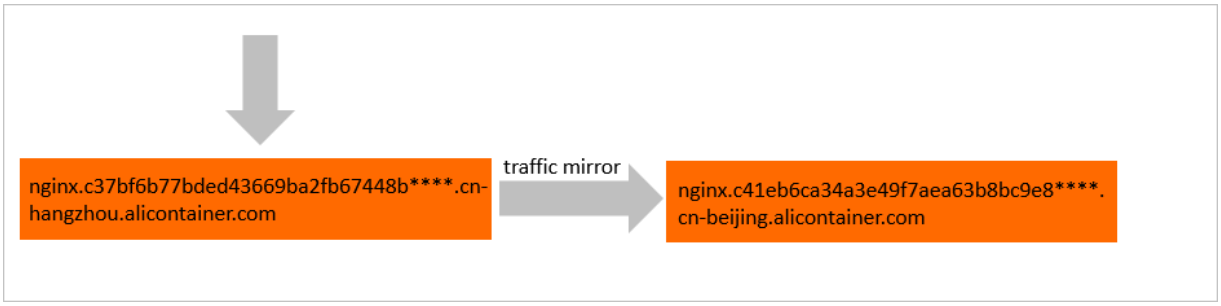
iii. 执行以下命令，测试访问应用域名。

```
curl http://nginx.c41eb6ca34a3e49f7aea63b8bc9e8****.cn-beijing.alicontainer.com
```

步骤二：配置流量复制

将K8s Product Cluster中该应用100%的访问流量复制到K8s Stage Cluster中对应的应用服务上，即将所有访问域名 `nginx.c37bf6b77bded43669ba2fb67448b****.cn-hangzhou.alicontainer.com` 的请求复制一份转发到 `nginx.c41eb6ca34a3e49f7aea63b8bc9e8****.cn-beijing.alicontainer.com` 。

> 说明
>
> - K8s Stage Cluster Ingress仅仅只作为复制流量的接收方，不需要做任何配置修改，只需要在K8s Product Cluster Ingress上进行配置即可。
> - 在将K8s Product Cluster中应用的访问流量复制到K8s Stage Cluster中对应的应用服务后，客户端只会收到K8s Product Cluster中的请求响应，K8s Stage Cluster中的请求响应会被丢弃。



1. 执行以下命令，编辑*nginx-configuration*，配置复制流量百分比和接收复制流量的应用域名。

```
kubectl -n kube-system edit cm nginx-configuration
```

在*nginx-configuration*中增加以下内容。

```
http-snippet: |
  split_clients "$date_gmt" $mirror_servers {
    100%   nginx.c41eb6ca34a3e49f7aea63b8bc9e8****.cn-beijing.alicontainer.com;
  }
# 配置说明:
# 1. 流量百分比取值范围：(0, 100]，百分比总和必须不大于100%。
# 2. 支持同时配置多个不同的复制流量接收目标应用。
```

2. 执行以下命令，编辑*nginx-ingress*。

通过configuration-snippet和server-snippet修改源Ingress，增加应用的流量复制配置。

```
kubectl  edit ingress nginx-ingress
```

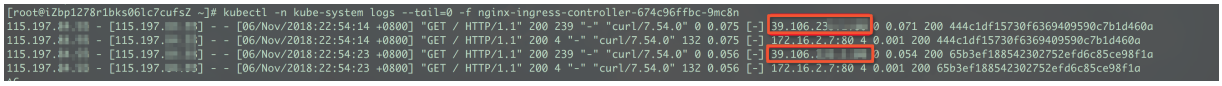
根据实际需求修改nginx-ingress的 `configuration-snippet` 和 `server-snippet` ，以下为本例修改后的YAML。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: nginx-ingress
 annotations:
  nginx.ingress.kubernetes.io/configuration-snippet: |
    mirror /mirror; # 配置N次该项则可放大N倍流量
  nginx.ingress.kubernetes.io/server-snippet: |
    location = /mirror {
      internal;
      # 不打印mirror请求日志
      #access_log off;
      # 设置proxy_upstream_name，格式必须为[Namespace]-[BackendServiceName]-[BackendServicePort]
      set $proxy_upstream_name   "default-nginx-service-80";
      # 自定义字符串，会作为请求头X-Shadow-Service值传给mirror server
      set $shadow_service_name   "nginx-product-service";
      proxy_set_header X-Shadow-Service  $shadow_service_name;
      proxy_set_header Host $mirror_servers;
      proxy_pass http://$mirror_servers$request_uri;
    }
spec:
 rules:
 - host: nginx.c37bf6b77bded43669ba2fb67448b****.cn-hangzhou.alicontainer.com
  http:
   paths:
   - path: /
    backend:
     service:
      name: nginx-service
      port:
       number: 80
    pathType: ImplementationSpecific
```

结果验证

访问K8s Product Cluster中的应用域名 `nginx.c37bf6b77bded43669ba2fb67448b****.cn-hangzhou.alicontainer.com` 。执行以下命令，可以看到每次请求K8s Product Cluster中的Ingress应用域名时，其都将复制一份同样的请求转发到K8s Stage Cluster中对应的应用服务中。

```
kubectl -n kube-system logs --tail-o -f nginx-ingress-controller-674c96ffbc-9mc8n
```

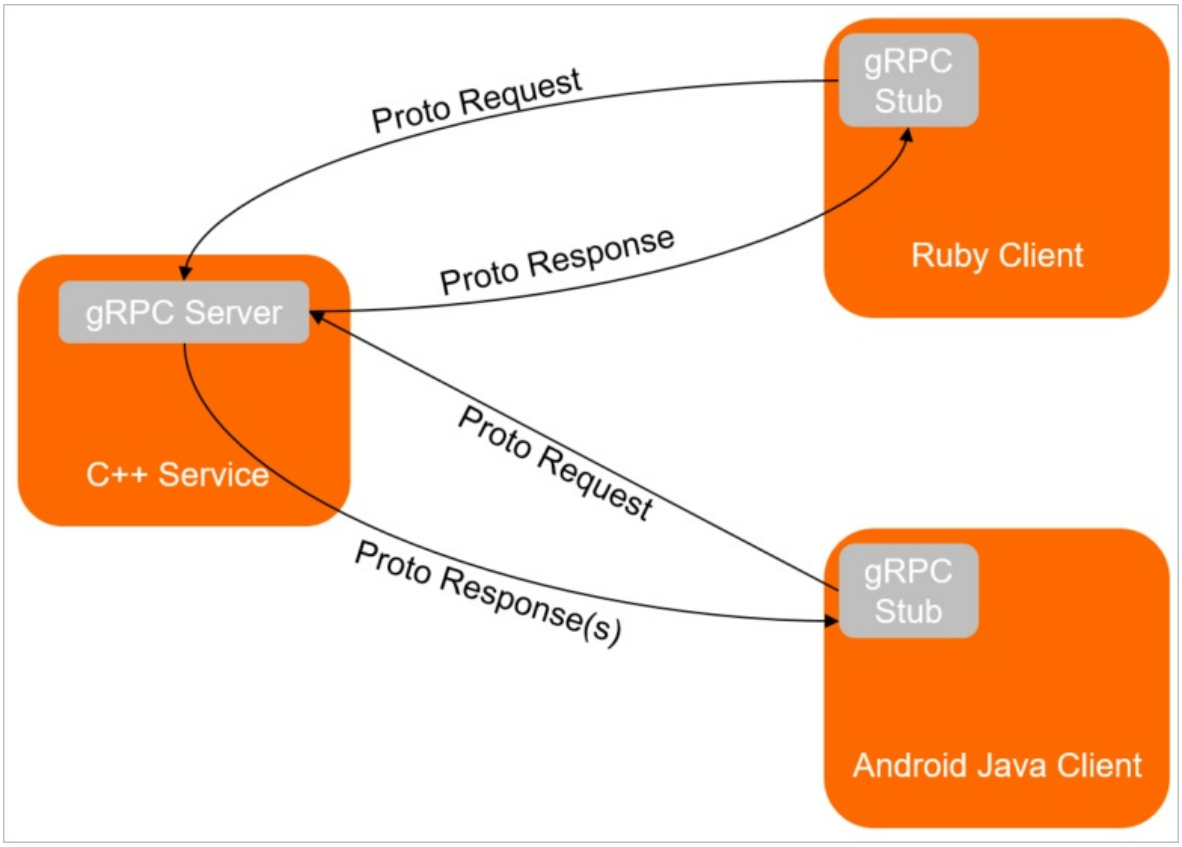


7.5.3.11. 通过Ingress Controller实现gRPC服务访问

远程过程调用gRPC（Google Remote Procedure Call）是基于HTTP/2协议标准和ProtoBuf（Protocol Buffers）序列化协议开发设计，且支持众多开发语言，继而提供了连接多路复用、头部压缩、流控等特性，极大地提高了客户端与服务端的通信效率。本文介绍如何通过阿里云容器服务Ingress Controller实现gRPC协议的服务后端的路由功能。

背景信息

gRPC是Google开源的一个高性能RPC通信框架，通过Protocol Buffers作为其IDL，在不同语言开发的平台上使用，同时gRPC基于HTTP/2协议实现，提供了多路复用、头部压缩、流控等特性，极大地提高了客户端与服务端的通信效率。



在gRPC服务中，客户端应用可以同本地方法一样调用到位于不同服务器上的服务端应用方法，可以很方便地创建分布式应用和服务。同其他RPC框架一样，gRPC也需要定义一个服务接口，同时指定被远程调用的方法和返回类型。服务端需要实现被定义的接口，同时运行一个gRPC服务器来处理客户端请求。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已安装Nginx Ingress Controller组件且其版本不低于0.22.0。关于Ingress Controller组件的版本信息，请参见Nginx Ingress Controller。
- 已使用kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。
- 已安装gRPCurl工具。具体操作，请参见gRPCurl。

gRPC服务示例

定义如下gRPC服务，客户端可调用helloworld.Greeter服务的SayHello接口：

```
option java_multiple_files = true;
option java_package = "io.grpc.examples.helloworld";
option java_outer_classname = "HelloWorldProto";
package helloworld;
// The greeting service definition.
service Greeter {
 // Sends a greeting
 rpc SayHello (HelloRequest) returns (HelloReply) {}
}
// The request message containing the user's name.
message HelloRequest {
 string name = 1;
}
// The response message containing the greetings
message HelloReply {
 string message = 1;
}
```

关于gRPC服务示例的源码，请参见gRPC Hello World。

示例说明

NGINX Ingress Controller中，gRPC服务只运行在HTTPS端口（默认443）上，因此在生产环境中，需要域名和对应的SSL证书。本示例使用 grpc.example.com 和自签SSL证书。

本示例基于1.20.4-aliyun集群，Nginx Ingress Controller组件版本为v0.44.0.3-8e83e7dc6-aliyun。

若您已有gRPC服务，可直接执行步骤三：创建Ingress路由规则。

步骤一：申请SSL证书

使用Ingress转发gRPC服务需要对应域名拥有SSL证书，使用TLS协议进行通信。

您可以在SSL证书控制台购买证书，也可以通过Let's Encrypt等第三方平台申请免费的SSL证书。

本示例使用OpenSSL生成的自签证书。

1. 复制以下内容并保存至*/tmp/openssl.cnf*文件中。

```
[ req ]
#default_bits       = 2048
#default_md         = sha256
#default_keyfile    = privkey.pem
distinguished_name    = req_distinguished_name
attributes          = req_attributes
req_extensions      = v3_req
[ req_distinguished_name ]
countryName             = Country Name (2 letter code)
countryName_min         = 2
countryName_max         = 2
stateOrProvinceName         = State or Province Name (full name)
localityName            = Locality Name (eg, city)
0.organizationName          = Organization Name (eg, company)
organizationalUnitName      = Organizational Unit Name (eg, section)
commonName              = Common Name (eg, fully qualified host name)
commonName_max          = 64
emailAddress            = Email Address
emailAddress_max        = 64
[ req_attributes ]
challengePassword           = A challenge password
challengePassword_min       = 4
challengePassword_max       = 20
[v3_req]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = grpc.example.com
```

2. 执行以下命令签署证书请求。

```
openssl req -new -nodes -keyout grpc.key -out grpc.csr -config /tmp/openssl.cnf -subj "/C=CN/ST=Zhejiang/L=Hangzhou/O=AlibabaCloud/OU=ContainerService/CN=grpc.example.com"
```

3. 执行以下命令签署证书。

```
openssl x509 -req -days 3650 -in grpc.csr -signkey grpc.key -out grpc.crt -extensions v3_req -extfile /tmp/openssl.cnf
```

命令执行成功后，可得到证书*grpc.crt*与私钥文件*grpc.key*。

4. 执行以下命令将名称为**grpc-secret**的TLS Secret添加到集群中。

或可通过控制台将TLS Secret添加到集群中。具体操作，请参见创建保密字典。

```
kubectl create secret tls grpc-secret --key grpc.key --cert grpc.crt
```

步骤二：创建gRPC服务所需资源

在集群中创建使用gRPC协议的后端服务。本示例使用镜像 `registry.cn-hangzhou.aliyuncs.com/acs-sample/grpc-server` 创建gRPC服务。

1. 复制以下YAML内容创建*grpc.yaml*文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: grpc-service
spec:
 replicas: 1
 selector:
  matchLabels:
   run: grpc-service
 template:
  metadata:
   labels:
    run: grpc-service
  spec:
   containers:
   - image: registry.cn-hangzhou.aliyuncs.com/acs-sample/grpc-server:latest
    imagePullPolicy: Always
    name: grpc-service
    ports:
    - containerPort: 50051
     protocol: TCP
   restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
 name: grpc-service
spec:
 ports:
 - port: 50051
  protocol: TCP
  targetPort: 50051
 selector:
  run: grpc-service
 sessionAffinity: None
 type: NodePort
```

2. 执行以下命令创建gRPC Deployment资源以及对应的服务。

```
kubectl apply -f grpc.yaml
```

预期输出：

```
deployment.apps/grpc-service created
service/grpc-service created
```

3. 执行以下命令查看Pod。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS       RESTARTS  AGE
grpc-service-57884679-2bkbr      1/1   Running      0     98s
```

从预期输出可得，grpc-service-57884679-2bkbr的状态为Running，已成功启动。

步骤三：创建Ingress路由规则

1. 复制以下YAML内容创建*grpc-ingress.yaml*文件。

   > **注意**
   > - 部署gRPC服务所使用的Ingress需要在annotation中加入 `nginx.ingress.kubernetes.io/backend-protocol` ，值为GRPC。
   > - 本示例使用的域名为 `grpc.example.com` ，请根据实际情况修改。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: grpc-ingress
 annotations:
  # 注意这里：必须要配置以指明后端服务为gRPC服务
  nginx.ingress.kubernetes.io/backend-protocol: "GRPC"
spec:
 # 指定证书
 tls:
 - hosts:
  - grpc.example.com
  secretName: grpc-secret
 rules:
 # gRPC服务域名
 - host: grpc.example.com
  http:
   paths:
   - path: /
    pathType: Prefix
    backend:
     service:
      # gRPC服务
      name: grpc-service
      port:
       number: 50051
```

2. 执行以下命令创建Ingress路由规则。

```
kubectl apply -f grpc-ingress.yaml
```

   预期输出：

```
ingress.networking.k8s.io/grpc-ingress created
```

结果验证

本地安装gRPCurl工具后，输入 `grpcurl <域名>:443 list` 验证请求是否成功转发到后端服务。

本示例中使用域名 `grpc.example.com` 以及自签证书，执行以下命令验证请求是否成功转发到后端服务。

```
grpcurl -insecure -authority grpc.example.com <ip_address>:443 list
```

> **说明** *ip_address*为Nginx Ingress Controller的Service外部IP，可通过 `kubectl get ingress` 获取。

预期输出：

```
grpc.reflection.v1alpha.ServerReflection
helloworld.Greeter
```

从预期输出可得，流量被Ingress成功转发到后端gRPC服务。

后续步骤

关于如何通过Ingress Controller实现gRPC服务的灰度发布，请参见通过Ingress实现灰度发布和蓝绿发布。

> 注意 由于Nginx grpc_pass的限制，目前对于gRPC服务，暂不支持service-weight的配置。

7.5.4. ALB Ingress管理

7.5.4.1. 通过ALB Ingress访问服务

ALB Ingress基于阿里云应用型负载均衡ALB（Application Load Balancer）实现的Ingress服务，适用于有明显波峰波谷的业务场景。本文介绍如何在ACK集群中使用ALB Ingress访问服务。

背景信息

Ingress是允许访问集群内Service的规则集合，您可以通过配置转发规则，实现不同URL访问集群内不同的Service。但传统的Nginx Ingress或者四层SLB Ingress，已无法满足云原生应用服务对复杂业务路由、多种应用层协议（例如：QUIC等）、大规模七层流量能力的需求。

前提条件

- ALB Ingress目前处于公测阶段。
- 已配置SNAT公网访问。
  - 若需要创建集群，请在**集群配置**配置向导页面，在**配置SNAT**字段右侧选中**为专有网络配置SNAT**。从而可以访问外网，下载容器镜像。
  - 若已创建集群，且未配置SNAT公网访问。关于如何为已有集群开启SNAT公网访问能力，请参见为已有集群开启SNAT公网访问能力。
- 已安装ALB Ingress Controller组件。
  - 若需要创建集群，请在**组件配置**配置向导页面，在**Ingress**字段右侧选择**ALB Ingress（公测）**。
  - 若已创建集群，且未安装ALB Ingress Controller组件。关于如何安装ALB Ingress Controller组件，请参见管理组件。
- 若需要在ACK专有版集群中通过ALB Ingress访问服务，在部署服务前需要为ALB Ingress Controller授权。具体操作，请参见为ACK专有版集群中的ALB Ingress Controller授权。
- 已通过kubectl工具连接ACK集群。具体操作，请参见通过kubectl工具连接集群。

注意事项

如果您使用的是Flannel网络插件，则ALB Ingress后端Service服务仅支持NodePort和LoadBalance类型。

步骤一：部署服务

1. 创建并拷贝以下内容到*cafe-service.yaml*文件中，用于部署两个名称分别为 `coffee` 和 `tea` 的Deployment，以及两个名称分别为 `coffee` 和 `tea` 的Service。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: coffee
spec:
 replicas: 2
```

```
 selector:
  matchLabels:
   app: coffee
 template:
  metadata:
   labels:
    app: coffee
  spec:
   containers:
   - name: coffee
    image: registry.cn-hangzhou.aliyuncs.com/acs-sample/nginxdemos:latest
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
 name: coffee-svc
spec:
 ports:
 - port: 80
  targetPort: 80
  protocol: TCP
 selector:
  app: coffee
 type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tea
spec:
 replicas: 1
 selector:
  matchLabels:
   app: tea
 template:
  metadata:
   labels:
    app: tea
  spec:
   containers:
   - name: tea
    image: registry.cn-hangzhou.aliyuncs.com/acs-sample/nginxdemos:latest
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
 name: tea-svc
 labels:
spec:
 ports:
 - port: 80
  targetPort: 80
  protocol: TCP
```

```
  selector:
    app: tea
  type: NodePort
```

2. 执行以下命令，部署两个Deployment和两个Service。

```
kubectl apply -f cafe-service.yaml
```

预期输出：

```
deployment "coffee" created
service "coffee-svc" created
deployment "tea" created
service "tea-svc" created
```

3. 执行以下命令，查看服务状态。

```
kubectl get svc,deploy
```

预期输出：

```
NAME       TYPE    CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
coffee-svc  NodePort  172.16.231.169  <none>    80:31124/TCP  6s
tea-svc   NodePort  172.16.38.182  <none>    80:32174/TCP  5s
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
deploy/coffee  2    2    2      2      1m
deploy/tea   1    1    1      1      1m
```

步骤二：配置Ingress

1. 创建并拷贝以下内容到*cafe-ingress.yaml*文件中。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: cafe-ingress
 annotations:
  kubernetes.io/ingress.class: alb
  alb.ingress.kubernetes.io/name: ingress_test_base
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****"
spec:
 rules:
 - http:
    paths:
    # 配置Context Path。
    - path: /tea
      backend:
       serviceName: tea-svc
       servicePort: 80
    # 配置Context Path。
    - path: /coffee
      backend:
       serviceName: coffee-svc
       servicePort: 80
```

相关参数解释如下表所示：

参数	说明
（可选）alb.ingress.kubernetes.io/name	表示ALB实例名称。
（可选）alb.ingress.kubernetes.io/address-type	表示负载均衡的地址类型。取值如下： ○ Internet（默认值）：负载均衡具有公网IP地址，DNS域名被解析到公网IP，因此可以在公网环境访问。 ○ Intranet：负载均衡只有私网IP地址，DNS域名被解析到私网IP，因此只能被负载均衡所在VPC的内网环境访问。
（必选）alb.ingress.kubernetes.io/vswitch-ids	用于设置ALB Ingress交换机ID，您需要至少指定两个不同可用区交换机ID。关于ALB Ingress支持的地域与可用区，请参见支持的地域与可用区。

2. 执行以下命令，配置 `coffee` 和 `tea` 服务对外暴露的域名和 `path` 路径。

```
kubectl apply -f cafe-ingress.yaml
```

预期输出：

```
ingress "cafe-ingress" created
```

3. 执行以下命令获取ALB实例IP地址。

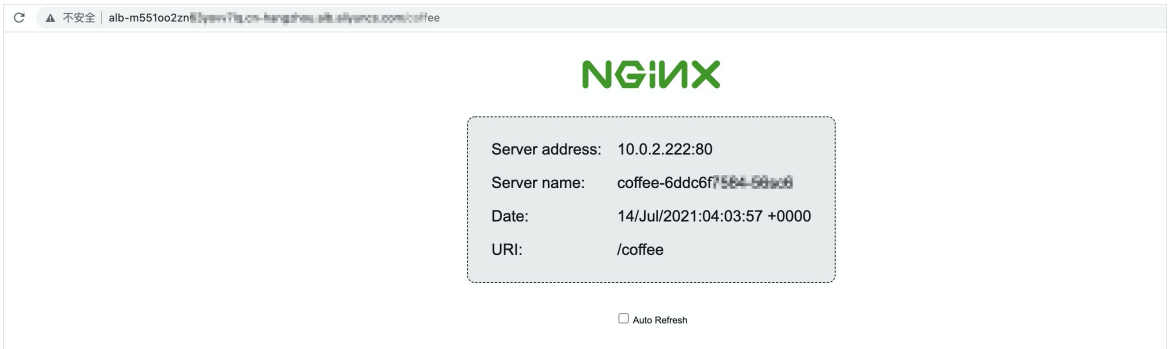
```
kubectl get ing
```

预期输出：

```
NAME      CLASS   HOSTS  ADDRESS                          PORTS  AGE
cafe-ingress  <none>  *    alb-m551oo2zn63yov****.cn-hangzhou.alb.aliyuncs.com  80   50s
```

步骤三：访问服务

- 利用获取的ALB实例IP地址，通过以下两种方式访问 `coffee` 服务：
  - 通过浏览器访问 `coffee` 服务。

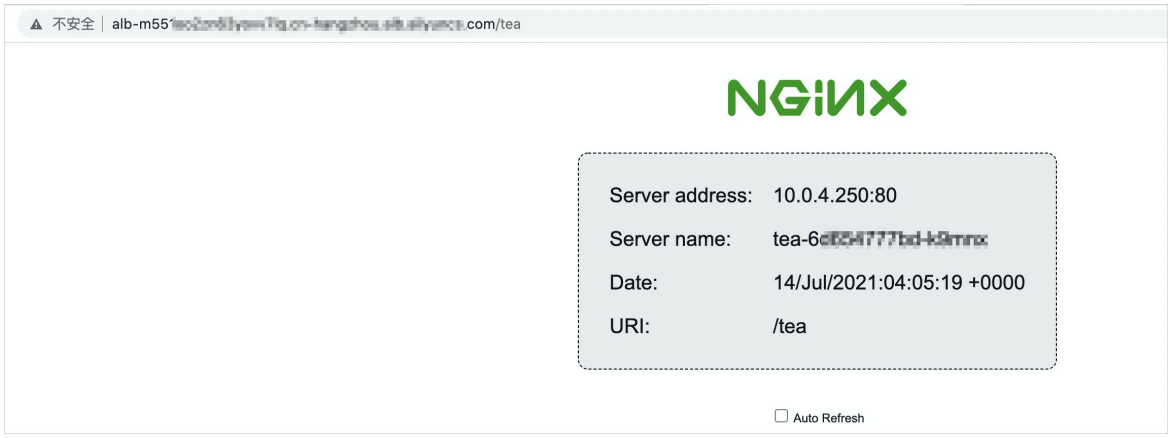


  - 通过命令行方式访问 `coffee` 服务。

```
curl http://alb-m551oo2zn63yov****.cn-hangzhou.alb.aliyuncs.com/coffee
```

- 利用获取的ALB实例IP地址，通过以下两种方式访问 `tea` 服务：

- 通过浏览器访问 tea 服务。



- 通过命令行方式访问 tea 服务。

```
curl http://alb-m551oo2zn63yov****.cn-hangzhou.alb.aliyuncs.com/tea
```

7.5.4.2. ALB Ingress服务高级用法

在Kubernetes集群中，ALB Ingress对集群服务（Service）中外部可访问的API对象进行管理，提供七层负载均衡能力。本文介绍如何使用ALB Ingress将来自不同域名或URL路径的请求转发给不同的后端服务器组、将HTTP访问重定向至HTTPS及实现灰度发布等功能。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已配置SNAT公网访问。
  - 若需要创建集群，请在**集群配置**配置向导页面，在**配置SNAT**字段右侧选中**为专有网络配置SNAT**。从而可以访问外网，下载容器镜像。
  - 若已创建集群，且未配置SNAT公网访问。关于如何为已有集群开启SNAT公网访问能力，请参见为已有集群开启SNAT公网访问能力。
- 已通过kubectl工具连接ACK集群。具体操作，请参见通过kubectl工具连接集群。

基于域名转发请求

通过以下命令创建一个简单的Ingress，根据指定的正常域名或空域名转发请求。

- 基于正常域名转发请求的示例如下：
  i. 部署以下模板，分别创建Service、Deployment和Ingress，将访问请求通过Ingress的域名转发至Service。

```
apiVersion: v1
kind: Service
metadata:
 name: demo-service
 namespace: default
spec:
 ports:
  - name: port1
    port: 80
    protocol: TCP
    targetPort: 8080
 selector:
  app: demo
 sessionAffinity: None
```

```
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo
 namespace: default
spec:
 replicas: 1
 selector:
  matchLabels:
   app: demo
 template:
  metadata:
   labels:
    app: demo
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/alb-sample/cafe:v1
      imagePullPolicy: IfNotPresent
      name: demo
      ports:
       - containerPort: 8080
         protocol: TCP
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 annotations:
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-2zeqgkyib34gw1fxs****,vsw-2zefv5qwao4przzlo****"
  kubernetes.io/ingress.class: alb
 name: demo
 namespace: default
spec:
 rules:
  - host: demo.domain.ingress.top
   http:
    paths:
     - backend:
        serviceName: demo-service
        servicePort: 80
       path: /hello
       pathType: ImplementationSpecific
```

ii. 执行以下命令，通过指定的正常域名访问服务。
替换ADDRESS为ALB实例对应的域名地址，可通过 `kubectl get ing` 获取。

```
curl -H "host: demo.domain.ingress.top" <ADDRESS>/hello
```

预期输出：

```
{"hello":"coffee"}
```

- 基于空域名转发请求的示例如下：
  i. 部署以下模板，创建Ingress。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 annotations:
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-2zeqgkyib34gw1fxs****,vsw-2zefv5qwao4przzlo****"
  kubernetes.io/ingress.class: alb
 name: demo
 namespace: default
spec:
 rules:
  - host: ""
    http:
     paths:
      - backend:
         serviceName: demo-service
         servicePort: 80
        path: /hello
        pathType: ImplementationSpecific
```

ii. 执行以下命令，通过空域名访问服务。
替换ADDRESS为ALB实例对应的域名地址，可通过 `kubectl get ing` 获取。

```
curl <ADDRESS>/hello
```

预期输出：

```
{"hello":"coffee"}
```

基于URL路径转发请求

ALB Ingress支持按照URL转发请求，可以通过 `pathType` 字段设置不同的URL匹配策略。 `pathType` 支持Exact、ImplementationSpecific和Prefix三种匹配方式。

三种匹配方式的示例如下：

- Exact：以区分大小写的方式精确匹配URL路径。

  i. 部署以下模板，创建Ingress。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 annotations:
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-2zeqgkyib34gw1fxs****,vsw-2zefv5qwao4przzlo****"
  kubernetes.io/ingress.class: alb
 name: demo-path
 namespace: default
spec:
 rules:
  - http:
     paths:
     - path: /hello
       backend:
        serviceName: demo-service
        servicePort: 80
       pathType: Exact
```

  ii. 执行以下命令，访问服务。
  替换ADDRESS为ALB实例对应的域名地址，可通过 `kubectl get ing` 获取。

```
curl <ADDRESS>/hello
```

预期输出：

```
{"hello":"coffee"}
```

- ImplementationSpecific：缺省。在ALB Ingress中与 `Exact` 做相同处理，但两者Ingress Controller的实现方式不一样。
  i. 部署以下模板，创建Ingress。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 annotations:
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-2zeqgkyib34gw1fxs****,vsw-2zefv5qwao4przzlo****"
  kubernetes.io/ingress.class: alb
 name: demo-path
 namespace: default
spec:
 rules:
  - http:
     paths:
     - path: /hello
       backend:
        serviceName: demo-service
        servicePort: 80
       pathType: ImplementationSpecific
```

  ii. 执行以下命令，访问服务。
  替换**ADDRESS**为ALB实例对应的域名地址，可通过 `kubectl get ing` 获取。

```
curl <ADDRESS>/hello
```

预期输出：

```
{"hello":"coffee"}
```

- Prefix：以 `/` 分隔的URL路径进行前缀匹配。匹配区分大小写，并且对路径中的元素逐个完成匹配。
  i. 部署以下模板，创建Ingress。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 annotations:
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-2zeqgkyib34gw1fxs****,vsw-2zefv5qwao4przzlo****"
  kubernetes.io/ingress.class: alb
 name: demo-path-prefix
 namespace: default
spec:
 rules:
  - http:
     paths:
     - path: /
       backend:
        serviceName: demo-service
        servicePort: 80
       pathType: Prefix
```

ii. 执行以下命令，访问服务。
替换**ADDRESS**为ALB实例对应的域名地址，可通过 `kubectl get ing` 获取。

```
curl <ADDRESS>/hello
```

预期输出：

```
{"hello":"coffee"}
```

配置健康检查

ALB Ingress支持配置健康检查，可以通过设置以下注解实现。

配置健康检查的YAML示例如下所示：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: cafe-ingress
 annotations:
  kubernetes.io/ingress.class: alb
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****"
  alb.ingress.kubernetes.io/healthcheck-enabled: "true"
  alb.ingress.kubernetes.io/healthcheck-path: "/"
  alb.ingress.kubernetes.io/healthcheck-protocol: "HTTP"
  alb.ingress.kubernetes.io/healthcheck-method: "HEAD"
  alb.ingress.kubernetes.io/healthcheck-httpcode: "http_2xx"
  alb.ingress.kubernetes.io/healthcheck-timeout-seconds: "5"
  alb.ingress.kubernetes.io/healthcheck-interval-seconds: "2"
  alb.ingress.kubernetes.io/healthy-threshold-count: "3"
  alb.ingress.kubernetes.io/unhealthy-threshold-count: "3"
spec:
 rules:
 - http:
    paths:
    # 配置Context Path。
    - path: /tea
      backend:
        serviceName: tea-svc
        servicePort: 80
    # 配置Context Path。
    - path: /coffee
      backend:
        serviceName: coffee-svc
        servicePort: 80
```

相关参数解释如下表所示。

参数	说明
alb.ingress.kubernetes.io/healthcheck-enabled	（可选）表示是否开启健康检查。默认开启（true）。
alb.ingress.kubernetes.io/healthcheck-path	（可选）表示健康检查路径。默认/。 • 输入健康检查页面的URL，建议对静态页面进行检查。长度限制为1~80个字符，支持使用字母、数字和短划线（-）、正斜线（/）、半角句号（.）、百分号（%）、半角问号（?）、井号（#）和and（&）以及扩展字符集_;~!()*[]@$^:',+。URL必须以正斜线（/）开头。 • HTTP健康检查默认由负载均衡系统通过后端ECS内网IP地址向该服务器应用配置的默认首页发起HTTP Head请求。如果您用来进行健康检查的页面并不是应用服务器的默认首页，需要指定具体的检查路径。

参数	说明
alb.ingress.kubernetes.io/healthcheck-protocol	（可选）表示健康检查协议。 • HTTP（默认）：通过发送HEAD或GET请求模拟浏览器的访问行为来检查服务器应用是否健康。 • TCP：通过发送SYN握手报文来检测服务器端口是否存活。 • GRPC：通过发送POST或GET请求来检查服务器应用是否健康。
alb.ingress.kubernetes.io/healthcheck-method	（可选）选择一种健康检查方法。 • HEAD（默认）：HTTP监听健康检查默认采用HEAD方法。请确保您的后端服务器支持HEAD请求。如果您的后端应用服务器不支持HEAD方法或HEAD方法被禁用，则可能会出现健康检查失败，此时可以使用GET方法来进行健康检查。 • POST：GRPC监听健康检查默认采用POST方法。请确保您的后端服务器支持POST请求。如果您的后端应用服务器不支持POST方法或POST方法被禁用，则可能会出现健康检查失败，此时可以使用GET方法来进行健康检查。 • GET：如果响应报文长度超过8 KB，会被截断，但不会影响健康检查结果的判定。
alb.ingress.kubernetes.io/healthcheck-httpcode	设置健康检查正常的状态码。 • 当健康检查协议为HTTP协议时，可以选择http_2xx（默认）、http_3xx、http_4xx和http_5xx。 • 当健康检查协议为GRPC协议时，状态码范围为0~99。支持范围输入，最多支持20个范围值，多个范围值使用半角逗号（,）隔开。
alb.ingress.kubernetes.io/healthcheck-timeout-seconds	表示接收健康检查的响应需要等待的时间。如果后端ECS在指定的时间内没有正确响应，则判定为健康检查失败。时间范围为1~300秒，默认值为5秒。
alb.ingress.kubernetes.io/healthcheck-interval-seconds	健康检查的时间间隔。取值范围1~50秒，默认为2秒。
alb.ingress.kubernetes.io/healthy-threshold-count	表示健康检查连续成功所设置的次数后会将后端服务器的健康检查状态由失败判定为成功。取值范围2~10，默认为3次。
alb.ingress.kubernetes.io/unhealthy-threshold-count	表示健康检查连续失败所设置的次数后会将后端服务器的健康检查状态由成功判定为失败。取值范围2~10，默认为3次。

配置自动发现HTTPS证书功能

ALB Ingress Controller提供证书自动发现功能。您需要首先在SSL证书控制台创建证书，然后ALB Ingress Controller会根据Ingress中TLS配置的域名自动匹配发现证书。

1. 执行以下命令，通过 `openssl` 创建证书。

```
openssl genrsa -out albtop-key.pem 4096
openssl req -subj "/CN=demo.alb.ingress.top" -sha256 -new -key albtop-key.pem -out albtop.csr
echo subjectAltName = DNS:demo.alb.ingress.top > extfile.cnf
openssl x509 -req -days 3650 -sha256 -in albtop.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out albtop-cert.pem -extfile extfile.cnf
```

2. 在SSL证书控制台上传证书。

   具体操作，请参见上传证书。

3. 在Ingress的YAML中添加以下命令，配置该证书对应的域名。

```
tls:
 - hosts:
  - demo.alb.ingress.top
```

   示例如下：

```
---
apiVersion: v1
kind: Service
metadata:
 name: demo-service-https
 namespace: default
spec:
 ports:
  - name: port1
    port: 443
    protocol: TCP
    targetPort: 8080
 selector:
  app: demo-cafe
 sessionAffinity: None
 type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo-cafe
 namespace: default
spec:
 replicas: 1
 selector:
  matchLabels:
   app: demo-cafe
 template:
  metadata:
   labels:
    app: demo-cafe
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/alb-sample/cafe:v1
      imagePullPolicy: IfNotPresent
      name: demo-cafe
      ports:
       - containerPort: 8080
         protocol: TCP
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 annotations:
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****"
  kubernetes.io/ingress.class: alb
```

```
  name: demo-https
  namespace: default
 spec:
  #配置证书对应的域名。
  tls:
  - hosts:
   - demo.alb.ingress.top
  rules:
   - host: demo.alb.ingress.top
     http:
      paths:
       - backend:
          serviceName: demo-service-https
          servicePort: 443
         path: /
         pathType: Prefix
```

4. 执行以下命令，查看证书。

```
curl https://demo.alb.ingress.top/tea
```

预期输出：

```
{"hello":"tee"}
```

配置HTTP重定向至HTTPS

ALB Ingress通过设置注解 `alb.ingress.kubernetes.io/ssl-redirect: "true"` ，可以将HTTP请求重定向到HTTPS 443端口。

配置示例如下：

```
---
apiVersion: v1
kind: Service
metadata:
 name: demo-service-ssl
 namespace: default
spec:
 ports:
  - name: port1
    port: 80
    protocol: TCP
    targetPort: 8080
 selector:
  app: demo-ssl
 sessionAffinity: None
 type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo-ssl
 namespace: default
spec:
 replicas: 1
 selector:
  matchLabels:
   app: demo-ssl
 template:
```

```
  metadata:
    labels:
      app: demo-ssl
  spec:
    containers:
      - image: registry.cn-hangzhou.aliyuncs.com/alb-sample/cafe:v1
        imagePullPolicy: IfNotPresent
        name: demo-ssl
        ports:
          - containerPort: 8080
            protocol: TCP
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****"
    alb.ingress.kubernetes.io/ssl-redirect: "true"
  name: demo-ssl
  namespace: default
spec:
  tls:
  - hosts:
    - ssl.alb.ingress.top
  rules:
    - host: ssl.alb.ingress.top
      http:
        paths:
          - backend:
              serviceName: demo-service-ssl
              servicePort: 80
            path: /
            pathType: Prefix
```

通过注解实现灰度发布

ALB提供复杂路由处理能力，支持基于Header、Cookie以及权重的灰度发布功能。灰度发布功能可以通过设置注解来实现，为了启用灰度发布功能，需要设置注解 `alb.ingress.kubernetes.io/canary: "true"` ，通过不同注解可以实现不同的灰度发布功能。

> **说明** 灰度优先级顺序：基于Header>基于Cookie>基于权重（从高到低）。

参数	说明	示例

参数	说明	示例
alb.ingress.kubernetes.io/canary-by-header 和 alb.ingress.kubernetes.io/canary-by-header-value	匹配的Request Header的值，该规则允许您自定义Request Header的值，但必须与 alb.ingress.kubernetes.io/canary-by-header 一起使用。 • 当请求中的 header 和 header-value 与设置的值匹配时，请求流量会被分配到灰度服务入口。 • 对于其他 header 值，将会忽略 header，并通过灰度优先级将请求流量分配到其他规则设置的灰度服务。	当请求Header为 location: hz 时将访问灰度服务；其它Header将根据灰度权重将流量分配给灰度服务。 apiVersion: networking.k8s.io/v1beta1 kind: Ingress metadata: annotations: kubernetes.io/ingress.class: alb alb.ingress.kubernetes.io/albconfig.order: "1" alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****" alb.ingress.kubernetes.io/canary: "true" alb.ingress.kubernetes.io/canary-by-header: "location" alb.ingress.kubernetes.io/canary-by-header-value: "hz"

参数	说明	示例
alb.ingress.kubernetes.io/canary-by-cookie	基于Cookie的流量切分： • 当配置的 cookie 值为 always 时，请求流量将被分配到灰度服务入口。 • 当配置的 cookie 值为 never 时，请求流量将不会分配到灰度服务入口。 说明 基于Cookie的灰度不支持设置自定义，只有 always 和 never 。	请求的Cookie为 demo=always 时将访问灰度服务。 apiVersion: networking.k8s.io/v1beta1 kind: Ingress metadata: annotations: kubernetes.io/ingress.class: alb alb.ingress.kubernetes.io/albconfig.order: "2" alb.ingress.kubernetes.io/address-type: internet alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****" alb.ingress.kubernetes.io/canary: "true" alb.ingress.kubernetes.io/canary-by-cookie: "demo"

参数	说明	示例
alb.ingress.kubernetes.io/canary-weight	设置请求到指定服务的百分比（值为0~100的整数）。	配置灰度服务的权重为50%。 apiVersion: networking.k8s.io/v1beta1 kind: Ingress metadata: annotations: kubernetes.io/ingress.class: alb alb.ingress.kubernetes.io/albconfig.order: "3" alb.ingress.kubernetes.io/address-type: internet alb.ingress.kubernetes.io/vswitch-ids: "vsw-2zeqgkyib34gw1fxs****,vsw-2zefv5qwao4przzlo****" alb.ingress.kubernetes.io/canary: "true" alb.ingress.kubernetes.io/canary-weight: "50"

7.5.4.3. 为ACK专有版集群中的ALB Ingress Controller授权

若需要在ACK专有版集群中通过ALB Ingress访问服务，部署服务前需要对云产品授予ALB Ingress Controller的相关权限。本文介绍如何对ACK专有版集群中的ALB Ingress Controller授权。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面上方单击**集群资源**页签。
5. 在**集群资源**页签单击**Worker RAM角色**右侧的**K8sWorkerRole-******。



6. 在**RAM控制台**确认并修改信任策略和RAM权限策略。
   i. 在**K8sWorkerRole-******页面，单击**信任策略管理**页签。
   ii. 查看当前信任策略是否与以下内容一致，若不一致，请单击**修改信任策略**，在**修改信任策略**面板修改为以下内容后单击**确定**。

```
{
 "Statement": [
  {
   "Action": "sts:AssumeRole",
   "Effect": "Allow",
   "Principal": {
    "Service": [
     "ecs.aliyuncs.com"
    ]
   }
  }
 ],
 "Version": "1"
}
```

   iii. 在**K8sWorkerRole-******页面，单击**权限管理**页签，然后单击**K8sWorkerRolePolicy-******权限策略名称。
   iv. 在**权限策略管理**页面，查看是否包含以下ALB Ingress权限。若不完全包含，请单击**修改策略内容**。在**修改策略内容**面板添加以下内容后，单击**确定**。

```
{
      "Action": [
        "alb:TagResources",
        "alb:ListServerGroups",
        "alb:ListServerGroupServers",
        "alb:AddServersToServerGroup",
        "alb:RemoveServersFromServerGroup",
        "alb:ReplaceServersInServerGroup",
        "alb:CreateLoadBalancer",
        "alb:DeleteLoadBalancer",
        "alb:UpdateLoadBalancerAttribute",
        "alb:UpdateLoadBalancerEdition",
        "alb:EnableLoadBalancerAccessLog",
        "alb:DisableLoadBalancerAccessLog",
        "alb:EnableDeletionProtection",
        "alb:DisableDeletionProtection",
        "alb:ListLoadBalancers",
```

```
        "alb:GetLoadBalancerAttribute",
        "alb:ListListeners",
        "alb:CreateListener",
        "alb:GetListenerAttribute",
        "alb:UpdateListenerAttribute",
        "alb:ListListenerCertificates",
        "alb:AssociateAdditionalCertificatesWithListener",
        "alb:DissociateAdditionalCertificatesFromListener",
        "alb:DeleteListener",
        "alb:CreateRule",
        "alb:DeleteRule",
        "alb:UpdateRuleAttribute",
        "alb:CreateRules",
        "alb:UpdateRulesAttribute",
        "alb:DeleteRules",
        "alb:ListRules",
        "alb:CreateServerGroup",
        "alb:DeleteServerGroup",
        "alb:UpdateServerGroupAttribute",
        "alb:DescribeZones"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:CreateServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": [
            "alb.aliyuncs.com",
            "logdelivery.alb.aliyuncs.com"
          ]
        }
      }
    },
    {
      "Action": [
        "yundun-cert:DescribeSSLCertificateList",
        "yundun-cert:DescribeSSLCertificatePublicKeyDetail"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
```

> **说明** 多个策略内容需要用半角逗号（,）分隔。

7. 确认ECS实例的RAM角色状态是否正常。
   i. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   ii. 在**节点**页面，单击目标节点的实例ID，例如**i-2ze5d2qi9iy90pzb******。

iii. 在目标实例页面，单击**实例详情**页签，然后在**其它信息**区域的**RAM角色**右侧查看是否存在该ECS实例的RAM角色。

若没有对应的RAM角色，请为ECS实例指定RAM角色。具体操作，请参见步骤二：指定RAM角色创建并设置ECS实例。

8. 手动删除alb-ingress-controller的Pod并确认重建后Pod的状态。

i. 执行以下命令删除alb-ingress-controller的Pod。

```
kubectl -n kube-system delete pod alb-ingress-controller-xxx
```

预期输出：

```
pod "alb-ingress-controller-***1" deleted
```

ii. 等待几分钟后，执行以下命令查看重建后Pod的状态。

```
kubectl -n kube-system get pod
```

预期输出：

```
NAME                          READY  STATUS   RESTARTS  AGE
alb-ingress-controller-***2  1/1    Running  0      60s
```

从预期输出可得，重建后Pod **alb-ingress-controller-***2**的状态为**Running**。

后续步骤

关于如何在ACK专有版集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。

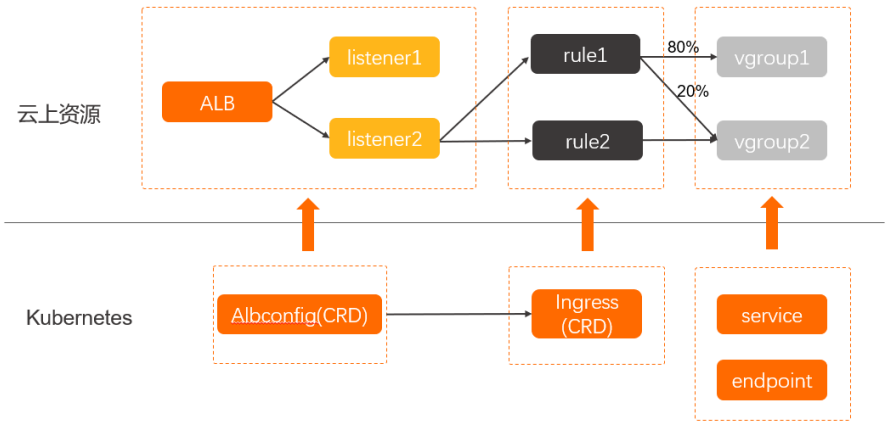
7.5.4.4. 配置Albconfig

Albconfig是由ALB Ingress Controller提供的CRD资源，ALB Ingress Controller使用Albconfig来配置ALB实例和监听。本文介绍如何创建、修改Albconfig以及开启日志服务等操作。

背景信息

ALB Ingress Controller通过API Server获取Ingress资源的变化，动态地生成Albconfig，然后依次创建ALB实例、监听、路由转发规则以及后端服务器组。Kubernetes中Service、Ingress与Albconfig有着以下关系：

- Service是后端真实服务的抽象，一个Service可以代表多个相同的后端服务。
- Ingress是反向代理规则，用来规定HTTP/HTTPS请求应该被转发到哪个Service上。例如：根据请求中不同的Host和URL路径，让请求转发到不同的Service上。
- Albconfig是在ALB Ingress Controller提供的CRD资源，使用ALBConfig CRD来配置ALB实例和监听。一个Albconfig对应一个ALB实例。



一个Albconfig对应一个ALB实例，如果一个ALB实例配置多个转发规则，那么一个Albconfig则对应多个Ingress，所以Albconfig与Ingress是一对多的对应关系。

创建Albconfig

创建Ingress时，会默认在kube-system命名空间下创建名称为default的Albconfig，无需您手动创建。

1. 部署以下模板，创建Ingress和Albconfig。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: cafe-ingress
 annotations:
  kubernetes.io/ingress.class: alb
  alb.ingress.kubernetes.io/address-type: internet
  alb.ingress.kubernetes.io/vswitch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****"
spec:
 rules:
 - http:
    paths:
    # 配置Context Path
    - path: /tea
     backend:
      serviceName: tea-svc
      servicePort: 80
    # 配置Context Path
    - path: /coffee
     backend:
      serviceName: coffee-svc
      servicePort: 80
```

2. 执行以下命令，查看Albconfig名称。

```
kubectl -n kube-system get albconfig
```

预期输出：

```
NAME    AGE
default  87m
```

Albconfig默认配置的内容如下：

```
apiVersion: alibabacloud.com/v1
kind: AlbConfig
metadata:
 name: default               #Albconfig名称。
 namespace: kube-system      #Albconfig所属命名空间。
spec:
 config:
  accessLogConfig:
   logProject: ""
   logStore: ""
  addressAllocatedMode: Dynamic
  addressType: Internet
  billingConfig:
   internetBandwidth: 0
   internetChargeType: ""
   payType: PostPay
  deletionProtectionEnabled: true
  edition: Standard
  forceOverride: false
  zoneMappings:
  - vSwitchId: vsw-wz92lvykqj1siwvif****     #Albconfig的vSwitch，Albconfig需要配置两个vSwitch。
  - vSwitchId: vsw-wz9mnucx78c7i6iog****     #Albconfig的vSwitch。
status:
 loadBalancer:
  dnsname: alb-s2em8fr9debkg5****.cn-shenzhen.alb.aliyuncs.com
  id: alb-s2em8fr9debkg5****
```

修改Albconfig的名称

如果您需要修改Albconfig的名称，可以执行以下命令。保存之后，新名称自动生效。

```
kubectl -n kube-system edit albconfig default
...
 spec:
  config:
   name: basic  #输入修改后的名称。
...
```

修改Albconfig的vSwitch配置

如果您需要修改Albconfig的vSwitch配置。保存之后，新配置自动生效。

```
kubectl -n kube-system edit albconfig default
...
 zoneMappings:
  - vSwitchId: vsw-wz92lvykqj1siwvif****
  - vSwitchId: vsw-wz9mnucx78c7i6iog****
...
```

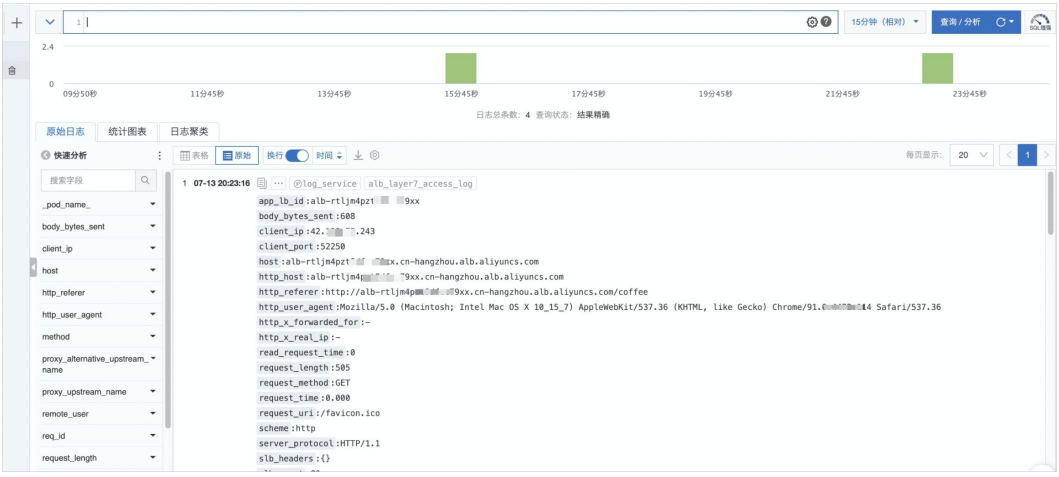
开启日志服务访问日志

如果您希望ALB Ingress能够收集访问日志Access Log，则只需要在AlbConfig中指定 `logProject` 和 `logStore` 。

```
apiVersion: alibabacloud.com/v1
kind: AlbConfig
metadata:
 name: default
 namespace: kube-system
spec:
 config:
  accessLogConfig:
   logProject: "k8s-log-xz92lvykqj1siwvif****"
   logStore: "alb_xxx"
  ...
```

> 说明 logStore命名需要以 alb_ 开头，若指定logStore不存在，系统则会自动创建。

保存命令之后，可以在日志服务控制台，单击目标Logstore，查看收集的访问日志。



删除ALB实例

一个ALB实例对应一个Albconfig， 因此可以通过删除Albconfig实现删除ALB实例，但前提是先需要删除Albconfig关联的所有Ingress。

```
kubectl -n kube-system delete albconfig default
```

default 可以替换为您实际需要删除的Albconfig。

7.5.4.5. 为ALB Ingress开启WAF防护暂时只支持中国站

Web应用防火墙WAF（Web Application Firewall）可以为网站或App提供一站式安全防护，防止数据泄密和恶意CC攻击，阻止木马上传和网页篡改，提供虚拟补丁。当ALB Ingress开启WAF防护后，ALB实例监听的端口流量将受到Web应用安全防护。本文介绍如何为ALB Ingress开启WAF防护。

前提条件

- 创建公网类型的ALB实例，且已通过透明接入的方式接入WAF。具体操作，请参见创建应用型负载均衡和透明接入 。
- 已安装ALB Ingress Controller， 并创建Ingress访问规则。具体操作，请参见通过ALB Ingress访问服务。

背景信息

- 目前支持为ALB Ingress开启WAF防护的地域包括：西南1（成都）、华北2（北京）、华东1（杭州）、华东2（上海）、华南1（深圳）。

- WAF提供CNAME接入和透明接入两种方式。在透明接入模式下，无需修改域名DNS解析、设置源站保护，同时无需改变服务器获取真实源链接IP的方式，保护您的Web业务正常运转。关于WAF的更多信息，请参见什么是Web应用防火墙。

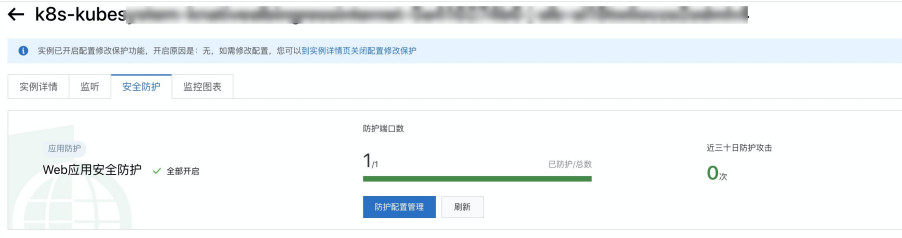
操作步骤

1. 登录应用型负载均衡ALB控制台。
2. 在左侧导航栏，选择**应用型负载均衡ALB > 实例**。
3. 在**实例**页面，单击目标ALB的实例ID。

   若需要查找目标ALB的实例ID，可以在kubectl中执行 `kubectl -n kube-system describe albconfig` 命令查看。具体操作，请参见ALB Ingress服务高级用法。
4. 单击**安全防护**页签，然后在**Web应用安全防护**区域，单击**全部开启**。
5. 在**开通Web应用安全防护能力**对话框，单击**确认开启**。

执行结果

开启WAF安全防护之后，在**安全防护**页签会显示防护的端口数等信息。



7.5.5. Ingress FAQ

本文主要为您介绍Ingress常见问题的处理方法。

- Ingress支持哪些SSL/TLS版本？
- Ingress L7请求头默认是透传的吗？
- 后端服务为HTTPS服务访问时是否可以通过Ingress-Nginx转发？
- Ingress L7透传客户端IP吗？
- nginx-ingress-controller组件支持HSTS吗？
- Ingress-Nginx支持哪些Rewrite配置？
- 当版本升级后SLS解析日志不正常怎样修复？
- 部署Ingress Controller使用私网SLB
- 在ACK组件管理中升级Nginx Ingress Controller组件时，系统所做的更新是什么？
- 如何将Ingress-nginx的监听由四层改为七层（HTTPS/HTTP）？
- ingress-nginx如何使用已有的SLB?
- 创建Ingress时报错“certificate signed by unknown authority”

Ingress支持哪些SSL/TLS版本？

Ingress-Nginx默认仅支持TLS V1.2版本，对于部分旧版本的浏览器，或者移动客户端TLS版本低于1.2时，会导致客户端在与Ingress-Nginx服务SSL版本协商时报错。

执行以下命令，修改 `kube-system/nginx-configuration configmap` 添加以下配置，为Ingress-Nginx开启支持更多TLS版本的功能。具体操作，请参见TLS/HTTPS。

```
ssl-ciphers: "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA"
ssl-protocols: "TLSv1 TLSv1.1 TLSv1.2 TLSv1.3"
```

Ingress L7请求头默认是透传的吗？

Ingress-Nginx默认透传客户端的请求头，有些不符合HTTP规则的请求头（例如Mobile Version），在转发到后端服务前会被过滤掉。为了不过滤掉这类请求头，您可以执行 `kubectl edit cm -n kube-system nginx-configuration` 命令在ConfigMap中添加配置。更多信息，请参见ConfigMap。

```
enable-underscores-in-headers：true
```

后端服务为HTTPS服务访问时是否可以通过Ingress-Nginx转发？

对于后端业务是HTTPS服务，但同样希望可以通过Ingress-Nginx转发时，执行以下命令，在Ingress资源配置中添加以下注解即可。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: xxxx
 annotations:
  # 注意这里：必须指定后端服务为HTTPS服务。
  nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
```

Ingress L7透传客户端IP吗？

Ingress-Nginx默认会通过X-Forward-For和X-Real-IP来透传客户端IP，但是当客户端主动在请求头里指定了X-Forward-For和X-Real-IP，会导致服务端获取不到真实的客户端IP。

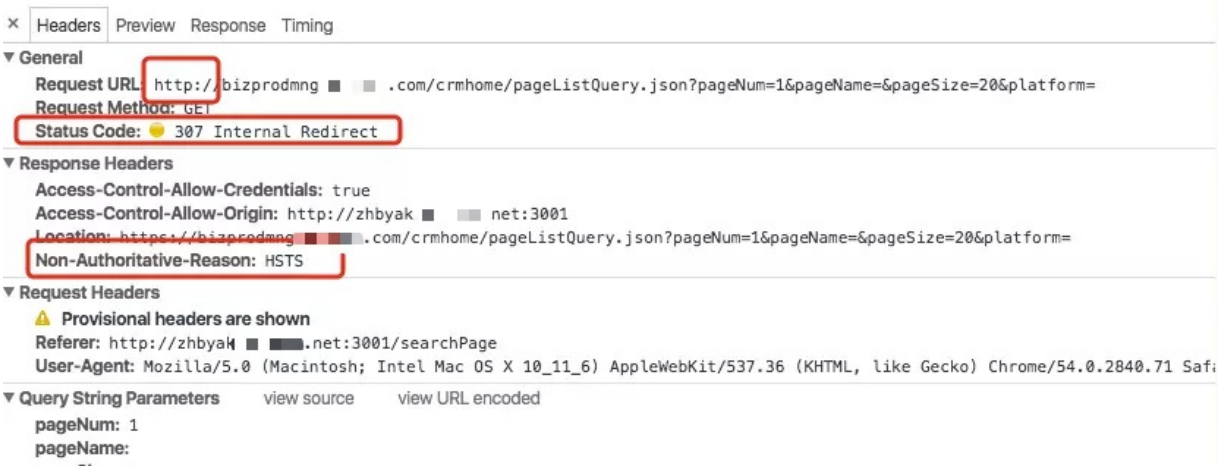
执行以下命令，开启 `kube-system/nginx-configuration configmap` 的以下配置可实现Ingress L7透传客户端IP。

```
compute-full-forwarded-for: "true"
forwarded-for-header: "X-Forwarded-For"
use-forwarded-headers: "true"
```

更多相关信息，请参见阿里云容器服务Ingress设置原IP透传。

nginx-ingress-controller组件支持HSTS吗？

nginx-ingress-controller组件默认是开启HSTS的，有些浏览器第一次基于PLAIN HTTP访问时，服务端（开启HSTS）会在返回给客户端的响应头里携带 Non-Authoritative-Reason: HSTS 字段，说明服务端支持HSTS，当客户端也支持的情况下下次会直接以HTTPS方式访问服务端。服务端返回的响应头消息体中包含有 307 Internal Redirect 状态码，具体如下图所示。



当客户端不希望支持自动转到HTTPS访问服务端时，您可以关闭nginx-ingress-controller组件的HSTS。具体操作，请参见HSTS。

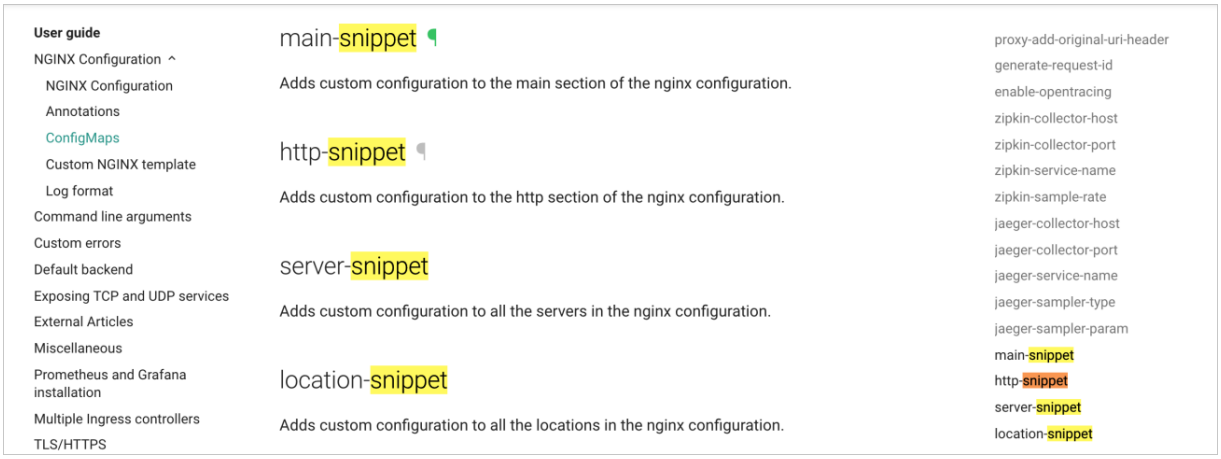
> 说明 在浏览器端HSTS默认是有缓存的，当关闭nginx-ingress-controller组件的HSTS后，请您清理缓存。

Ingress-Nginx支持哪些Rewrite配置？

目前Ingress-Nginx支持一些简单的Rewrite配置，具体请参见Rewrite。但是，对于一些高级的特别的Rewrite需求，您可以通过下面方式来配置。

- configuration-snippet：请参见Configuration snippet，扩展一些配置到Location章节中。
- server-snippet：请参见Server snippet，扩展一些配置到Server章节中。

同时，snippet也支持一些全局配置，具体如下图所示。更多相关信息，请参见main-snippet。



当版本升级后SLS解析日志不正常怎样修复？

ingress-nginx-controller组件当前主要有0.20和0.30两个版本，当通过控制台的**组件管理**从0.20升级到0.30版本后，由于0.20和0.30默认输出格式不同，在使用Ingress的灰度或蓝绿发布功能时，Ingress Dashboard会出现无法正确展示实际后端服务访问的情况。

执行以下操作步骤进行修复，更新 nginx-configuration configmap 和 k8s-nginx-ingress 配置。

1. 更新 nginx-configuration configmap 。
   - 如果您没有修改过 nginx-configuration configmap ，保存以下内容为 nginx-configuration.yaml ，然后执行 kubectl apply -f nginx-configuration.yaml 命令进行部署。

     ```
     apiVersion: v1
     kind: ConfigMap
     data:
      allow-backend-server-header: "true"
      enable-underscores-in-headers: "true"
      generate-request-id: "true"
      ignore-invalid-headers: "true"
      log-format-upstream: $remote_addr - [$remote_addr] - $remote_user [$time_local] "$request" $status $b
     ody_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time [$proxy_upstream_na
     me] $upstream_addr $upstream_response_length $upstream_response_time $upstream_status $req_id $h
     ost [$proxy_alternative_upstream_name]
      max-worker-connections: "65536"
      proxy-body-size: 20m
      proxy-connect-timeout: "10"
      reuse-port: "true"
      server-tokens: "false"
      ssl-redirect: "false"
      upstream-keepalive-timeout: "900"
      worker-cpu-affinity: auto
     metadata:
      labels:
        app: ingress-nginx
      name: nginx-configuration
      namespace: kube-system
     ```

   - 如果您修改过 nginx-configuration configmap ，为了避免覆盖您的配置，执行以下命令进行修复：

     ```
     kubectl edit configmap nginx-configuration -n kube-system
     ```

     在 log-format-upstream 字段末尾，添加 [$proxy_alternative_upstream_name] ， 然后保存退出。
2. 更新 k8s-nginx-ingress 配置。

   将以下内容保存为 k8s-nginx-ingress.yaml 文件，然后执行 kubectl apply -f k8s-nginx-ingress.yaml 命令进行部署。

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
  namespace: kube-system
 # 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
      - proxy_alternative_upstream_name
      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*)\s*\[*([^]]*)\]*.*
      SourceKey: content
```

在ACK组件管理中升级Nginx Ingress Controller组件时，系统所做的更新是什么？

Nginx Ingress Controller组件在0.44之前的版本，包含以下资源：

- serviceaccount/ingress-nginx
- configmap/nginx-configuration
- configmap/tcp-service
- configmap/udp-services
- clusterrole.rbac.authorization.k8s.io/ingress-nginx
- clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx
- role.rbac.authorization.k8s.io/ingress-nginx
- rolebinding.rbac.authorization.k8s.io/ingress-nginx
- service/nginx-ingress-lb
- deployment.apps/nginx-ingress-controller

Nginx Ingress Controller组件在0.44版本及其之后的版本，额外包含以下资源：

- validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission
- service/ingress-nginx-controller-admission
- serviceaccount/ingress-nginx-admission
- clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission
- clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission
- role.rbac.authorization.k8s.io/ingress-nginx-admission
- rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission
- job.batch/ingress-nginx-admission-create
- job.batch/ingress-nginx-admission-patch

在ACK的组件管理页面对Nginx Ingress Controller组件进行升级时，系统保留以下资源配置不变更：

- configmap/nginx-configuration
- configmap/tcp-services
- configmap/udp-services
- service/nginx-ingress-lb

所有其他资源的配置都会被覆盖成默认配置。以 `deployment.apps/nginx-ingress-controller` 资源配置为例，其默认的replicas参数为2。如果您升级Nginx Ingress Controller组件之前的replicas为5，但是通过组件管理升级Ingress后，其replicas将会为2，和默认配置一致。

如何将Ingress-nginx的监听由四层改为七层（HTTPS/HTTP）？

Ingress Pod的负载均衡默认是TCP 443 和TCP 80，您可以创建一个HTTPS/HTTP类型的负载均衡，将Ingress-nginx的监听由从四层改为七层。

> **说明** 修改监听时服务会有短暂中断，建议在业务低谷期进行修改监听操作。

1. 创建证书，并记录cert-id。具体操作，请参见选择阿里云签发证书。
2. 通过Annotation将Ingress所用负载均衡的监听由四层改为七层。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 服务**。

v. 在**服务**页面顶部设置**命名空间**为kube-system，单击ingress-nginx-lb右侧**操作**列下的**查看YAML**。

vi. 在编辑YAML对话框中annotations参数下添加以下内容，然后单击**更新**。

```
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-port: "http:80,https:443"
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id: "${YOUR_CERT_ID}"
```

3. 验证通过Annotation将负载均衡的监听由四层改为七层成功。

i. 在**服务**页面单击ingress-nginx-lb右侧的**监控信息**。

ii. 单击**监听**页签，可以看到监听的**前端协议**显示HTTP:80和HTTPS:443，说明通过Annotation将负载均衡的监听由四层改为七层成功。



ingress-nginx如何使用已有的SLB？

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面搜索ack-ingress-nginx，然后单击ack-ingress-nginx。
4. 在**创建**面板选择**集群**和**命名空间**。
5. 单击**参数**页签，设置参数。

i. 删除service的annotations参数下的注解。

```
service:
  # enable internet slb
  enabled: true
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type: "internet"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: "slb.s1.small"
```

ii. 在service的annotations参数中补充以下注解。

```
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: "${YOUR_LOADBALACER_ID}
```

6. 在**创建**面板单击**创建**。

创建Ingress时报错“certificate signed by unknown authority”



创建Ingress时，出现上图报错，原因是布置了多套Ingress，而各Ingress之间使用了相同的资源 (可能包括Secret、服务、Webhook配置等)，导致Webhook执行时与后端服务通信时使用的SSL证书不一致，从而出现错误。

解决办法：重新部署两套Ingress，两套Ingress包含的资源不能重复。关于Ingress中包含的资源信息，请参见在ACK组件管理中升级Nginx Ingress Controller组件时，系统所做的更新是什么？。

7.6. 服务发现DNS

7.6.1. 基于DNS的服务发现概述

阿里云容器服务Kubernetes版ACK部署了CoreDNS作为集群内的DNS服务器，为集群内的工作负载提供域名解析服务。本文主要介绍CoreDNS的概念、作用，以及CoreDNS作为集群内的DNS服务器的域名解析原理。

CoreDNS介绍

CoreDNS是Kubernetes集群中负责DNS解析的组件，能够支持解析集群内部自定义服务域名和集群外部域名。CoreDNS具备丰富的插件集，在集群层面支持自建DNS、自定义hosts、CNAME、rewrite等需求。与Kubernetes一样，CoreDNS项目由 CNCF托管。关于CoreDNS的更多信息，请参见CoreDNS: DNS and Service Discovery。

ACK集群使用CoreDNS负责集群的服务发现，您可以根据不同使用场景配置CoreDNS及使用CoreDNS提升集群DNS QPS性能：

- 关于ACK集群CoreDNS的详细配置说明，请参见集群CoreDNS配置。
- 关于如何使用CoreDNS提高ACK集群整体DNS查询性能，请参见优化集群DNS性能。

> **说明**
> 您还可以在ACK集群中部署Node Local DNS提升服务发现的稳定性和性能，Node Local DNS通过在集群节点上作为DaemonSet运行DNS缓存代理来提高集群DNS性能。关于如何在集群中部署Node Local DNS的具体步骤，请参见在ACK集群中使用NodeLocal DNSCache。

CoreDNS版本与集群Kubernetes版本关系

ACK采用CoreDNS版本与集群Kubernetes版本绑定的策略。只有ACK集群升级时才会升级CoreDNS版本。CoreDNS版本与集群Kubernetes版本对照关系如下表。

> **说明** 通过在**容器组**中查看名为coredns-xxx的Pod的镜像版本，可以查看CoreDNS当前的版本。具体操作，请参见管理容器组（Pod）。

关系项	版本			
Kubernetes	1.12	1.14	1.16	1.18
CoreDNS	1.2.6	1.3.1	1.6.2	1.6.7

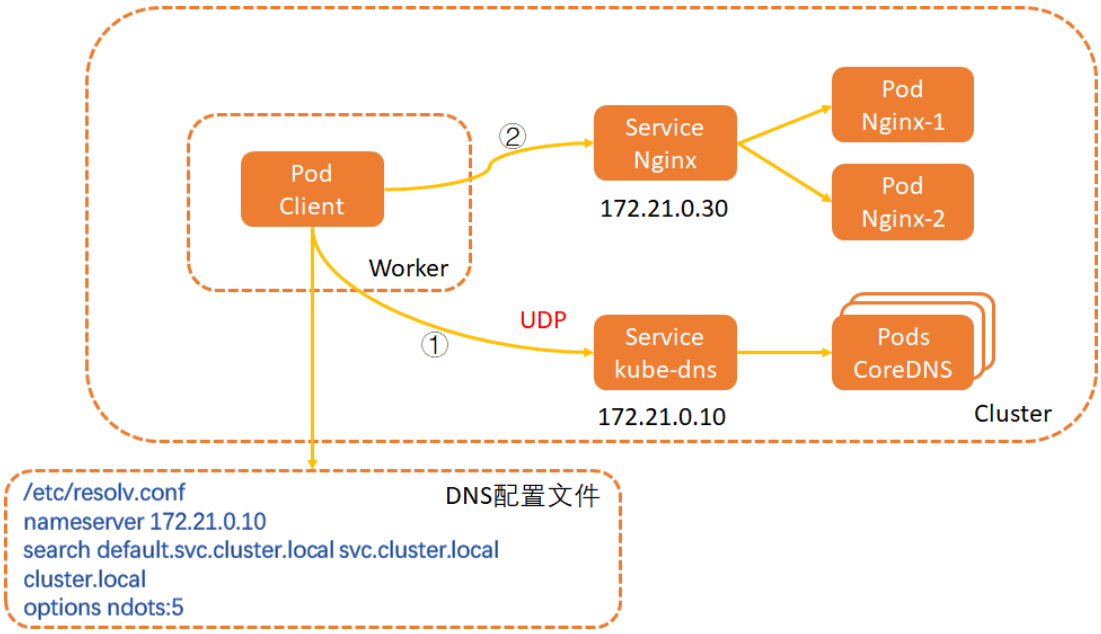
ACK集群内DNS域名解析原理

ACK集群中kubelet的启动参数有 `--cluster-dns=<dns-service-ip>` 和 `--cluster-domain=<default-local-domain>` ，这两个参数分别被用来设置集群DNS服务器的IP地址和主域名后缀。

ACK集群默认部署了一套CoreDNS工作负载，并通过**kube-dns**的服务名暴露DNS服务。ACK部署的CoreDNS工作负载后端是两个名为**coredns**的Pod，集群会根据Pod内的配置，将域名请求发往集群DNS服务器获取结果。Pod内的DNS域名解析配置文件为*/etc/resolv.conf*，文件内容如下。

```
nameserver 172.xx.x.xx
search kube-system.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

DNS解析原理示例图



序号	描述
①	业务Pod（Pod Client）试图访问Nginx服务（Service Nginx）时，先会请求本地DNS配置文件（/etc/resolv.conf）中指向的DNS服务器（nameserver 172.21.0.10，即Service kube-dns）获取服务IP地址，得到解析结果为172.21.0.30的IP地址。
②	业务Pod（Pod Client）再直接发起往该IP地址的请求，请求最终经过Nginx服务（Service Nginx）转发到达后端的Nginx容器（Pod Nginx-1和Pod Nginx-2）上。

关于集群DNS域名解析原理的详细说明，请参见集群DNS域名解析原理。

相关文档

- ACK集群内DNS原理和配置说明
- 优化集群DNS性能
- 在ACK集群中使用NodeLocal DNSCache

7.6.2. ACK集群内DNS原理和配置说明

在容器服务ACK集群中，推荐使用Service名作为服务的访问地址，因此需要一个ACK集群范围的DNS服务实现从Service名到Cluster IP（Service的IP地址）的解析。阿里云容器服务ACK部署了CoreDNS作为集群内的DNS服务器，为集群内的工作负载提供域名解析服务。本文介绍ACK集群DNS服务原理，并针对不同场景介绍如何进行DNS策略配置及CoreDNS配置。

前提条件

在进行内置DNS配置前，请确保您已完成以下操作：

- 创建Kubernetes托管版集群
- 通过kubectl工具连接集群

背景信息

ACK集群默认部署了一套DNS服务，通过kube-dns的服务名暴露DNS服务。您可执行以下命令查看kube-dns的服务详情。

```
kubectl get svc kube-dns -n kube-system
```

预期输出：

```
NAME    TYPE    CLUSTER-IP   EXTERNAL-IP  PORT(S)         AGE
kube-dns  ClusterIP  172.24.0.10  <none>    53/UDP,53/TCP,9153/TCP  27d
```

ACK部署的DNS服务后端是两个名为CoreDNS的Pod。您可执行以下命令查看CoreDNS的Pod详情。

```
kubectl get deployment coredns -n kube-system
```

预期输出：

```
NAME   READY  UP-TO-DATE  AVAILABLE  AGE
coredns  2/2   2      2      27d
```

集群DNS域名解析原理

ACK集群中kubelet的启动参数有`--cluster-dns=<dns-service-ip>`、`--cluster-domain=<default-local-domain>`，这两个参数分别被用来设置集群DNS服务器的IP地址和主域名后缀。

Pod内的DNS域名解析配置文件为*/etc/resolv.conf*，文件内容如下。

```
nameserver xx.xx.0.10
search kube-system.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

域名解析配置文件参数说明

参数	描述
nameserver	定义DNS服务器的IP地址。
search	设置域名的查找后缀规则，查找配置越多，说明域名解析查找匹配次数越多。ACK集群匹配有`kube-system.svc.cluster.local`、`svc.cluster.local`、`cluster.local`3个后缀，最多进行8次查询才能得到正确解析结果，因为集群里面进行IPV4和IPV6查询各四次。

参数	描述
options	定义域名解析配置文件选项，支持多个KV值。例如该参数设置成 `ndots:5` ，说明如果访问的域名字符串内的点字符数量超过 `ndots` 值，则认为是完整域名，并被直接解析；如果不足 `ndots` 值，则追加search段后缀再进行查询。

根据上述Pod内的配置，集群会将域名请求（集群内部定义的服务或是集群外部域名）查询发往集群DNS服务器获取结果。

集群dnsPolicy配置和场景说明

ACK支持通过dnsPolicy字段为每个Pod配置不同的DNS策略。目前ACK集群支持四种策略：

- ClusterFirst：通过CoreDNS来做域名解析，Pod内*/etc/resolv.conf*配置的DNS服务地址是集群DNS服务的kube-dns地址。该策略是集群工作负载的默认策略。
- None：忽略集群DNS策略，需要您提供dnsConfig字段来指定DNS配置信息。
- Default：Pod直接继承集群节点的域名解析配置。即在ACK集群直接使用ECS的*/etc/resolv.conf*文件（文件内配置的是阿里云DNS服务）。
- ClusterFirstWithHostNet：强制在hostNetWork网络模式下使用ClusterFirst策略（默认使用Default策略）。

针对上述四种策略，本文列举四种场景分别介绍如何配置dnsPolicy。

- **场景一：使用ACK集群提供的CoreDNS来做域名解析**
  针对这种场景，可使用 `dnsPolicy: ClusterFirst` 策略。示例配置如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: alpine
 namespace: default
spec:
 containers:
 - image: alpine
   command:
    - sleep
    - "10000"
   imagePullPolicy: Always
   name: alpine
 dnsPolicy: ClusterFirst
```

- **场景二：Pod层面自定义DNS配置**
  当您需要给Deployment类型的工作负载指定DNS配置时，可使用 `dnsPolicy: None` 策略。示例配置如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: alpine
 namespace: default
spec:
 containers:
 - image: alpine
  command:
   - sleep
   - "10000"
  imagePullPolicy: Always
  name: alpine
 dnsPolicy: None
 dnsConfig:
  nameservers: ["169.254.xx.xx"]
  searches:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local
  options:
  - name: ndots
   value: "2"
```

其中，dnsConfig中的参数说明如下：

参数	描述
nameservers	将用作Pod的DNS服务器的IP地址列表。最多可以指定3个IP地址。当Pod的dnsPolicy设置为 `None` 时，列表必须至少包含一个IP地址，否则此属性是可选的。列出的DNS的IP列表将合并到基于dnsPolicy生成的域名解析文件的nameserver字段中，并删除重复的地址。
searches	Pod中主机名查找的DNS搜索域列表。此属性是可选的。指定后，提供的列表将合并到从所选DNS策略生成的基本搜索域名中，并删除重复的域名。Kubernetes最多允许6个搜索域。
options	可选的对象列表，其中每个对象可以具有name属性（必需）和value属性（可选）。此属性中的内容将合并到从指定的DNS策略生成的选项中，并删除重复的条目。

更多信息，请参见Kubernetes官网的dns配置说明。

- **场景三：采用阿里云ECS的DNS配置**
  当您的应用Pod不需要访问集群内的其它服务，只需要通过阿里云DNS来做解析，也不希望DNS解析经过CoreDNS，可以采用 `dnsPolicy: Default` 策略。示例配置如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: alpine
 namespace: default
spec:
 containers:
 - image: alpine
  command:
   - sleep
   - "10000"
  imagePullPolicy: Always
  name: alpine
 dnsPolicy: Default
```

- **场景四：在HostNetwork网络模式下访问集群服务**
  如果您的应用Pod使用hostNetwork:true来配置网络，Pod中运行的应用程序可以直接看到宿主机的网络接口，其DNS策略默认为Default，不能访问集群内的服务。如果您希望在此网络模式下访问集群内服务，可使用 dnsPolicy: ClusterFirstWithHostNet 策略。示例配置如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: alpine
 namespace: default
spec:
 hostNetwork: true
 dnsPolicy: ClusterFirstWithHostNet
 containers:
 - image: alpine
  command:
   - sleep
   - "10000"
  imagePullPolicy: Always
  name: alpine
```

集群CoreDNS配置

针对场景一：使用ACK集群提供的CoreDNS来做域名解析，本文将介绍ACK集群CoreDNS的默认配置及针对常见功能的扩展配置。

> **说明** 如果您安装了local-dns，则命名空间kube-system下有一个node-local-dns配置项文件，其配置原理与CoreDNS的配置原理一致。

CoreDNS的默认配置

在命名空间kube-system下，ACK集群有一个CoreDNS配置项（有关如何查看配置项的具体步骤，请参见查看配置项）。CoreDNS会基于该配置项启用和配置插件。不同CoreDNS版本的配置项有略微差异，修改配置前请仔细阅读CoreDNS官方文档，以下是一个1.6.2版本CoreDNS默认采用的配置文件：

```
Corefile: |
 .:53 {
    errors
    health {
      lameduck 15s
    }
    ready
    kubernetes {{.ClusterDomain}} in-addr.arpa ip6.arpa {
     pods verified
     fallthrough in-addr.arpa ip6.arpa
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
 }
```

> **说明** 配置文件中 `ClusterDomain` 代指集群创建过程中填写的集群本地域名，默认值为 `cluster.local` 。

默认插件配置说明

参数	描述
errors	错误信息到标准输出。
health	CoreDNS自身健康状态报告，默认监听端口8080，一般用来做健康检查。您可以通过 `http://localhost:8080/health` 获取健康状态。
ready	CoreDNS插件状态报告，默认监听端口8181，一般用来做可读性检查。可以通过 `http://localhost:8181/ready` 获取可读状态。当所有插件都运行后，ready状态为200。
kubernetes	CoreDNS Kubernetes插件，提供集群内服务解析能力。
prometheus	CoreDNS自身metrics数据接口。可以通过 `http://localhost:9153/metrics` 获取prometheus格式的监控数据。
forward（或proxy）	将域名查询请求转到预定义的DNS服务器。默认配置中，当域名不在Kubernetes域时，将请求转发到预定义的解析器（*/etc/resolv.conf*）中。默认使用宿主机的*/etc/resolv.conf*配置。
cache	DNS缓存。
loop	环路检测，如果检测到环路，则停止CoreDNS。
reload	允许自动重新加载已更改的Corefile。编辑ConfigMap配置后，请等待两分钟以使更改生效。
loadbalance	循环DNS负载均衡器，可以在答案中随机A、AAAA、MX记录的顺序。

CoreDNS的扩展配置

针对以下不同场景，您可以扩展CoreDNS的配置：

- **场景一：开启日志服务**
  如果需将CoreDNS每次域名解析的日志打印出来，您可以开启log插件，在Corefile里加上log。示例配置如下：

```
Corefile: |
 .:53 {
    errors
    log
    health {
      lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
      pods insecure
      upstream
      fallthrough in-addr.arpa ip6.arpa
      ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
 }
```

- **场景二：特定域名使用自定义DNS服务器**
  如果example.com类型后缀的域名需要经过自建DNS服务器（IP为10.10.0.10）进行解析的话，您可为域名配置一个单独的服务块。示例配置如下：

```
example.com:53 {
 errors
 cache 30
 forward . 10.10.0.10
}
```

  完整配置如下：

```
Corefile: |
 .:53 {
    errors
    health {
      lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
     pods insecure
     upstream
     fallthrough in-addr.arpa ip6.arpa
     ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
 }
 example.com:53 {
    errors
    cache 30
    forward . 10.10.0.10
 }
```

- **场景三：外部域名完全使用自建DNS服务器**
  如果您需要使用的自建DNS服务的域名没有统一的域名后缀，您可以选择所有集群外部域名走自建DNS服务器（此时需要您将自建的DNS服务不能解析的域名转发到阿里云DNS，禁止直接更改集群ECS上的*/etc/resolv.conf*文件）。例如，您自建的DNS服务器IP为10.10.0.10和10.10.0.20，可以更改forward参数进行配置。示例配置如下：

```
Corefile: |
 .:53 {
    errors
    health {
      lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
     pods insecure
     upstream
     fallthrough in-addr.arpa ip6.arpa
     ttl 30
    }
    prometheus :9153
    forward . 10.10.0.10 10.10.0.20
    cache 30
    loop
    reload
    loadbalance
 }
```

- **场景四：自定义hosts**
  如果您需要为特定域名指定hosts，如为www.example.com指定IP为127.0.0.1，可以使用hosts插件来配置。示例配置如下：

```
Corefile: |
 .:53 {
    errors
    health {
      lameduck 5s
    }
    ready
    hosts {
     127.0.0.1 www.example.com
     fallthrough
    }
    kubernetes cluster.local in-addr.arpa ip6.arpa {
     pods insecure
     upstream
     fallthrough in-addr.arpa ip6.arpa
     ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
 }
```

> **注意** 请配置fallthrough，否则会造成非定制hosts域名解析失败。

- **场景五：集群外部访问集群内服务**
  如果您希望运行在集群ECS上的进程能够访问到集群内的服务，虽然可以通过将ECS的*/etc/resolv.conf*文件内nameserver配置为集群kube-dns的ClusterIP地址来达到目的，但不推荐您直接更改ECS的*/etc/resolv.conf*文件的方式来达到任何目的。
  内网场景下，您可以将集群内的服务通过内网SLB进行暴露，然后在云解析PrivateZone控制台通过添加A记录到该SLB的内网IP进行解析。具体操作，请参见添加解析记录。
- **场景六：统一域名访问服务或是在集群内对域名的做CNAME解析**
  您可以实现在公网、内网和集群内部通过统一域名foo.example.com访问您的服务，原理如下：
  - 集群内的服务 `foo.default.svc.cluster.local` 通过公网SLB进行了暴露，且有域名 `foo.example.com` 解析到该公网SLB的IP。
  - 集群内服务 `foo.default.svc.cluster.local` 通过内网SLB进行了暴露，且通过云解析PrivateZone在VPC内网中将 `foo.example.com` 解析到该内网SLB的IP。具体步骤，请参见上述为特定域名指定hosts。

- 在集群内部，您可以通过Rewrite插件将 foo.example.com CNAME到 foo.default.svc.cluster.local 。示例配置如下：

```
Corefile: |
 .:53 {
    errors
    health {
      lameduck 5s
    }
    ready
    rewrite stop {
     name regex foo.example.com foo.default.svc.cluster.local
     answer name foo.default.svc.cluster.local foo.example.com
    }
    kubernetes cluster.local in-addr.arpa ip6.arpa {
     pods insecure
     upstream
     fallthrough in-addr.arpa ip6.arpa
     ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
 }
```

7.6.3. 优化集群DNS性能

CoreDNS是Kubernetes集群中负责DNS解析的组件，能够支持解析集群内部自定义服务域名和集群外部域名。CoreDNS具备丰富的插件集，在集群层面支持自建DNS、自定义hosts、CNAME、rewrite等需求。ACK集群中默认部署形态在DNS QPS较高场景下，可能会出现解析压力，导致部分查询失败。本文介绍如何优化集群DNS性能。

前提条件

- 创建Kubernetes托管版集群
- 通过kubectl工具连接集群

背景信息

有关CoreDNS的更多信息，请参见CoreDNS。

合理调整集群CoreDNS副本数

调整CoreDNS副本数与集群节点数到合适比率有助于提升集群服务发现的性能，该比值推荐为1:8，即一个CoreDNS Pod支撑8个集群节点。

- 当集群节点无需大规模扩缩容时，执行以下命令调整目标副本数到目标值。

```
kubectl scale --replicas={target} deployment/coredns -n kube-system
```

> **说明** 将目标副本数 {target} 设置成目标值。

- 集群节点需要大规模扩缩容时，推荐部署以下YAML模板，使用集群水平伸缩器cluster-proportional-autoscaler动态调整副本数量。不建议使用针对QPS、CPU或MEM指标的水平伸缩器，实测效果较差。

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: dns-autoscaler
 namespace: kube-system
 labels:
  k8s-app: dns-autoscaler
spec:
 selector:
  matchLabels:
   k8s-app: dns-autoscaler
 template:
  metadata:
   labels:
    k8s-app: dns-autoscaler
  spec:
   serviceAccountName: admin
   containers:
   - name: autoscaler
    image: registry.cn-hangzhou.aliyuncs.com/acs/cluster-proportional-autoscaler:1.8.4
    resources:
     requests:
      cpu: "200m"
      memory: "150Mi"
    command:
    - /cluster-proportional-autoscaler
    - --namespace=kube-system
    - --configmap=dns-autoscaler
    - --nodelabels=type!=virtual-kubelet
    - --target=Deployment/coredns
    - --default-params={"linear":{"coresPerReplica":64,"nodesPerReplica":8,"min":2,"max":100,"preventSinglePo
intFailure":true}}
    - --logtostderr=true
    - --v=9
```

说明 上述使用线程伸缩策略中，CoreDNS副本数的计算公式为replicas = max (ceil (cores × 1/coresPerReplica), ceil (nodes × 1/nodesPerReplica))，且CoreDNS副本数受到 max ， min 限制。线程伸缩策略参数如下。

```
{
    "coresPerReplica": 64,
    "nodesPerReplica": 8,
    "min": 2,
    "max": 100,
    "preventSinglePointFailure": true
}
```

使用节点级NodeLocal DNSCache

在ACK集群中部署NodeLocal DNSCache可以提升服务发现的稳定性和性能，NodeLocal DNSCache通过在集群节点上作为DaemonSet运行DNS缓存代理来提高集群DNS性能。

关于更多NodeLocal DNSCache的介绍及如何在ACK集群中部署NodeLocal DNSCache的具体步骤，请参见在ACK集群中使用NodeLocal DNSCache。

相关文档

- 基于DNS的服务发现概述
- ACK集群内DNS原理和配置说明
- 在ACK集群中使用NodeLocal DNSCache

7.6.4. 在ACK集群中使用NodeLocal DNSCache

在ACK集群中部署NodeLocal DNSCache可以提升服务发现的稳定性和性能，NodeLocal DNSCache通过在集群节点上作为DaemonSet运行DNS缓存代理来提高集群DNS性能。本文介绍如何安装NodeLocal DNSCache，并在应用中使用NodeLocal DNSCache。

前提条件

- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 使用kubectl命令连接集群。具体操作，请参见通过kubectl工具连接集群。

使用限制

- NodeLocal DNSCache不支持ASK集群、以及托管版、专有版集群中部署的ECI类型的Pod。
- 如果集群网络类型为Terway，请更新Terway至v1.0.10.301及以上版本。如果是基于IPVLAN的Terway多IP模式，需要对Terway进行配置修改。更多信息，请参见修改Terway配置。
- 组件管理中的NodeLocal DNSCache和应用目录中的ack-node-local-dns为同一应用，不需要重复安装。
- NodeLocal DNSCache不提供Hosts、Rewrite等插件能力，仅作为CoreDNS的透明缓存代理。如有需要，可在CoreDNS配置中修改。
- 使用NodeLocal DNSCache前，您需要对CoreDNS配置进行修改，否则可能导致集群中PrivateZone域名或部分云产品提供的*.vpc-proxy.aliyuncs.com等域名无法解析。具体操作，请参见CoreDNS修改说明。

NodeLocal DNSCache简介

ACK NodeLocal DNSCache是基于社区开源项目NodeLocal DNSCache的一套DNS本地缓存解决方案。ACK NodeLocal DNSCache在Helm Chart名称是ack-node-local-dns。该方案主要由DNS本地缓存DaemonSet和DNSConfig动态注入控制器Deployment组成：

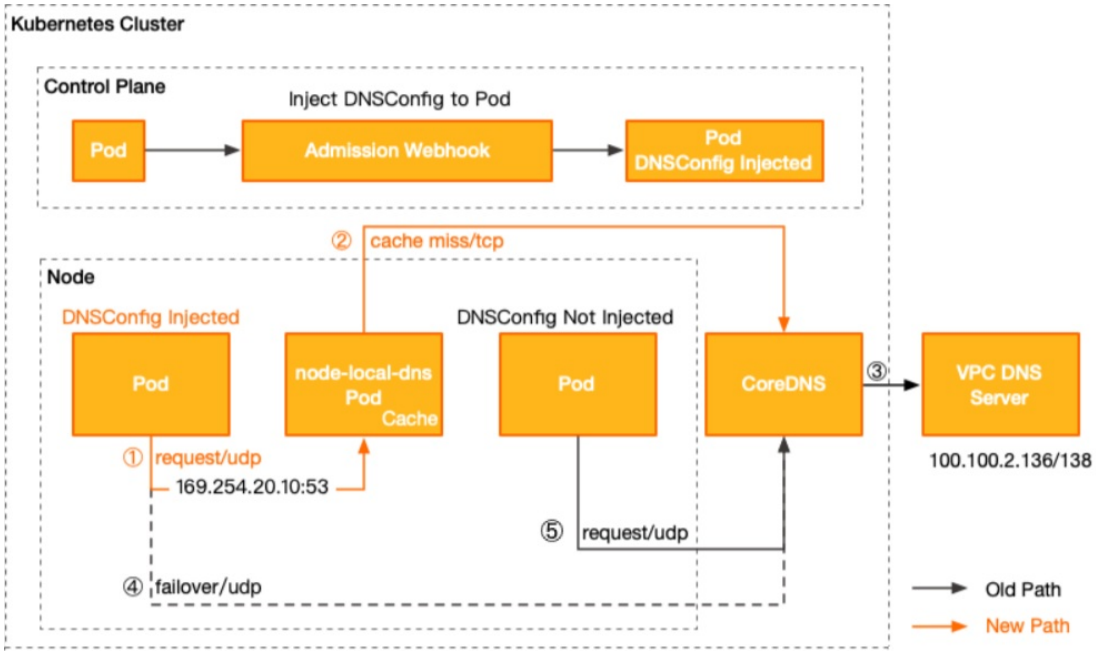
- DNSConfig动态注入控制器Deployment，基于Admission Webhook机制拦截Pod创建的请求，自动注入使用DNS缓存的Pod DNSConfig信息。

  > **说明** 如果不启用DNSConfig动态注入控制器Deployment，您需要手动注入DNS域名相关配置至Pod配置中。具体操作，请参见在应用中使用NodeLocal DNSCache下的方式二：手动指定DNSConfig。

- DNS缓存DaemonSet可以在每个节点上创建一个虚拟网络接口（默认监听IP 169.254.20.10，如需修改请提交工单咨询），配合Pod的DNSConfig和节点上的网络配置，Pod内产生的DNS请求会被该DaemonSet服务所代理。

  > **注意** DaemonSet服务内同样基于CoreDNS提供服务，但其仅具有代理和缓存的功能，请勿启用额外的插件能力（如Hosts、Rewrite等）。

  NodeLocal DNSCache工作原理图



序号	描述
①	已注入DNS本地缓存的Pod，默认会通过NodeLocal DNSCache监听于节点上的IP（169.254.20.10）解析域名。
②	NodeLocal DNSCache本地若无缓存应答解析请求，则会通过kube-dns服务请求CoreDNS进行解析。
③	CoreDNS对于非集群内域名，会通过VPC DNS服务器进行解析。
④	已注入DNS本地缓存的Pod，当无法连通NodeLocal DNSCache时，会继而直接通过kube-dns服务连接到CoreDNS进行解析，此链路为备用链路。
⑤	未注入DNS本地缓存的Pod，会通过标准的kube-dns服务链路连接到CoreDNS进行解析。

安装NodeLocal DNSCache

您可以通过组件管理或者应用目录页面安装NodeLocal DNSCache，建议您通过组件管理安装NodeLocal DNSCache。

通过组件管理安装NodeLocal DNSCache

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，选择目标集群名称或者目标集群右侧**操作**列下的**更多 > 系统组件管理**。
4. 在**组件管理**页面单击**网络**页签，找到ACK NodeLocal DNSCache。
5. 单击ACK NodeLocal DNSCache的**安装**，在弹出的对话框中单击**确定**。

通过应用目录安装NodeLocal DNSCache

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在应用目录页面搜索ack-node-local-dns，找到并单击**ack-node-local-dns**。
4. 在**创建**面板选择集群。
5. 仔细阅读**说明**页签中关于**ack-node-local-dns**的介绍及参数说明，然后单击**参数**页签，配置参数。

需设置的参数说明如下表。

参数	描述	获取路径
upstream_ip	kube-system命名空间下 kube-dns服务的ClusterIP。NodeLocal DNSCache通过该服务与CoreDNS通信来解析集群内域名。	如果您需要指定不同的上游DNS服务器，可以修改此参数。
clusterDomain	集群主域名。	查看节点上kubelet进程的--cluster-domain参数，默认为cluster.local。

6. 在右侧**创建**面板单击**创建**。

在应用中使用NodeLocal DNSCache

> **说明** NodeLocal DNSCache默认不会部署至Master节点。如果您的业务需要部署至Master节点，且您的Master节点设置了污点，您需要修改命名空间kube-system下node-local-dns DaemonSet的污点容忍。

为了能使应用原本请求CoreDNS的流量改为由DNS缓存DaemonSet代理，需要使Pod内部的中nameservers配置成169.254.20.10和kube-dns对应的IP地址，您有以下几种方式可以选择：

- 方式一：借助DNSConfig动态注入控制器在Pod创建时配置DNSConfig自动注入，推荐使用此方式。
- 方式二：创建Pod时手动指定DNSConfig。
- 方式三：修改kubelet参数，并重启节点kubelet。存在业务中断风险，不推荐使用此方式。

方式一：配置DNSConfig自动注入

DNSConfig动态注入控制器可用于自动注入DNSConfig至新建的Pod中，避免您手工配置Pod YAML进行注入。本应用默认会监听包含 `node-local-dns-injection=enabled` 标签的命名空间中新建Pod的请求，您可以通过以下命令给命名空间打上Label标签。

```
kubectl label namespace default node-local-dns-injection=enabled
```

> **说明**
> - 上述命令仅会开启default命名空间的自动注入，如需对其它命名空间开启自动注入，则需要替换 `default` 为目标命名空间名称。
> - 在命名空间DNSConfig自动注入开启的情况下，如需对部分Pod进行豁免（即不进行注入），可以调整其Pod Template中Labels标签字段，加上 `node-local-dns-injection=disabled` 标签。

开启自动注入后，您创建的Pod会被增加以下字段，为了最大程度上保证业务DNS请求高可用，nameservers中会额外加入kube-dns的ClusterIP地址作为备份的DNS服务器。

```
dnsConfig:
  nameservers:
  - 169.254.20.10
  - 172.21.0.10
  options:
  - name: ndots
    value: "3"
  - name: attempts
    value: "2"
  - name: timeout
    value: "1"
  searches:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local
 dnsPolicy: None
```

自动注入需满足以下条件。如果自动注入失败，请检查是否满足：

- 新建Pod不位于kube-system和kube-public命名空间。
- 新建Pod所在命名空间的Labels标签包含 `node-local-dns-injection=enabled` 。
- 新建Pod所在命名空间的Labels不包含ECI Pod相关标签，例如 `virtual-node-affinity-injection` 、 `eci` 、 `alibabacloud.com/eci` 等。
- 新建Pod没有被打上 `eci` 、 `alibabacloud.com/eci` 等ECI相关标签，或打上禁用DNS注入 `node-local-dns-injection=disabled` 标签。
- 新建Pod的网络为 `hostNetwork` 且DNSPolicy为 `ClusterFirstWithHostNet` ，或Pod为非 `hostNetwork` 且DNSPolicy为 `ClusterFirst` 。

方式二：手动指定DNSConfig

如果不适用Admission Webhook Deployment自动注入，您可以通过修改Pod进行手动指定DNSConfig。

```
apiVersion: v1
kind: Pod
metadata:
 name: alpine
 namespace: default
spec:
 containers:
 - image: alpine
  command:
   - sleep
   - "10000"
  imagePullPolicy: Always
  name: alpine
 dnsPolicy: None
 dnsConfig:
  nameservers: ["169.254.20.10","172.21.0.10"]
  searches:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local
  options:
  - name: ndots
   value: "3"
  - name: attempts
   value: "2"
  - name: timeout
   value: "1"
```

- dnsPolicy：必须为 `None` 。
- nameservers：配置成169.254.20.10和kube-dns的ClusterIP对应的IP地址。
- searches：设置搜索域，保证集群内部域名能够被正常解析。
- ndots：默认为5，可以适当降低ndots以提升解析效率。更多信息，请参见resolv.conf。

方式三：配置kubelet启动参数

kubelet通过--cluster-dns和--cluster-domain两个参数来全局控制Pod DNSConfig。
在*/etc/systemd/system/kubelet.service.d/10-kubeadm.conf* 配置文件中需要增加一个--cluster-dns参数，设置值为链路本地地址169.254.20.10。修改后执行 `systemctl daemon-reload` 和 `systemctl restart kubelet` 来生效。

```
--cluster-dns=169.254.20.10 --cluster-dns=<kube-dns ip> --cluster-domain=<search domain>
```

- cluster-dns：部署Pod时，默认采用的DNS服务器地址，默认只引用了 `` `kube-dns` `` 的服务IP，需要增加一个对链路本地地址169.254.20.10的引用。
- cluster-domain：部署Pod时，默认采用的DNS搜索域，保持原有搜索域即可，一般为 `` `cluster.local` `` 。

配置应用使用NodeLocal DNSCache示例

通过在default命名空间下部署Deployment类型示例应用，演示如何配置应用使用NodeLocal DNSCache。

1. 执行以下命令，给接入NodeLocal DNSCache的应用所在的命名空间（default）设置标签。

   ```
   kubectl label namespace default node-local-dns-injection=enabled
   ```

   > **注意** Admission Controller会忽略kube-system和kube-public命名空间下的应用，请勿在这两个命名空间下进行自动注入dnsConfig操作。

2. 在集群的default命名空间下部署示例应用。
   i. 使用以下YAML模板创建名为ubuntu-deployment的示例应用。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
 name: ubuntu
 labels:
  app: ubuntu
spec:
 replicas: 2
 selector:
  matchLabels:
   app: ubuntu
 template:
  metadata:
   labels:
    app: ubuntu
  spec:
   containers:
   - name: ubuntu
    image: ubuntu
    command: ["sh", "-c"]
    args: ["sleep 100000"]
```

   ii. 执行以下命令，使示例应用在集群中生效。

```
kubectl apply -f ubuntu-deployment.yaml
```

   预期输出：

```
deployment.apps/ubuntu created
```

   iii. 执行以下命令，查看应用信息。

```
kubectl get deployment ubuntu
```

   预期输出：

```
NAME   READY  UP-TO-DATE  AVAILABLE  AGE
ubuntu  2/2   2       2       7s
```

3. 查看是否成功注入dnsConfig。

   i. 执行以下命令，查看Pod信息。

```
kubectl get pods
```

   预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
ubuntu-766448f68c-mj8qk  1/1   Running  0     4m39s
ubuntu-766448f68c-wf5hw  1/1   Running  0     4m39s
```

ii. 执行以下命令，查看Pod的dnsConfig是否已经接入NodeLocal DNSCache。

```
kubectl get pod ubuntu-766448f68c-mj8qk -o=jsonpath='{.spec.dnsConfig}'
```

预期输出：

```
map[nameservers:[169.254.20.10 172.21.0.10] options:[map[name:ndots value:5]] searches:[default.svc.cluster.local svc.cluster.local cluster.local]]
```

当输出以上内容时，说明成功为应用接入NodeLocal DNSCache。
开启自动注入后，您创建的Pod会被增加以下字段，为了最大程度上保证业务DNS请求高可用，nameservers中会额外加入kube-dns服务的ClusterIP地址（172.21.0.10）作为备份的DNS服务器。

```
dnsConfig:
 nameservers:
 - 169.254.20.10
 - 172.21.0.10
 options:
 - name: ndots
  value: "3"
 - name: attempts
  value: "2"
 - name: timeout
  value: "1"
 searches:
 - default.svc.cluster.local
 - svc.cluster.local
 - cluster.local
dnsPolicy: None
```

升级NodeLocal DNSCache

如果您是在组件管理页面安装NodeLocal DNSCache，您需要在组件管理页面升级NodeLocal DNSCache。如果您是在应用目录页面安装NodeLocal DNSCache，您需要在Helm页面卸载NodeLocal DNSCache，然后再重新安装NodeLocal DNSCache。

通过组件管理页面升级NodeLocal DNSCache

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**组件管理**页面单击NodeLocal DNSCache下的**升级**，在弹出的对话框中单击**确定**。

> **说明** 若您对node-local-dns DaemonSet进行过污点容忍等修改操作，升级时污点容忍等操作将会被覆盖，升级后需要再次设置。

通过应用目录页面升级NodeLocal DNSCache

如果您是通过应用目录页面安装NodeLocal DNSCache，您可以先卸载NodeLocal DNSCache，再重新安装。具体操作，请参见卸载NodeLocal DNSCache和安装NodeLocal DNSCache。

卸载NodeLocal DNSCache

如果您是在组件管理页面安装NodeLocal DNSCache，您需要在组件管理页面卸载NodeLocal DNSCache。如果您是在应用目录页面安装NodeLocal DNSCache，您需要在Helm页面卸载NodeLocal DNSCache。

通过组件管理页面卸载NodeLocal DNSCache

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**组件管理**页面单击NodeLocal DNSCache下的**卸载**，在弹出的对话框中单击**确定**。

> **说明** 卸载后所有解析流量会请求至CoreDNS，建议提前对CoreDNS副本数进行扩容。

通过应用目录页面卸载NodeLocal DNSCache

1. 卸载前检查。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏选择**应用 > Helm**。
   v. 在**Helm**页面单击**ack-node-local-dns-default**。
   vi. 单击**参数**页签，检查参数。
   - 如果本地监听IP地址（local_dns_ip）等同于kube-dns的ClusterIP地址，则您可以执行步骤2卸载NodeLocal DNSCache。
   - 如果高可用开关（high_availability）为打开状态，且本地监听IP地址（local_dns_ip）为169.254.20.10。则您可以执行步骤2卸载NodeLocal DNSCache。
   - 如果参数不满足以上条件，请按照以下流程进行卸载：
     a. 执行 `kubectl get ns -o yaml` 命令，检查哪些命名空间被打上了node-local-dns-injection=enabled的标签，并将标签删除。
     b. 执行 `kubectl get pod -o yaml` 命令，检查哪些业务Pod被注入DNSConfig，删除这些Pod并重建。
     c. 执行步骤2卸载NodeLocal DNSCache。
2. 卸载NodeLocal DNSCache。
   i. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏选择**应用 > Helm**。
   v. 在**Helm**页面单击ack-node-local-dns-default**操作**列的**删除**，在弹出的对话框中单击**确定**。

CoreDNS修改说明

NodeLocal DNSCache会默认采用TCP协议与CoreDNS进行通信，CoreDNS会根据请求来源使用的协议与上游进行通信，即与VPC内DNS服务器进行通信。

VPC内DNS服务器目前无法以TCP协议提供PrivateZone域名和部分云产品提供的*.vpc-proxy.aliyuncs.com域名的解析，因此通过NodeLocal DNSCache解析此类域名时，解析会失败。您需要参考以下方式修改CoreDNS配置，强制CoreDNS默认以UDP协议与上游VPC内DNS服务器进行通信。

建议您使用以下方式修改CoreDNS配置文件，即修改命名空间kube-system下**coredns**。具体操作，请参见管理配置项。在forward插件中指定请求上游的协议为 `perfer_udp` ，修改之后CoreDNS会优先使用UDP协议与上游通信。修改方式如下所示：

```
# 修改前
forward . /etc/resolv.conf
# 修改后
forward . /etc/resolv.conf {
 prefer_udp
}
```

修改Terway配置

1. 执行以下命令，查看Terway配置文件。

   ```
   kubectl -n kube-system edit cm eni-config -o yaml
   ```

2. 检查Terway配置文件。
   - 检查配置文件中 `eniip_virtual_type` 字段是否开启IPVLAN模式。如果配置文件中此项不存在或不是IPVLAN，则您不需要按后续步骤进行配置，可以直接安装NodeLocal DNSCache。具体操作，请参见安装NodeLocal DNSCache。
   - 检查配置文件中是否已配置 `host_stack_cidrs` 字段。如果配置文件中已配置 `host_stack_cidrs` 字段，则您不需要按后续步骤进行配置，可以直接安装NodeLocal DNSCache。具体操作，请参见安装NodeLocal DNSCache。
3. 在Terway配置文件中添加 `host_stack_cidrs` 字段，并输入网段169.254.20.10/32，保存并退出。

   ```
   10-terway.conf: |
   {
    "cniVersion": "0.3.0",
    "name": "terway",
    "eniip_virtual_type": "IPVlan",
    "host_stack_cidrs": ["169.254.20.10/32"],
    "type": "terway"
   }
   ```

4. 执行以下命令，查看当前Terway的所有DaemonSet容器组。

   ```
   kubectl -n kube-system get pod | grep terway-eniip
   ```

   预期输出：

   ```
   terway-eniip-7d56l     2/2   Running  0      30m
   terway-eniip-s7m2t     2/2   Running  0      30m
   ```

5. 执行以下命令，触发Terway容器组重建。

   ```
   kubectl -n kube-system delete pod terway-eniip-7d56l terway-eniip-s7m2t
   ```

6. 登录任意集群节点，执行以下命令，查看Terway配置文件。

   如果Terway配置文件包含添加的网段，则说明配置成功。

   ```
   cat /etc/cni/net.d/*
   ```

   预期输出：

```
{
 "cniVersion": "0.3.0",
 "name": "terway-chainer",
 "plugins": [
  {
   "eniip_virtual_type": "IPVlan",
   "host_stack_cidrs": [
    "169.254.20.10/32",
   ],
   "type": "terway"
  },
  {
   "type": "cilium-cni"
  }
 ]
}
```

所有Terway Pod重建并正常后，您可以继续安装NodeLocal DNSCache了。

相关文档

- 基于DNS的服务发现概述
- ACK集群内DNS原理和配置说明
- 优化集群DNS性能

7.6.5. 使用ACK CoreDNS DNSTAP Analyser诊断CoreDNS域名解析异常

ACK集群的ACK CoreDNS DNSTAP Analyser组件，可以从CoreDNS和客户端之间，CoreDNS和上游DNS服务器之间的请求和响应过程中，对所有产生的DNS网络报文进行归类和识别。借助此工具，您可以快速定位域名无法解析的原因。本文介绍如何使用ACK CoreDNS DNSTAP Analyser诊断CoreDNS域名解析异常。

前提条件

- 创建ACK Pro版集群
- 通过kubectl工具连接集群

背景信息

DNSTAP是一种DNS报文日志格式协议，可以被用于诊断DNS服务器上异常解析问题。更多信息，请参见dnstap。

步骤一：安装ACK CoreDNS DNSTAP Analyser

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面搜索*ack-coredns-dnstap-analyser*，找到并单击**ack-coredns-dnstap-analyser**。
4. 在**应用目录-ack-coredns-dnstap-analyser**页面右侧**创建**面板中，选择目标集群，然后在页面左侧单击**说明**页签并阅读相关信息，然后在**参数**页签中根据需求设置参数。
5. 在页面右侧，单击**创建**。

步骤二：配置CoreDNS的DNSTAP插件

CoreDNS中内置了发送DNSTAP报文的DNSTAP插件，DNSTAP插件收到和发出的DNS报文可以发送至ACK CoreDNS DNSTAP Analyser。使用ACK CoreDNS DNSTAP Analyser诊断CoreDNS域名解析异常前，您需要先配置CoreDNS的DNSTAP插件。

1. 执行以下命令，获取kube-system命名空间下ACK CoreDNS DNSTAP Analyser服务的ClusterIP地址（例如

172.21.0.10）用于配置DNSTAP插件。

```
kubectl -n kube-system get svc dnstap-analyser
```

2. 执行以下命令，编辑CoreDNS配置文件。

```
kubectl -n kube-system edit cm coredns -o yaml
```

3. 参考以下注释修改CoreDNS的默认配置。

修改配置的目的是启用DNSTAP插件，并指向地址至CoreDNS DNSTAP Analyser的ClusterIP地址，从而将DNSTAP插件收到和发出的DNS报文发送至ACK CoreDNS DNSTAP Analyser。

```
json    Corefile: |
    .:53 {
      // 省略
      ready
      // 添加以下一行，将172.21.0.10替换成获取到的CoreDNS DNSTAP Analyser ClusterIP地址。
      dnstap tcp://172.21.0.10:6000 full
      kubernetes cluster.local in-addr.arpa ip6.arpa {
       // 省略
      }
      // 省略
      reload
      loadbalance
    }
```

4. 执行以下命令，查看当前CoreDNS的所有容器组。

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

预期输出：

```
coredns-7d56l     1/1    Running  0      30m
coredns-s7m2t     1/1    Running  0      30m
```

5. 执行以下命令，持续查看CoreDNS日志。

```
kubectl -n kube-system logs -f --tail=500 coredns-7d56l
```

> **说明** 替换容器组名称coredns-7d56l为步骤3中出现的任意一容器组名称。

等待约一分钟，观察日志中出现以下输出，并无其它异常时，说明配置修改成功。

```
[INFO] Reloading complete
```

通过以上操作您已启用CoreDNS DNSTAP插件，此时CoreDNS会将接收和发送到的DNS报文转发至ACK CoreDNS DNSTAP Analyser进行自动化诊断。

步骤三：查看ACK CoreDNS DNSTAP Analyser日志

1. 执行以下命令，查看ACK CoreDNS DNSTAP Analyser的所有容器组。

```
kubectl -n kube-system get pod | grep dnstap-analyser
```

预期输出：

```
dnstap-analyser-bbdf879-gpfkm     1/1    Running  0      30m
```

2. 执行以下命令，查看ACK CoreDNS DNSTAP Analyser日志。

```
kubectl -n kube-system logs -f dnstap-analyser-bbdf879-gpfkm
```

> **说明** 替换容器组名称dnstap-analyser-bbdf879-gpfkm为步骤1中出现的容器组名称。

步骤四：分析ACK CoreDNS DNSTAP Analyser日志

在ACK CoreDNS DNSTAP Analyser日志中，每一行是一个经过聚合的DNS请求上下文（Session），包含了从客户端、CoreDNS、上游DNS服务器之间的所有请求和响应的报文。DNS请求上下文（Session）包括：

- Status字段：不同的Status代表不同的场景。

上下文状态	描述
Succeeded	完整的域名查询请求，域名的响应是正常的（状态为NXDOMAIN、NOERROR）。
Failed	有问题的域名查询请求，域名响应中状态码异常或上游服务器连接超时等情况。
SampleLoss	CoreDNS或Analyser负载较高，导致DNSTAP报文丢失，出现这类状态时，应调整CoreDNS或Analyser副本数，使单个容器的负载下降，提高诊断可信度。

- BitMap字段：BitMap的值代表了该请求上下文中截取到的DNSTAP报文的种类，根据截取到的不同种类，得到可能的异常原因。

上下文Session BitMap值	含义
1	CoreDNS接收到了客户端的查询请求，但未返回结果给客户端。
3	一次完整的内部域名查询，从客户端发起了请求，CoreDNS本地缓存中或Kubernetes中存在命中该域名的解析结果，将结果返回给了客户端。
5	无法连接到上游服务器，从客户端发起了请求，CoreDNS将请求转发给上游服务器进行查询，上游服务器未返回结果。
15	一次完整的外部域名查询，从客户端发起了请求，CoreDNS将请求转发给上游服务器进行查询，上游服务器完成了查询并返回，CoreDNS又将结果返回给了客户端。

- Messages字段：DNS请求上下文Session中Messages字段记录了DNS请求、响应的原始报文的JSON输出，可以根据dns-parameters-6对报文内容进行问题诊断。

（可选）

（可选）步骤五：通过SLS仪表盘查看异常DNS报文信息

如果您配置了阿里云日志服务SLS的仪表盘，并且在ACK CoreDNS DNSTAP Analyser组件安装时使用了推荐的`env`日志采集配置（参考步骤4中的**说明**页签的内容），您可以在ACK集群对应的SLS Project的Kubernetes CoreDNS DNSTAP报文监控仪表盘中看到异常DNS报文信息。具体操作，请参见分析和监控CoreDNS日志。

相关文档

- 基于DNS的服务发现概述
- ACK集群内DNS原理和配置说明

7.6.6. CoreDNS升级说明

在CoreDNS组件的升级过程中，ACK会进行CoreDNS配置文件即Corefile的配置迁移，同时进行CoreDNS容器组的重建。整个升级过程中，您可能会遇到部分DNS解析请求失败或者超时的情况。本文介绍CoreDNS升级前的检查和优化操作，以降低DNS解析请求失败或者超时的可能性。

前提条件

已通过kubectl工具连接集群。具体操作，请参见通过kubectl工具连接集群。

升级CoreDNS过程说明

CoreDNS升级过程中，ACK会用新版本的YAML模板覆盖您集群中旧版本的CoreDNS YAML模板，只有以下类型变更会被保留：

- CoreDNS的副本数。
- CoreDNS配置项，即ConfigMap **coredns**。

从1.7.0版本起，CoreDNS的默认内存限制会被调整至2 GB，正常情况下不会出现Pod内存溢出OOM（Out of Memory）情况，无需对内存限制再次修改。

若您此前对YAML模板进行手动自定义变更，修改了容忍Toleration、内存和CPU的资源请求和限制等字段，将会被覆盖。针对这种情况，您可以手动升级CoreDNS，或者在自动升级CoreDNS后再次更新YAML模板内容。具体操作，请参见【组件升级】CoreDNS升级公告。

升级CoreDNS组件前，进行以下操作可降低DNS解析请求失败或者超时的可能性：

- 配置IPVS类型集群的UDP超时时间
- 配置Kube-DNS服务
- 配置CoreDNS优雅退出

配置IPVS类型集群的UDP超时时间

如果您的ACK集群使用了kube-proxy IPVS模式，IPVS的会话保持策略会导致整个集群在升级完成后五分钟内出现概率性解析失败的问题。您可以按以下方式降低IPVS UDP类型的会话保持超时时间至10秒，以减少解析失败的次数。如果您ACK集群中包含UDP类型的业务，需要您提前评估该操作是否有影响再执行该操作。如果您无法评估，您可以提交工单咨询。

> **说明** 如果您的集群不是IPVS类型，请忽略配置IPVS类型集群的UDP超时时间的操作。关于如何查看kube-proxy代理模式，请参见查看基本信息。

K8s 1.18及以上版本集群

控制台操作方式

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在顶部选择**kube-system**命名空间，然后单击配置项*kube-proxy-worker*右侧的**YAML编辑**。
6. 在**查看YAML**面板中的ipvs字段下，添加 `udpTimeout: 10s` ，然后单击**确定**。

```
apiVersion: v1
data:
 config.conf: |
  apiVersion: kubeproxy.config.k8s.io/v1alpha1
  kind: KubeProxyConfiguration
  // 其它不相关字段已省略。
  mode: ipvs
  // 如果ipvs键不存在，需要添加此键。
  ipvs:
   udpTimeout: 10s
```

7. 重建所有名为kube-proxy-worker的容器。
   i. 在集群管理页左侧导航栏中，选择**工作负载 > 守护进程集**。
   ii. 在守护进程集列表中，找到并单击**kube-proxy-worker**。
   iii. 在**kube-proxy-worker**页面中的**容器组**页签下对应容器组右侧，选择**更多 > 删除**，然后单击**确定**。
   重复操作删除所有容器组。删除容器组后，系统会自动重建所有容器。
8. 验证UDP超时时间的配置是否成功。
   i. 执行以下命令安装**ipvsadm**。
   ipvsadm是IPVS模块的管理工具。更多信息，请参见ipvsadm。

   ```
   yum install -y ipvsadm
   ```

   ii. 在集群任意一台ECS节点中执行以下命令查看第三个数字。

   ```
   ipvsadm -L --timeout
   ```

   如果输出结果中第三个数字是10，则说明IPVS类型集群的UDP超时时间变更成功。

   > **说明** 变更成功后，请观察至少五分钟后再进行下一步操作。

命令行操作方式

1. 执行以下命令修改kube-proxy的配置文件*kube-proxy-worker*。

```
kubectl -n kube-system edit configmap kube-proxy-worker
```

2. 在kube-proxy配置文件中的ipvs字段下，添加 `udpTimeout: 10s` 并保存退出。

```
apiVersion: v1
data:
 config.conf: |
  apiVersion: kubeproxy.config.k8s.io/v1alpha1
  kind: KubeProxyConfiguration
  // 其它不相关字段已省略。
  mode: ipvs
  // 如果ipvs键不存在，需要添加此键。
  ipvs:
   udpTimeout: 10s
```

3. 执行以下命令重建所有名为kube-proxy-worker的容器。
   i. 执行以下命令查看存在的容器组信息。

   ```
   kubectl -n kube-system get pod -o wide | grep kube-proxy-worker
   ```

ii. 执行以下命令删除上步骤中查看所有容器，系统将会自动重建名为kube-proxy-worker容器。

```
kubectl -n kube-system delete pod <kube-proxy-worker-****>
```

> **说明** 将<kube-proxy-worker-****>替换为上述打印的所有容器组名称。

4. 验证UDP超时时间的配置是否成功。

i. 执行以下命令安装**ipvsadm**。

ipvsadm是IPVS模块的管理工具。更多信息，请参见ipvsadm。

```
yum install -y ipvsadm
```

ii. 在集群任意一台ECS节点中执行以下命令查看第三个数字。

```
ipvsadm -L --timeout
```

如果输出结果中第三个数字是10，则说明IPVS类型集群的UDP超时时间变更成功。

> **说明** 变更成功后，请观察至少五分钟后再进行下一步操作。

K8s 1.16及以下版本集群

此类版本集群的kube-proxy不支持udpTimeout参数，推荐使用OOS服务批量在所有集群机器上执行 `ipvsadm` 命令以调整UDP超时时间配置。命令如下：

```
yum install -y ipvsadm
ipvsadm -L --timeout > /tmp/ipvsadm_timeout_old
ipvsadm --set 900 120 10
ipvsadm -L --timeout > /tmp/ipvsadm_timeout_new
diff /tmp/ipvsadm_timeout_old /tmp/ipvsadm_timeout_new
```

关于OOS的批量操作实例介绍，请参见批量操作实例。

配置Kube-DNS服务

控制台操作方式

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在kube-system命名空间下，单击服务**kube-dns**右侧的**查看YAML**。
   - 如果发现sessionAffinity字段为 `None` ，则无需进行以下步骤。
   - 如果发现sessionAffinity为 `ClientIP` ，则进行以下步骤。
6. 删除sessionAffinity、sessionAffinityConfig及所有子键，然后单击**更新**。

```
// 删除以下所有内容。
sessionAffinity: ClientIP
 sessionAffinityConfig:
   clientIP:
    timeoutSeconds: 10800
```

7. 再次单击服务**kube-dns**右侧的**查看YAML**，校验sessionAffinity字段是否为 `None` ，为 `None` 则Kube-DNS服务变更成功。

命令行操作方式

1. 执行以下命令查看kube-dns服务配置信息。

   ```
   kubectl -n kube-system get svc kube-dns -o yaml
   ```

   - 如果发现sessionAffinity字段为 `None` ，则无需执行以下步骤。
   - 如果发现sessionAffinity为 `ClientIP` ，则执行以下步骤。
2. 执行以下命令打开并编辑名为kube-dns的服务。

   ```
   kubectl -n kube-system edit service kube-dns
   ```

3. 删除sessionAffinity相关设置（sessionAffinity、sessionAffinityConfig及所有子键），并保存退出。

   ```
   // 删除以下所有内容。
   sessionAffinity: ClientIP
    sessionAffinityConfig:
     clientIP:
      timeoutSeconds: 10800
   ```

4. 修改完成后，再次运行以下命令查看sessionAffinity字段是否为 `None` ，为 `None` 则Kube-DNS服务变更成功。

   ```
   kubectl -n kube-system get svc kube-dns -o yaml
   ```

配置CoreDNS优雅退出

> **说明** CoreDNS刷新配置过程中，可能会占用额外内存。修改CoreDNS配置项后，请观察Pod运行状态，如果出现Pod内存不足的情况，请及时修改CoreDNS Deployment中容器内存限制，建议将内存调整至2 GB。

控制台操作方式

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在kube-system命名空间下，单击配置项*coredns*右侧的**YAML编辑**。
6. 参考下列的YAML文件，保证health插件开启，并调整lameduck参数为 `15s` ，然后单击**确定**。

   ```
   .:53 {
       errors
       // health插件在不同的CoreDNS版本中可能有不同的设置情形。
       // 情形一：默认未启用health插件。
       // 情形二：默认启用health插件，但未设置lameduck时间。
       health
       // 情形三：默认启用health插件，设置了lameduck时间为5秒。
       health {
         lameduck 5s
       }
       // 对于以上三种情形，应统一修改成以下，调整lameduck参数为15秒。
       health {
         lameduck 15s
       }
       // 其它插件不需要修改，此处省略。
     }
   ```

如果CoreDNS Pod正常运行则说明CoreDNS优雅退出的配置更新成功。如果CoreDNS Pod出现异常，可以通过查看Pod事件及日志定位原因。

命令行操作方式

1. 执行以下命令打开CoreDNS配置文件。

```
kubectl -n kube-system edit configmap coredns
```

2. 参考下列的YAML文件，保证health插件开启，并调整lameduck参数为 `15s` 。

```
.:53 {
    errors
    // health插件在不同的CoreDNS版本中可能有不同的设置情形。
    // 情形一：默认未启用health插件。
    // 情形二：默认启用health插件，但未设置lameduck时间。
    health
    // 情形三：默认启用health插件，设置了lameduck时间为5秒。
    health {
      lameduck 5s
    }
    // 对于以上三种情形，应统一修改成以下，调整lameduck参数为15秒。
    health {
      lameduck 15s
    }
    // 其它插件不需要修改，此处省略。
  }
```

3. 修改CoreDNS配置文件后保存退出。
   如果CoreDNS正常运行则说明CoreDNS优雅退出的配置更新成功。如果CoreDNS Pod出现异常，可以通过查看Pod事件及日志定位原因。

相关文档

- In IPVS mode, kube-proxy doesn't handle graceful termination of UDP flows
- CoreDNS
- 容器网络FAQ

7.6.7. Kubernetes域名解析异常问题排查

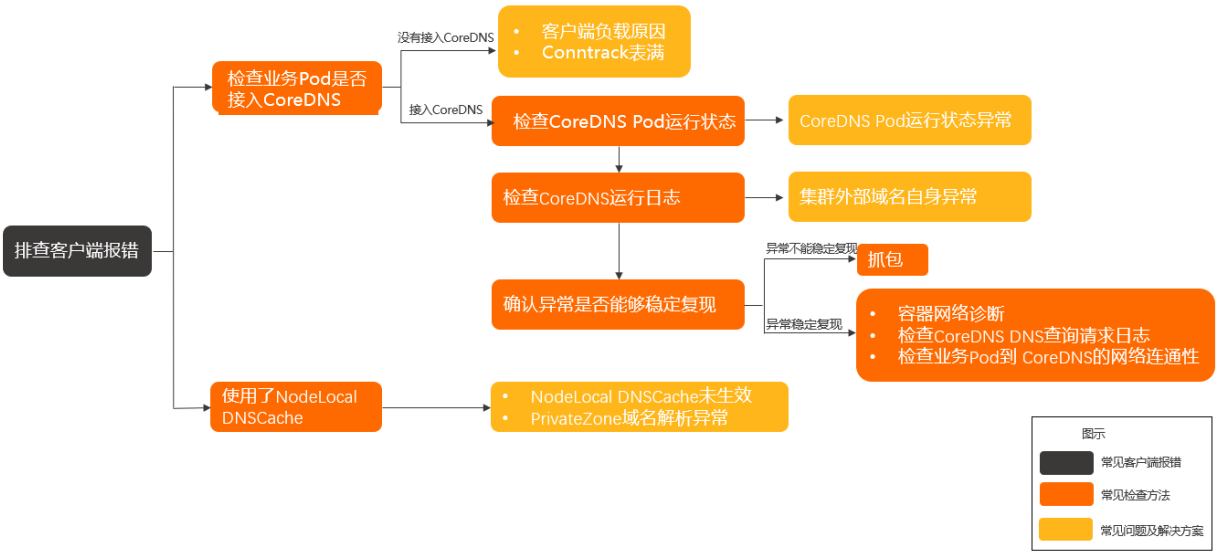
7.6.7.1. 诊断流程与排查思路

本文介绍关于Kubernetes集群域名解析异常的诊断流程、排查思路，以及常见的客户端报错内容。

基本概念

- **集群内部域名**：CoreDNS会将集群中的服务暴露为集群内部域名，默认以 `.cluster.local` 结尾，这类域名的解析通过CoreDNS内部缓存完成，不会从上游DNS服务器查询。
- **集群外部域名**：除了集群内部域名以外的域名。外部域名可由CoreDNS配置文件提供解析结果，或从DNSConfig配置文件中指定的上游DNS服务器查询，默认使用阿里云VPC中的私网DNS服务地址（100.100.2.136和100.100.2.138）。您可以修改其为您自建的DNS服务器。
- **业务Pod**：您部署在Kubernetes集群中的容器Pod，不包含Kubernetes自身系统组件的容器。
- **接入CoreDNS的业务Pod**：容器内DNS服务器指向了CoreDNS的业务Pod。
- **接入NodeLocal DNSCache的业务Pod**：集群中安装了NodeLocal DNSCache插件后，通过自动或手动方式注入DNSConfig的业务Pod。这类Pod在解析域名时，会优先访问本地缓存组件。如果访问本地缓存组件不通时，会访问CoreDNS提供的kube-dns服务。

异常诊断流程



1. 判断当前的异常原因。具体信息，请参见常见客户端报错。
   - 如果异常原因是域名不存在，请参见排查思路的*按解析异常的域名类型*。
   - 如果异常原因是无法连接至域名服务器，请参见排查思路的*按解析异常出现频次*。
2. 如果以上排查无果，请按以下步骤排查。
   - 检查业务Pod的DNS配置，是否已经接入CoreDNS。具体操作，请参见检查业务Pod的DNS配置。
     - 如果没有接入CoreDNS，则考虑是客户端负载原因或Conntrack表满导致解析失败。具体操作，请参见客户端负载原因导致解析失败和Conntrack表满。
     - 如果接入了CoreDNS，则按以下步骤排查。
       a. 通过检查CoreDNS Pod运行状态进行诊断。具体操作，请参见检查CoreDNS Pod运行状态和CoreDNS Pod运行状态异常。
       b. 通过检查CoreDNS运行日志进行诊断。具体操作，请参见检查CoreDNS运行日志和集群外部域名自身异常。
       c. 确认异常是否能够稳定复现。
          a. 如果异常稳定复现，请参见检查CoreDNS DNS查询请求日志、容器网络诊断和检查业务Pod到CoreDNS的网络连通性。
          b. 如果异常不能稳定复现，请参见抓包。
   - 如果使用了NodeLocal DNSCache，请参见NodeLocal DNSCache未生效和PrivateZone域名解析异常。
3. 如果以上排查无果，请提交工单排查。

常见客户端报错

客户端	报错日志	可能异常
ping	`ping: xxx.yyy.zzz: Name or service not known`	域名不存在或无法连接域名服务器。如果解析延迟大于5秒，一般是无法连接域名服务器。
curl	`curl: (6) Could not resolve host: xxx.yyy.zzz`	

客户端	报错日志	可能异常
PHP HTTP客户端	php_network_getaddresses: getaddrinfo failed: Name or service not known in xxx.php on line yyy	
Golang HTTP客户端	dial tcp: lookup xxx.yyy.zzz on 100.100.2.136:53: no such host	域名不存在。
dig	;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: xxxxx	
Golang HTTP客户端	dial tcp: lookup xxx.yyy.zzz on 100.100.2.139:53: read udp 192.168.0.100:42922->100.100.2.139:53: i/o timeout	无法连接域名服务器。
dig	;; connection timed out; no servers could be reached	

排查思路

排查思路	排查依据	问题及解决方案
按解析异常的域名类型排查	集群内外域名都异常	• 安全组、交换机ACL配置错误 • 容器网络连通性异常 • CoreDNS Pod负载高 • CoreDNS Pod负载不均 • CoreDNS Pod运行状态异常 • Conntrack表满 • AutoPath插件异常 • A记录和AAAA记录并发解析异常 • IPVS缺陷导致解析异常
	仅集群外部域名异常	集群外部域名自身异常
	仅PrivateZone 、vpc-proxy域名解析异常	PrivateZone域名解析异常
	仅Headless类型服务域名异常	• 新增Headless类型域名无法解析 • StatefulSets Pod域名无法解析
	完全无法解析	• CoreDNS Pod运行状态异常 • PrivateZone域名解析异常 • 安全组、交换机ACL配置错误 • 集群外部域名自身异常 • 新增Headless类型域名无法解析 • StatefulSets Pod域名无法解析 • 容器网络连通性异常

排查思路	排查依据	问题及解决方案
按解析异常出现频次排查	异常仅出现在业务高峰时期	• CoreDNS Pod负载高 • CoreDNS Pod负载不均
	异常出现频次非常高	• IPVS缺陷导致解析异常 • NodeLocal DNSCache未生效
	异常出现频次非常低	• AutoPath插件异常 • A记录和AAAA记录并发解析异常
	异常仅出现在节点扩缩容或CoreDNS缩容时	IPVS缺陷导致解析异常

7.6.7.2. 常见检查方法

本文介绍关于Kubernetes集群域名解析异常的常见检查命令和检查步骤。

- 检查业务Pod的DNS配置
- 检查CoreDNS Pod运行状态
- 检查CoreDNS运行日志
- 检查CoreDNS DNS查询请求日志
- 检查CoreDNS Pod的网络连通性
- 检查业务Pod到CoreDNS的网络连通性
- 容器网络诊断
- 抓包

检查业务Pod的DNS配置

- **命令**

```
#查看foo容器的YAML配置，并确认DNSPolicy字段是否符合预期。
kubectl get pod foo -o yaml
#当DNSPolicy符合预期时，可以进一步进入Pod容器中，查看实际生效的DNS配置。
#通过bash命令进入foo容器，若bash不存在可使用sh代替。
kubectl exec -it foo bash
#进入容器后，可以查看DNS配置，nameserver后面为DNS服务器地址。
cat /etc/resolv.conf
```

- **DNS Policy配置说明**
  DNS Policy示例如下所示。

```
apiVersion: v1
kind: Pod
metadata:
 name: <pod-name>
 namespace: <pod-namespace>
spec:
 containers:
 - image: <container-image>
   name: <container-name>
#默认场景下的DNS Policy。
 dnsPolicy: ClusterFirst
#使用了NodeLocal DNSCache时的DNS Policy。
 dnsPolicy: None
 dnsConfig:
   nameservers:
   - 169.254.20.10
   - 172.21.0.10
   options:
   - name: ndots
     value: "3"
   - name: timeout
     value: "1"
   - name: attempts
     value: "2"
   searches:
   - default.svc.cluster.local
   - svc.cluster.local
   - cluster.local
 securityContext: {}
 serviceAccount: default
 serviceAccountName: default
 terminationGracePeriodSeconds: 30
```

DNSPolicy字段值	使用的DNS服务器
Default	只适用于不需要访问集群内部服务的场景。Pod创建时会从ECS节点*/etc/resolv.conf*文件继承DNS服务器列表。
ClusterFirst	此为DNSPolicy默认值，Pod会将CoreDNS提供的kube-dns服务IP作为DNS服务器。开启HostNetwork的Pod，如果选择ClusterFirst模式，效果等同于Default模式。
ClusterFirstWithHostNet	开启HostNetwork的Pod，如果选择ClusterFirst模式，效果等同于Default模式。
None	配合DNSConfig字段，可用于自定义DNS服务器和参数。在NodeLocal DNSCache开启注入时，DNSConfig会将DNS服务器指向本地缓存IP及CoreDNS提供的kube-dns服务IP。

检查CoreDNS Pod运行状态

命令

- 执行以下命令，查看容器组信息。

```
kubectl -n kube-system get pod -o wide -l k8s-app=kube-dns
```

预期输出：

```
NAME           READY  STATUS   RESTARTS  AGE  IP      NODE
coredns-xxxxxxxxx-xxxxx  1/1    Running  0     25h  172.20.6.53  cn-hangzhou.192.168.0.198
```

- 执行以下命令，查看Pod的实时资源使用情况。

  ```
  kubectl -n kube-system top pod -l k8s-app=kube-dns
  ```

  预期输出：

  ```
  NAME           CPU(cores)  MEMORY(bytes)
  coredns-xxxxxxxxx-xxxxx  3m      18Mi
  ```

- 如果Pod不处于Running状态，可以通过 `kubectl -n kube-system describe pod <CoreDNS Pod名称>` 命令，查询问题原因。

检查CoreDNS运行日志

命令

执行以下命令，检查CoreDNS运行日志。

```
kubectl -n kube-system logs -f --tail=500 --timestamps coredns-xxxxxxxxx-xxxxx
```

参数	描述
`f`	持续输出。
`tail=500`	输出最后500行日志。
`timestamps`	同时显示日志打印的时间。
`coredns-xxxxxxxxx-xxxxx`	CoreDNS Pod副本的名称。

检查CoreDNS DNS查询请求日志

命令

DNS查询请求日志仅会在开启CoreDNS的Log插件后，才会打印到容器日志中。关于开启Log插件的具体操作，请参见ACK集群内DNS原理和配置说明。

命令与检查CoreDNS运行日志相同，请参见检查CoreDNS运行日志。

检查CoreDNS Pod的网络连通性

操作步骤

1. 登录CoreDNS Pod所在集群节点。
2. 执行 `ps aux | grep coredns` ，查询CoreDNS的进程ID。
3. 执行 `nsenter -t <pid> -n bash` ，进入CoreDNS所在容器网络命名空间，其中 `pid` 为上一步得到的进程ID。
4. 测试网络连通性。
   i. 运行 `telnet <apiserver_slb_ip> 443` ，测试Kubernetes APIServer的连通性。

      其中 `apiserver_slb_ip` 为default命名空间下Kubernetes服务的IP地址。

   ii. 运行 `dig <domain> @<upstream_dns_server_ip>` ，测试CoreDNS Pod到上游DNS服务器的连通性。

      其中 `domain` 为测试域名， `upstream_dns_server_ip` 为上游DNS服务器地址，默认为100.100.2.136和100.100.2.138。

常见问题

现象	原因	处理方案
CoreDNS无法连通Kubernetes APIServer	APIServe异常、机器负载高、kube-proxy 没有正常运行等。	可提交工单排查。
CoreDNS无法连通上游DNS服务器	机器负载高、CoreDNS配置错误、专线路由问题等。	可提交工单排查。

检查业务Pod到CoreDNS的网络连通性

操作步骤

1. 选择以下任意一种方式，进入客户端Pod容器网络。
   - 方法一：使用 `kubectl exec` 命令。
   - 方法二：
     a. 登录业务Pod所在集群节点。
     b. 执行 `ps aux | grep <业务进程名>` 命令，查询业务容器的进程ID。
     c. 执行 `nsenter -t <pid> -n bash` 命令，进入业务Pod所在容器网络命名空间。
     其中 `pid` 为上一步得到的进程ID。
   - 方法三：如果频繁重启，请按以下步骤操作。
     a. 登录业务Pod所在集群节点。
     b. 执行 `docker ps -a | grep <业务容器名>` 命令，查询 `k8s_POD_` 开头的沙箱容器，记录容器ID。
     c. 执行 `docker inspect <沙箱容器 ID> | grep netns` 命令，查询*/var/run/docker/netns/xxxx*的容器网络命名空间路径。
     d. 执行 `nsenter -n<netns 路径> bash` 命令，进入容器网络命名空间。
     其中 `netns 路径` 为上一步得到的路径。

     > 说明 `-n` 和 `<netns 路径>` 之间不加空格。

2. 测试网络连通性。
   i. 执行 `dig <domain> @<kube_dns_svc_ip>` 命令，测试业务Pod到CoreDNS服务kube-dns解析查询的连通性。
   其中 `<domain>` 为测试域名，`<kube_dns_svc_ip>` 为kube-system命名空间中kube-dns的服务IP。
   ii. 执行 `ping <coredns_pod_ip>` 命令，测试业务Pod到CoreDNS容器副本的连通性。
   其中 `<coredns_pod_ip>` 为kube-system命名空间中CoreDNS Pod的IP。
   iii. 执行 `dig <domain> @<coredns_pod_ip>` 命令，测试业务Pod到CoreDNS容器副本解析查询的连通性。
   其中 `<domain>` 为测试域名，`<coredns_pod_ip>` 为kube-system命名空间中CoreDNS Pod的IP。

常见问题

现象	原因	处理方案
业务Pod无法通过CoreDNS服务kube-dns解析	机器负载高、kube-proxy没有正常运行、安全组没有放开UDP协议53端口等。	检查安全组是否放开UDP 53端口，若已放开请提交工单排查。
业务Pod无法连通CoreDNS容器副本	容器网络异常或安全组没有放开ICMP。	运行容器网络诊断。
业务Pod无法通过CoreDNS容器副本解析	机器负载高、安全组没有放开UDP协议53端口等。	检查安全组是否放开UDP 53端口，若已放开请提交工单排查。

容器网络诊断

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在控制台左侧导航栏中，单击**集群**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 集群检查**。
5. 在**容器智能运维**的左侧导航栏，选择**检查 > 故障诊断**。
6. 在**故障诊断**页面，单击**网络诊断**页签。
7. 在**源地址**输入业务Pod IP，在**目标地址**输入CoreDNS服务kube-dns IP，在**端口**输入53，同时选中**报文跟踪**，选中**我已知晓并同意**，单击**发起诊断**。
8. 在故障诊断列表，单击目标诊断项右侧**操作**列的**诊断详情**。

   根据**诊断结果**、**访问全图**及**展开所有访问路径**查看网络诊断详情，诊断结果将包含异常原因。关于网络诊断的具体操作，请参见通过集群故障诊断功能定位集群问题。

抓包

当无法定位问题时，需要抓包进行辅助诊断。

1. 登录出现异常的业务Pod、CoreDNS Pod所在节点。
2. 在ECS（非容器内）执行以下命令，可以将最近所有的53端口信息抓取到文件中。

   ```
   tcpdump -i any port 53 -C 20 -W 200 -w /tmp/client_dns.pcap
   ```

3. 结合业务日志的报错定位到精准的报错时间的报文信息。

   > **说明**
   > - 在正常情况下，抓包对业务无影响，仅会增加小部分的CPU负载和磁盘写入。
   > - 以上命令会对抓取到的包进行rotate，最多可以写200个20 MB的.pcap文件。

相关文档

- ACK集群内DNS原理和配置说明
- 分析和监控CoreDNS日志
- CoreDNS升级说明
- 使用ACK CoreDNS DNSTAP Analyser诊断CoreDNS域名解析异常
- 在ACK集群中使用NodeLocal DNSCache

7.6.7.3. 常见问题及解决方案

本文介绍了关于Kubernetes集群内域名解析异常的常见问题及解决方案。

- 集群外部域名自身异常
- 新增Headless类型域名无法解析
- StatefulSets Pod域名无法解析
- 安全组、交换机ACL配置错误
- 容器网络连通性异常
- CoreDNS Pod负载高
- CoreDNS Pod负载不均
- CoreDNS Pod运行状态异常
- 客户端负载原因导致解析失败

- Conntrack表满
- AutoPath插件异常
- A记录和AAAA记录并发解析异常
- IPVS缺陷导致解析异常
- NodeLocal DNSCache未生效
- PrivateZone域名解析异常

集群外部域名自身异常

问题原因

上游服务器返回记录显示域名异常。

问题现象

业务Pod可以正常解析集群内部域名，但无法解析某些集群外部域名。

解决方案

检查CoreDNS DNS查询请求日志。

常见请求日志

CoreDNS接收到请求并回复客户端后会打印一行日志，示例如下：

```
# 其中包含状态码RCODE NOERROR，代表解析结果正常返回。
[INFO] 172.20.2.25:44525 - 36259 "A IN redis-master.default.svc.cluster.local. udp 56 false 512" NOERROR qr,aa,rd 11
0 0.000116946s
```

常见返回码RCODE

关于返回码RCODE定义的具体信息，请参见规范。

返回码RCODE	含义	原因
NXDOMAIN	域名不存在	容器内请求域名时，会被拼接上search后缀，若拼接的结果域名不存在，则会出现该请求码。如果确认日志中请求的域名内容存在，则说明存在异常。
SERVFAIL	上游服务器异常	常见于无法连接上游DNS服务器等情况。
REFUSED	拒绝应答	常见于CoreDNS配置或集群节点*/etc/resolv.conf*文件指向的上游DNS服务器无法处理该域名的情况，请排查CoreDNS配置文件。

新增Headless类型域名无法解析

问题原因

1.7.0以前版本CoreDNS会在APIServer抖动时异常退出，导致Headless域名停止更新。

问题现象

接入CoreDNS的业务Pod无法解析新增的Headless类型域名。

解决方案

升级CoreDNS至1.7.0以上。具体操作，请参见【组件升级】CoreDNS升级公告。

StatefulSets Pod域名无法解析

问题原因

StatefulSets Pod YAML中ServiceName必须和其暴露SVC的名字一致，否则无法访问Pod域名（例如pod.headless-svc.ns.svc.cluster.local），只能访问到服务域名（例如headless-svc.ns.svc.cluster.local）。

问题现象
Headless服务无法通过Pod域名解析。

解决方案
修改StatefulSets Pod YAML中ServiceName名称。

安全组、交换机ACL配置错误

问题原因
修改了ECS或容器使用的安全组（或交换机ACL），拦截了UDP协议下53端口的通信。

问题现象
部分节点或全部节点上接入CoreDNS的业务，Pod解析域名持续性失败。

解决方案
恢复安全组、交换机ACL的配置，放开其以UDP协议对53端口的通信。

容器网络连通性异常

问题原因
由于容器网络或其它原因导致的UDP协议53端口持续性不通。

问题现象
部分节点或全部节点上接入CoreDNS的业务，Pod解析域名持续性失败。

解决方案
运行容器网络诊断。具体操作，请参见通过集群故障诊断功能定位集群问题。

CoreDNS Pod负载高

问题原因
由于CoreDNS副本数不足、业务请求量高等情况导致的CoreDNS负载高。

问题现象

- 部分节点或全部节点接入CoreDNS的业务，Pod解析域名的延迟增加、概率性或持续性失败。
- 检查CoreDNS Pod运行状态发现各副本CPU、Memory使用量接近其资源限制。

解决方案

- 考虑采用NodeLocal DNSCache缓存方案，提升DNS解析性能，降低CoreDNS负载。具体操作，请参见在ACK集群中使用NodeLocal DNSCache。
- 适当扩充CoreDNS副本数，使每个Pod的峰值CPU始终低于节点空闲CPU数。

CoreDNS Pod负载不均

问题原因
由于CoreDNS副本调度不均、Service亲和性设置导致CoreDNS Pod负载不均衡。

问题现象

- 部分接入CoreDNS的业务Pod解析域名的延迟增加、概率性或持续性失败。
- 检查CoreDNS Pod运行状态发现各副本CPU使用量负载不均衡。
- CoreDNS副本数少于两个，或多个CoreDNS副本位于同节点上。

解决方案

- 扩容并打散CoreDNS副本到不同的节点上。
- 负载不均衡时，可禁用kube-dns服务的亲和性属性。具体操作，请参见配置Kube-DNS服务。

CoreDNS Pod运行状态异常

问题原因
由于CoreDNS YAML模板、配置文件等导致CoreDNS运行异常。

问题现象

- 部分接入CoreDNS的业务Pod解析域名的延迟增加、概率性或持续性失败。
- CoreDNS副本状态Status不处于Running状态，或重启次数RESTARTS持续增加。
- CoreDNS运行日志中出现异常。

解决方案
检查CoreDNS Pod运行状态和运行日志。

常见异常日志及处理方案

日志中字样	原因	处理方案
`/etc/coredns/Corefile:4 - Error during parsing: Unknown directive 'ready'`	配置文件和CoreDNS不兼容，`Unknown directive` 代表当前运行的CoreDNS版本不支持ready插件。	从kube-system命名空间中CoreDNS配置项中删除ready插件，其它报错同理。
`pkg/mod/k8s.io/client-go@v0.18.3/tools/cache/reflector.go:125: Failed to watch *v1.Pod: Get "https://192.168.0.1:443/api/v1/": dial tcp 192.168.0.1:443: connect: connection refused`	日志出现时间段内，APIServer中断。	如果是日志出现时间和异常不吻合，可以排除该原因，否则请检查CoreDNS Pod网络连通性。具体操作，请参见检查CoreDNS Pod的网络连通性。
`[ERROR] plugin/errors: 2 www.alivun.com. A: read udp 172.20.6.53:58814->100.100.2.136:53: i/o timeout`	日志出现时间段内，CoreDNS无法连接到上游DNS服务器。	

客户端负载原因导致解析失败

问题原因
接入CoreDNS的业务Pod所在ECS负载达到100%等情况导致UDP报文丢失。

问题现象
业务高峰期间或突然偶发的解析失败，ECS监控显示机器网卡重传率、CPU负载有异常。

解决方案

- 建议提交工单排查原因。
- 考虑采用NodeLocal DNSCache缓存方案，提升DNS解析性能，降低CoreDNS负载。具体操作，请参见在ACK集群中使用NodeLocal DNSCache。

Conntrack表满

问题原因
Linux内Conntrack表条目有限，无法进行新的UDP或TCP请求。

问题现象

- 部分节点或全部节点上接入CoreDNS的业务，Pod解析域名在业务高峰时间段内出现大批量域名解析失败，高峰结束后失败消失。
- 运行 `dmesg -H` ，滚动到问题对应时段的日志，发现出现 `conntrack full` 字样的报错信息。

解决方案
增加Conntrack表限制。具体操作，请参见如何提升Linux连接跟踪Conntrack数量限制？。

AutoPath插件异常

问题原因
CoreDNS处理缺陷导致AutoPath无法正常工作。

问题现象

- 解析集群外部域名时，概率性解析失败或解析到错误的IP地址，解析集群内部域名无异常。
- 高频创建容器时，集群内部服务域名解析到错误的IP地址。

解决方案

关闭AutoPath插件。

1. 执行 `kubectl -n kube-system edit configmap coredns` 命令，打开CoreDNS配置文件。
2. 删除 `autopath @kubernetes` 一行后保存退出。
3. 检查CoreDNS Pod运行状态和运行日志，运行日志中出现 `reload` 字样后说明修改成功。

A记录和AAAA记录并发解析异常

问题原因

并发A和AAAA的DNS请求触发Linux内核Conntrack模块缺陷，导致UDP报文丢失。

问题现象

- 接入CoreDNS的业务Pod解析域名概率性失败。
- 从抓包或检查CoreDNS DNS查询请求日志可以发现，A和AAAA通常在同一时间的出现，并且请求的源端口一致。

解决方案

- 考虑采用NodeLocal DNSCache缓存方案，提升DNS解析性能，降低CoreDNS负载。具体操作，请参见在ACK集群中使用NodeLocal DNSCache。
- CentOS、Ubuntu等基础镜像，可以通过 `options timeout:2 attempts:3 rotate single-request-reopen` 等参数优化。
- 如果容器镜像是以Alpine制作的，建议更换基础镜像。更多信息，请参见Alpine。
- PHP类应用短连接解析问题较多，如果使用的是PHP Curl的调用，可以使用 `CURL_IPRESOLVE_V4` 参数仅发送IPv4解析。更多信息，请参见函数说明。

IPVS缺陷导致解析异常

问题原因

在CoreDNS Pod从IPVS服务后端摘除时，DNS请求的源端口复用会导致IPVS丢包。

问题现象

当集群节点扩缩容、节点关机、CoreDNS缩容时，出现概率性解析失败，通常时长在五分钟左右。

解决方案

- 考虑采用NodeLocal DNSCache缓存方案，可以容忍IPVS丢包。具体操作，请参见在ACK集群中使用NodeLocal DNSCache。
- 优化IPVS UDP超时时间。具体操作，请参见配置IPVS类型集群的UDP超时时间。

NodeLocal DNSCache未生效

问题原因

- 未配置DNSConfig注入，业务Pod实际仍配置了CoreDNS kube-dns服务IP作为DNS服务器地址。
- 业务Pod采用Alpine作为基础镜像，Alpine基础镜像会并发请求所有nameserver，包括本地缓存和CoreDNS。

问题现象

NodeLocal DNSCache没有流量进入，所有请求仍在CoreDNS上。

解决方案

- 配置DNSConfig自动注入。具体操作，请参见在ACK集群中使用NodeLocal DNSCache。
- 如果容器镜像是以Alpine制作的，建议更换基础镜像。更多信息，请参见Alpine。

PrivateZone域名解析异常

问题原因

PrivateZone不支持TCP协议，需要使用UDP协议访问。

问题现象
对于接入NodeLocal DNSCache的业务，Pod无法解析PrivateZone上注册的域名，或无法解析包含vpc-proxy字样的阿里云云产品API域名，或解析结果不正确。

解决方案
在CoreDNS中配置 `prefer_udp` 。具体操作，请参见CoreDNS修改说明。

7.7. 网络管理FAQ

本文汇总网络管理中容器网络、Service和Ingress的常见问题。

容器网络FAQ

- Terway网络模式下增加了虚拟交换机后，集群无法访问公网怎么办？
- 手动升级了Flannel镜像版本后，如何解决无法兼容1.16以上版本集群的问题？
- 如何解决Pod启动后存在时延的问题？
- 如何让Pod访问自己暴露的服务？
- 如何选择Kubernetes集群Terway和Flannel网络插件？
- 如何规划集群网络？
- ACK是否支持hostPort的端口映射？
- ACK集群是否支持VPC多路由表？
- 如何查看集群的网络类型及对应的虚拟交换机？
- 如何查看集群中使用的云资源？
- Kubernetes集群中Pod出现网络异常
- 如何确认Kubernetes集群内应用的出网地址
- 使用Terway网络的ENI模式出现网络异常
- Kubernetes管控集群链路异常的排查思路
- Terway网络场景中交换机的IP资源不足
- Kubernetes集群中访问LoadBalancer暴露出去的SLB地址不通
- 如何修改kube-proxy配置？
- 如何提升Linux连接跟踪Conntrack数量限制？
- 如何修改kube-proxy中IPVS负载均衡模式？
- 如何修改kube-proxy中IPVS UDP会话保持的超时时间？
- 如何解决IPv6双栈的部分常见问题？

Service FAQ

SLB相关

- 为什么看不到Service与LoadBalancer同步过程的事件Event信息？
- SLB创建一直处于Pending状态如何处理？
- SLB虚拟服务器组未更新如何处理？
- Service注解不生效如何处理？
- 为何SLB的配置被修改？
- 为什么集群内无法访问SLB IP？
- 误删除SLB怎么办？
- 删除Service是否会删除SLB？
- 旧版本CCM如何支持SLB重命名？
- Local模式下如何自动设置Node权重？

CCM升级相关

- CCM升级失败如何处理？

使用已有SLB常见问题

- 为什么复用已有SLB没有生效？
- 为什么复用已有SLB时没有配置监听？

其他

- Kubernetes的Service如何进行会话保持？

Ingress FAQ

- Ingress支持哪些SSL/TLS版本？
- Ingress L7请求头默认是透传的吗？
- 后端服务为HTTPS服务访问时是否可以通过Ingress-Nginx转发？
- Ingress L7透传客户端IP吗？
- nginx-ingress-controller组件支持HSTS吗？
- Ingress-Nginx支持哪些Rewrite配置？
- 当版本升级后SLS解析日志不正常怎样修复？
- 部署Ingress Controller使用私网SLB
- 在ACK组件管理中升级Nginx Ingress Controller组件时，系统所做的更新是什么？
- ingress-nginx如何使用已有的SLB？
- 如何将Ingress-nginx的监听由四层改为七层（HTTPS/HTTP）？
- 创建Ingress时报错“certificate signed by unknown authority”

8.应用

8.1. 工作负载

8.1.1. 创建无状态工作负载Deployment

您可以通过镜像、编排模板及命令创建一个无状态应用。容器服务Kubernetes集群支持通过Web界面使用镜像密钥。本文以Nginx为例说明如何通过镜像、编排模板及命令来创建无状态应用。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已通过kubectl连接kubernetes集群。具体操作，请参见通过Kubectl连接Kubernetes集群。
- 已构建私有镜像仓库，并将镜像上传到该仓库中。本例中以阿里云容器镜像服务为示例。具体操作，请参见管理镜像。

使用镜像创建无状态Deployment应用

步骤一：配置应用基本信息

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用镜像创建**。
6. 在**应用基本信息**配置向导页面中，设置应用的基本信息。

   在配置Deployment负载前，您需在页面上方设置应用所在的命名空间。本例设置命名空间为**default**。

参数	描述
应用名称	设置应用的名称。
副本数量	即应用包含的Pod数量，默认数量为2。
类型	定义资源对象的类型，可选择**无状态**、**有状态**、**任务**、**定时**、**定时任务**、**守护进程集**。
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）。
时区同步	容器与节点是否使用相同的时区。

> **说明** 本例中选择**无状态**（Deployment）类型。

7. 单击**下一步**。

   进入**容器配置**配置向导页面。

步骤二：配置容器

在**容器配置**配置向导页面中，配置容器的镜像与资源、端口、环境变量、健康检查、生命周期、数据卷和日志。

> **说明** 在**容器1**页签右侧，单击**添加容器**为应用的Pod设置多个容器。

1. 在**基本配置**区域，完成容器的基本配置。

参数	描述
镜像名称	○ 您可以单击**选择镜像**，选择所需的镜像并单击**确定**。本例中为官方Nginx镜像，可以在**搜索**页签中，选择**Docker镜像**，然后在文本框中输入*nginx*进行搜索。 ■ 容器镜像服务：在**容器镜像服务**页签中，可以选择托管在容器镜像服务ACR中的镜像。需要选择镜像所属地域，以及镜像服务实例。关于ACR的更多信息，请参见什么是容器镜像服务ACR。 **说明** 您可以在**容器镜像服务**页签，根据名称搜索托管在容器镜像服务ACR中的镜像。 ■ Docker官方镜像：在**Docker官方镜像**页签中，可以选择Docker官方提供的容器镜像。 ■ 用户收藏：在**用户收藏**页签中，可以选择您收藏的Docker官方镜像。 ■ 搜索：在**搜索**页签中，可以分别在ACR的特定地域和Docker官方镜像中搜索容器镜像。 ○ 您还可以填写私有镜像。填写的格式为 `domainname/namespace/imagename:tag`。
镜像Tag	○ 您可以单击**选择镜像Tag**选择镜像的版本。若不指定，默认为最新版。 ○ ACK支持以下三种镜像拉取策略（imagePullPolicy）： ■ **优先使用本地镜像（IfNotPresent）**：如果本地有该镜像（之前拉取过该镜像至宿主机中），则使用本地镜像，本地不存在时拉取镜像。 ■ **总是拉取镜像（Always）**：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。 ■ **仅使用本地镜像（Never）**：仅使用本地镜像。 **说明** 当您选中**镜像拉取策略**时，默认不设置镜像拉取策略。 ○ 单击**设置镜像密钥**，您可以实现免密拉取镜像。具体操作，请参见使用免密组件拉取容器镜像。
资源限制	可指定该应用所能使用的资源上限，包括CPU、内存和ephemeral-storage三种资源，防止占用过多资源。
所需资源	即为该应用预留资源额度，包括CPU和内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争夺资源，导致应用不可用。
容器启动项	○ stdin：表示为该容器开启标准输入。 ○ 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。例如，一个交互式的程序从用户获取标准输入，并显示到终端中。
特权容器	○ 选择特权容器，则privileged=true，开启特权模式。 ○ 不选择特权容器，则privileged=false，关闭特权模式。

参数	描述
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。具体操作，请参见Init Containers。

2. （可选）在**端口设置**区域，单击**新增**设置容器的端口。

参数	描述
名称	设置容器端口名称。
容器端口	设置暴露的容器访问端口或端口名，端口号必须介于1~65535。
协议	支持TCP和UDP。

3. （可选）在**环境变量**区域，单击**新增**设置环境变量。

   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，具体操作，请参见Pod variable。

参数	描述
类型	设置环境变量的类型，支持： ○ **自定义** ○ **配置项** ○ **保密字典** ○ **变量/变量引用** ○ **资源引用** 配置项、保密字典支持全部文件的引用，以保密字典为例。选择**保密字典**类型，只选择目标保密字典，则默认引用全部文件。  对应的*YAML*，也引用了整个密钥（Secret）。 
变量名称	设置环境变量名称。
变量/变量引用	设置变量引用的值。

4. （可选）在**健康检查**区域，根据需要开启**存活检查**、**就绪检查**及**启动探测**。

   更多信息，请参见配置存活、就绪和启动探测器。

参数	请求类型	配置说明
○ 存活检查（Liveness）：用于检测何时重启容器。 ○ 就绪检查（Readiness）：确定容器是否已经就绪，且可以接受流量。 ○ 启动探测（Startup Probes）：用于检测何时启动容器。 **说明** 仅Kubernetes集群1.18及之后版本支持启动探测。	HTTP请求	即向容器发送一个HTTP Get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HTTP Headers，HTTP请求中自定义的请求头，HTTP允许重复的Header。支持键值对的配置方式。 ○ 延迟探测时间（秒）：即initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为3秒。 ○ 执行探测频率（秒）：即periodSeconds，指执行探测的时间间隔，默认为10秒，最小为1秒。 ○ 超时时间（秒）：即timeoutSeconds，探测超时时间。默认1秒，最小1秒。 ○ 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 ○ 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3，最小值是1。
	TCP连接	即向容器发送一个TCP Socket，Kubelet将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： ○ 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 ○ 延迟探测时间（秒）：即initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为15秒。 ○ 执行探测频率（秒）：即periodSeconds，指执行探测的时间间隔，默认为10秒，最小为1秒。 ○ 超时时间（秒）：即timeoutSeconds，探测超时时间。默认1秒，最小1秒。 ○ 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 ○ 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3，最小值是1。

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

5. （可选）在**生命周期**区域，设置容器的生命周期。

   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。具体操作，请参见配置生命周期。



参数	描述
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。

6. （可选）在**数据卷**区域，增加本地存储或云存储声明PVC（Persistent Volume Claim）。

参数	描述
增加本地存储	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，请参见volumes。
增加云存储声明（PersistentVolumeClaim）	支持通过PVC挂载云存储卷。在选择目标挂载源前，您需要创建云存储声明。具体操作，请参见创建持久化存储卷声明。

本例中配置了一个云存储类型的数据卷声明disk-ssd，将其挂载到容器的*/tmp*路径下。



7. （可选）在**日志配置**区域，进行采集配置和自定义Tag设置。

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

配置项	描述
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：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。
自定义Tag	您还可以设置自定义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。

8. 单击**下一步**。

进入**高级配置**配置向导页面。

步骤三：完成高级配置

在**高级配置**配置向导页面中设置访问、伸缩、调度和标签注解。

1. 在**访问设置**区域，设置暴露后端Pod的方式。

> **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，您可以在创建服务时，根据需要选择**虚拟集群IP**或**节点端口**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。
>   - 创建负载均衡类型的服务：您可以在创建服务时，选择**负载均衡**类型的服务。通过阿里云提供的负载均衡服务SLB（Server Load Balancer），使得该服务提供公网访问能力。
>   - 创建路由（Ingress）：通过创建路由（Ingress）提供公网访问能力。更多信息，请参见Ingress。

您可以设置暴露后端Pod的方式。本例中选择虚拟集群IP和路由（Ingress），构建一个公网可访问的Nginx应用。

- 配置服务（Service）：在**服务（Service）**右侧，单击**创建**设置创建服务配置项。

配置项	描述
名称	输入服务的名称。 本例为nginx-svc。
类型	选择服务类型，即服务访问的方式。 ■ **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。 **说明** 您的服务类型为**虚拟集群IP**时，才能设置**实例间发现服务**（Headless Service）。 ■ **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，该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），可选择公网访问或私网访问。阿里云负载均衡服务可以路由到NodePort服务和ClusterIP服务。 ■ 新建SLB：您可以通过单击**修改**，修改SLB规格。 ■ 使用已有SLB：您可以在已有的列表中选择SLB规格。 **说明** 负载均衡类型支持新建SLB和使用已有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。 本例中选择**虚拟集群IP**。
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。
外部流量策略	■ Local：流量只发给本机的Pod。 ■ Cluster：流量可以转发到其他节点上的Pod。 **说明** 您的服务类型为**节点端口**或**负载均衡**时，才能设置**外部流量策略**。

配置项	描述
注解	为该服务添加一个注解（Annotation），配置负载均衡的参数。例如设置 service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth:20 表示将该服务的带宽峰值设置为20Mbit/s，从而控制服务的流量。更多参数，请参见通过Annotation配置负载均衡。
标签	为该服务添加一个标签，标识该服务。

- 配置路由（Ingress）：在**路由（Ingress）**右侧，单击**创建**设置后端Pod的路由规则。
  关于路由配置信息，请参见配置路由信息。

  注意 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*文件中添加一条域名映射（*Ingress外部端点* + *Ingress域名*）。在实际工作场景中，请使用备案域名。

  ```
  101.37.XX.XX  foo.bar.com  #即Ingress的IP。
  ```

配置项	描述
名称	输入路由的名称。 本例为nginx-ingress。
规则	路由规则是指授权入站到达集群服务的规则。更多信息，请参见配置路由信息。 ■ **域名**：输入Ingress域名。 ■ **路径**：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**：选择服务的名称和对应端口。 ■ **开启TLS**：配置安全的路由服务。具体操作，请参见Ingress高级用法。 本例中使用测试域名 foo.bar.com ；设置服务为nginx-svc。
服务权重	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。

配置项	描述
灰度发布策略	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。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。 ■ 基于Request Header的流量切分 ■ 基于Cookie的流量切分 ■ 基于Query Param的流量切分 **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。 配置参数如下所示： ■ **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**：您自定义的请求字段，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。
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。 ■ 您也可以单击**添加**按钮，输入注解名称和值，即Ingress的Annotation键值对，关于Ingress的注解，请参见Annotations。
标签	为Ingress添加对应的标签，标示该Ingress的特点。

在**访问设置**区域，您可以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。

2. （可选）在**伸缩配置**区域，配置是否开启**指标伸缩**和**定时伸缩**，从而满足应用在不同负载下的需求。
   - 容器服务支持容器组的指标伸缩，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。

参数	描述
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	该负载类型可扩容的容器数量上限。
最小副本数量	该负载类型可缩容的容器数量下限。

   - 容器服务支持容器组的定时伸缩，即定时地对容器服务Kubernetes集群进行扩缩容。开启定时伸缩前，需先安装ack-kubernetes-cronhpa-controller组件。关于定时伸缩的详细说明，请参见创建定时伸缩CronHPA任务。
3. （可选）在**调度设置**区域，设置**升级方式**、**节点亲和性**、**应用亲和性**和**应用非亲和性**，更多信息，请参见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。

配置项	描述
升级方式	升级方式包括滚动升级（rollingupdate）和替换升级（recreate），详细请参见详情请参见Deployments。
节点亲和性	设置**节点亲和性**，通过Worker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Worker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。调度会尽量调度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。
应用亲和性	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。

配置项	描述
应用非亲和性	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。
调度容忍	容忍被应用于Pod，允许这个Pod被调度到相对应的污点上。
调度到虚拟节点	设置是否调度到虚拟节点。如果您集群中没有虚拟节点，则无法设置该配置项。

4. （可选）在**标签和注释**区域，单击**添加**设置容器组的标签和注释。

参数	描述
Pod标签	为该Pod添加一个标签，标识该应用。
Pod注解	为该Pod添加一个注解（Annotation）。

5. 单击**创建**。

步骤四：查看应用

在**创建完成**配置向导页面中查看应用任务。

1. 在 **创建应用任务已提交**面板中单击**查看应用详情**。



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**，也可以创建服务。
- 单击路由右侧的**创建**，也可以创建路由。

2. 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。



3. 在浏览器中访问路由测试域名，您可访问Nginx欢迎页。



相关操作

查看无状态工作负载详情

在左侧导航栏单击**集群**，单击目标集群名称或目标集群**操作**列下的**详情**，选择**工作负载 > 无状态**，在**无状态**页面单击目标应用名称或目标应用**操作**列下的**详情**。

> **说明** 在无状态列表中单击**标签**字段，输入对应应用标签的key与 value ，单击**确定**后可过滤列表中的应用。

在应用详情页面您可以编辑、伸缩、查看YAML、重新部署、及刷新应用。

操作项	描述
编辑	在应用详情页面单击**编辑**，您可以修改应用信息。
伸缩	在应用详情页面单击**伸缩**，您可以修改所需容器组数量。
查看YAML	在应用详情页面单击**查看Yaml**，您可以**更新**、**下载**、**另存为**YAML文件。
重新部署	在应用详情页面单击**重新部署**，您可以重新部署应用。
刷新	在应用详情页面单击**刷新**，您可以刷新应用。

设置已创建的无状态工作负载

在**无状态**页面的目标应用**操作**列下，单击**更多**后，您可以进行以下操作：

操作项	描述
查看Yaml	查看应用的YAML内容。
重新部署	重新部署应用。
编辑标签	为应用设置标签。
节点亲和性	为应用设置节点亲和性。更多信息，请参见调度配置。
弹性伸缩	为应用设置弹性伸缩。更多信息，请参见容器水平伸缩（HPA）和容器定时伸缩（CronHPA）。
调度容忍	设置应用的容忍策略。更多信息，请参见调度配置。

操作项	描述
升级策略	设置应用升级的策略： • **滚动升级**：分批次逐步替换应用Pod实例。 • **替换升级**：先暂停老版应用Pod，用新版Pod替换。
复制创建	复制应用相同的容器设置。
回滚	回滚至应用之前发布的版本。
日志	查看应用的日志信息。
删除	删除应用。

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。

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。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态（Deployment）**页面，单击右上角的**使用YAML创建资源**。
6. 对模板进行相关配置，完成配置后单击**创建**。
   - **命名空间**：在**创建**页面的顶部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。
   - **添加工作负载**：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。
   - **保存模板**：您可以保存设置好的编排模板。

   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。
   > - （可选）在应用中通过Volume方式挂载存储卷时，默认情况下挂载目录下的文件在挂载后将会被覆盖。若要不影响挂载目录下的其他文件，您可以配置 `subPath` 参数。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-basic
  labels:
    app: nginx
spec:
```

```
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
        volumeMounts:
          - name: nginx-config
            mountPath: /etc/nginx/nginx.conf
            subPath: nginx.conf  #设置subPath。
    volumes:
      - name: nginx-config
        configMap:
          name: nginx-conf
---
apiVersion: v1
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。
---
#存储卷的配置项
apiVersion: v1
kind: ConfigMap
metadata:
 name: nginx-conf
 namespace: default
data:
 nginx.conf: |-
  user  nginx;
  worker_processes  1;
  error_log  /var/log/nginx/error.log warn;
  pid      /var/run/nginx.pid;
  events {
     worker_connections  1024;
  }
  http {
     include     /etc/nginx/mime.types;
     default_type  application/octet-stream;
```

```
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;
}
```

7. 单击**创建**后。会提示部署状态信息。

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。

通过命令创建应用

1. 执行以下命令启动容器（本示例中为Nginx Web服务器）。

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

2. 执行以下命令，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-d****       1/1    Running  1     9h
```

使用镜像密钥创建应用

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。

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。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面，设置命名空间，然后单击右上角**使用镜像创建**。
6. 在**应用基本信息**页面设置相关参数。更多信息，请参见应用基本信息。
7. 设置容器配置。

   本例仅对容器镜像密钥的配置进行说明，关于容器配置说明，请参见容器基本配置。

   i. 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。

ii. 在镜像版本栏中，输入私有镜像地址版本。

本文示例使用Tomcat私有镜像。

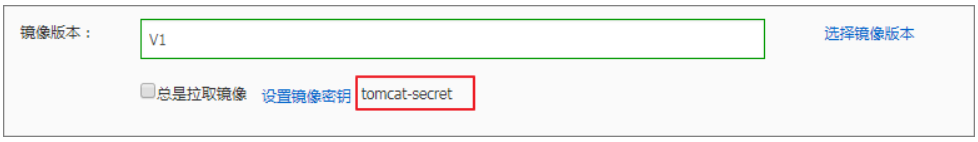


iii. 单击**设置镜像密钥**，在弹出的对话框中进行设置。

- 选择**新建密钥**，您需要配置以下参数。

参数	描述
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（包括阿里云主账号和子账号）。
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，即容器镜像独立登录密码。
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，具体操作，请参见如何支持私有镜像和使用私有镜像仓库创建应用。



8. 根据页面引导，完成其他配置，最后单击**创建**。

更多信息，请参见本文的步骤三：完成高级配置。

9. 在控制台左侧导航栏中，单击**集群**。
10. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
11. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

12. 在**无状态**页面，查看目标应用运行情况。
    您可看到，Tomcat应用正常运行，成功通过密钥使用Tomcat私有镜像。

相关文档

- 创建有状态工作负载StatefulSet
- 通过Annotation配置负载均衡

8.1.2. 创建有状态工作负载StatefulSet

阿里云容器服务Kubernetes版ACK（Container Service for 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。

体验实验室

为了帮助您快速上手本教程，ACK为您提供了可以动手操作的交互式实验环境。如您需要体验，请访问使用kubectl进行StatefulSet应用部署。

前提条件

使用镜像创建有状态StatefulSet应用前，请确保您已经进行以下操作：

- 创建Kubernetes托管版集群
- 创建持久化存储卷声明
- 通过kubectl工具连接集群

背景信息

StatefulSet包括以下特性：

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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 有状态**。
5. 在**有状态**页面中，单击**使用镜像创建**。
6. 在**应用基本信息**页签中，设置应用的基本信息。
   本例中选择**有状态**类型，创建StatefulSet类型的应用。

配置项	描述
应用名称	设置应用的名称。

配置项	描述
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	应用包含的Pod数量。
类型	定义资源对象的类型，可选择**无状态**、**有状态**、**任务**、**定时**、**定时任务**、**守护进程集**。
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）。
时区同步	容器与节点是否使用相同的时区。

7. 单击**下一步**进入**容器配置**页签。
8. 设置容器相关的配置。

> **说明** 在**容器配置**页签上方，单击**添加容器**为应用的Pod设置多个容器。

容器配置的配置项及描述如下。

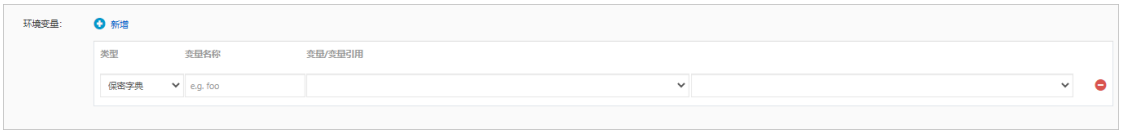
- 基本配置

配置项	描述
镜像名称	■ 您可以单击**选择镜像**，选择所需的镜像并单击**确定**。本例中为官方Nginx镜像，可以在**搜索**页签中，选择**Docker镜像**，然后在文本框中输入*nginx*进行搜索。 ■ 容器镜像服务：在**容器镜像服务**页签中，可以选择托管在容器镜像服务ACR中的镜像。需要选择镜像所属地域，以及镜像服务实例。关于ACR的更多信息，请参见什么是容器镜像服务ACR。 **说明** 您可以在**容器镜像服务**页签，根据名称搜索托管在容器镜像服务ACR中的镜像。 ■ Docker官方镜像：在**Docker官方镜像**页签中，可以选择Docker官方提供的容器镜像。 ■ 用户收藏：在**用户收藏**页签中，可以选择您收藏的Docker官方镜像。 ■ 搜索：在**搜索**页签中，可以分别在ACR的特定地域和Docker官方镜像中搜索容器镜像。 ■ 您还可以填写私有镜像。填写的格式为 `domainname/namespace/imagename:tag` 。

配置项	描述
镜像Tag	■ 您可以单击**选择镜像Tag**选择镜像的版本。若不指定，默认为最新版。 ■ ACK支持以下三种镜像拉取策略（imagePullPolicy）： ■ **优先使用本地镜像（IfNotPresent）**：如果本地有该镜像（之前拉取过该镜像至宿主机中），则使用本地镜像，本地不存在时拉取镜像。 ■ **总是拉取镜像（Always）**：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。 ■ **仅使用本地镜像（Never）**：仅使用本地镜像。 **说明** 当您选中**镜像拉取策略**时，默认不设置镜像拉取策略。 ■ 单击**设置镜像密钥**，您可以实现免密拉取镜像。具体操作，请参见使用免密组件拉取容器镜像。
资源限制	可指定该应用所能使用的资源上限，包括CPU、内存和Ephemeral-storage三种资源，防止占用过多资源。
所需资源	即为该应用预留资源额度，包括CPU和内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争夺资源，导致应用不可用。
容器启动项	■ stdin：表示为该容器开启标准输入。 ■ 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。例如，一个交互式的程序从用户获取标准输入，并显示到终端中。
特权容器	■ 选择特权容器，则 `privileged=true` ，开启特权模式。 ■ 不选择特权容器，则 `privileged=false` ，关闭特权模式。
Init Container	选中该项，表示创建一个Init Container。Init Container包含一些实用的工具，详情请参见Init Containers。

- （可选）端口设置
  设置容器的端口。
  - 名称：设置容器端口名称。
  - 容器端口：设置暴露的容器访问端口或端口名，端口号必须介于1~65535。
  - 协议：支持TCP和UDP。
- （可选）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。

- 类型：设置环境变量的类型，支持**自定义**、**配置项**、**保密字典**、**变量/变量引用**及**资源引用**。配置项、保密字典支持全部文件的引用，以保密字典为例。
  选择**保密字典**，只选择目标保密字典，则默认全部文件引用。



对应的*yaml*，则引用了整个Secret。

```
envFrom:
  - secretRef:
      name: test
```

- 变量名称：设置环境变量名称。
- 变量/变量引用：设置变量引用的值。

- （可选）健康检查
  支持存活检查（Liveness）、就绪检查（Readiness）和启动探测。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；启动探测用于检测何时启动容器。

> **说明** 仅Kubernetes集群1.18及之后版本支持启动探测。

关于健康检查的更多信息，请参见Configure Liveness, Readiness and Startup 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HTTP Headers，HTTP请求中自定义的请求头，HTTP允许重复的Header。支持键值对的配置方式。 ■ 延迟探测时间（秒）：即initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为3秒。 ■ 执行探测频率（秒）：即periodSeconds，指执行探测的时间间隔，默认为10秒，最小为1秒。 ■ 超时时间（秒）：即timeoutSeconds，探测超时时间。默认1秒，最小1秒。 ■ 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（Liveness）必须是1。 ■ 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3，最小值是1。

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

- 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。具体参见配置生命周期。
  - **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。



- （可选）数据卷
  支持增加本地存储、云存储声明（PVC）或增加存储声明模板。
  - **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，请参见Volumes。
  - **增加云存储声明（PersistentVolumeClaim）**：支持云存储。
  - **增加存储声明模板**：支持通过VolumeClaimTemplate为每个Pod创建PVC及PV。关于如何通过VolumeClaimTemplates自动创建PVC及PV，请参见创建StatefulSet服务。

    > **说明**
    > - 由于配置存储声明模板后不可编辑，因此不推荐使用。
    > - 单击**增加存储声明模板**后，在**编辑**对话框的**模板**中输入存储声明模板，例如：

    ```
    metadata:
     name: www
    spec:
     accessModes:
      - ReadWriteOnce
     resources:
      requests:
       storage: 20Gi
    ```

  本例中配置了一个云存储类型的数据卷声明disk-ssd，将其挂载到容器的/tmp路径下。



- （可选）日志配置

配置**日志服务**，您可进行采集配置和自定义Tag设置。

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

配置项	描述
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：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。
自定义Tag	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。

9. 完成容器配置后，单击**下一步**。
10. （可选）进行高级配置。
    - 访问设置

> **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SLB（Server Load Balancer），该服务提供公网访问能力。
> ■ 创建路由（Ingress）：通过路由（Ingress）提供公网访问能力，详情请参见Ingress。

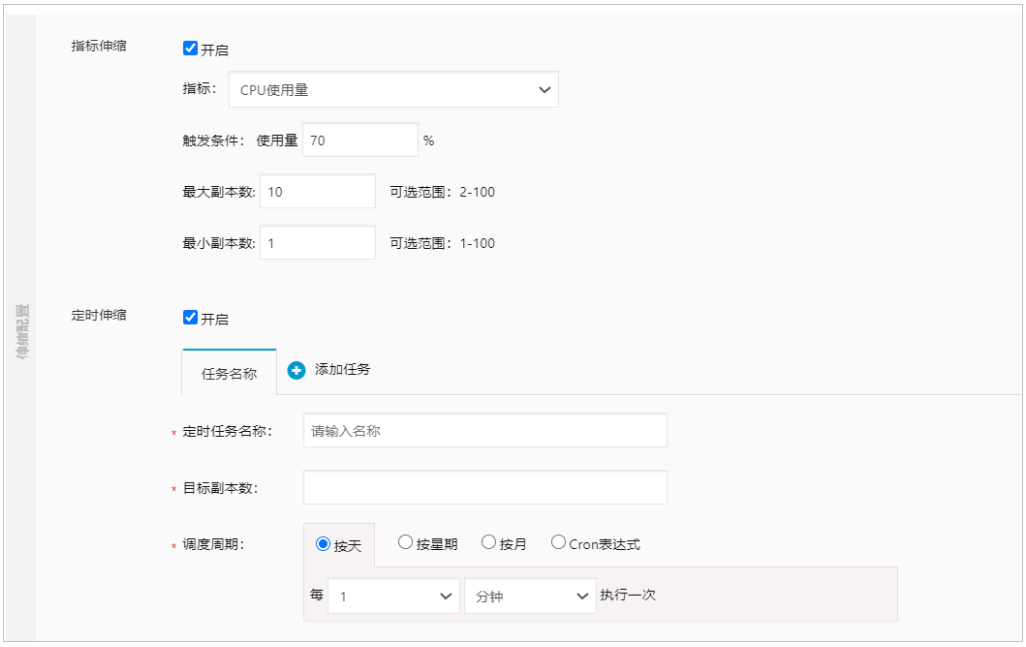
您可以设置暴露后端Pod的方式。本例中选择ClusterIP服务和路由（Ingress），构建一个公网可访问的Nginx应用。

配置项	描述
服务（Service）	在**服务（Service）**右侧，单击**创建**设置**创建服务**的配置项。配置项的详情，请参见管理服务。本例中服务类型选择**ClusterIP**。
路由（Ingress）	在**路由（Ingress）**右侧，单击**创建**。在弹出的对话框中，为后端Pod配置路由规则。详细的路由配置信息，请参见创建Ingress。 **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.XX.XX foo.bar.com #即ingress的IP`

在**访问设置**区域中，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。

- 伸缩配置

在**伸缩配置**区域，配置是否开启**指标伸缩**和**定时伸缩**，从而满足应用在不同负载下的需求。



- 容器服务支持容器组的指标伸缩，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。

配置项	描述
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	该应用可扩容的容器数量上限。
最小副本数量	该应用可缩容的容器数量下限。

- 容器服务支持容器组的定时伸缩，即定时地对容器服务Kubernetes集群进行扩缩容。关于定时伸缩的详细说明，请参见创建定时伸缩CronHPA任务。

- 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，详情请参见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。

配置项	描述
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Deployments。

配置项	描述
节点亲和性	设置**节点亲和性**，通过Worker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Worker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。
应用亲和性	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。

配置项	描述
应用非亲和性	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。
调度容忍	容忍被应用于Pod，允许这个Pod被调度到相对应的污点上。
调度到虚拟节点	设置是否调度到虚拟节点。如果您集群中没有虚拟节点，则无法设置该配置项。

- 标签和注释
  - Pod标签：为该Pod添加一个标签，标识该应用。
  - Pod注解：为该Pod添加一个注解（Annotation）。

11. 单击**创建**。
12. 查看创建的应用。
    i. 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。
    ii. 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。

相关操作

在**有状态**页面单击目标应用名称或目标应用**操作**列下的**详情**。

说明 在有状态列表中单击**标签**字段，输入对应应用标签的key与`value`，单击**确定**后可过滤列表中的应用。

在应用详情页面您可以**编辑**、**伸缩**、**查看YAML**、**重新部署**、**刷新**应用。

- 编辑：在应用详情页面单击**编辑**，您可以修改应用信息。
- 伸缩：在应用详情页面单击**伸缩**，您可以修改所需容器组数量。
  以创建的Nginx应用为例，验证服务伸缩性。
  i. 选择目标Nginx应用，单击右侧**伸缩**。
  ii. 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。
  iii. 单击左侧导航栏中的**存储卷 > 存储声明**，您可发现，随着应用扩容，会随着Pod创建新的云存储卷。缩容后，已创建的PV和PVC不会删除。
- 查看YAML：在应用详情页面单击**查看YAML**，您可以**更新**、**下载**、**另存为** *YAML*文件。
- 重新部署：在应用详情页面单击**重新部署**，您可以重新部署应用。
- 刷新：在应用详情页面单击**刷新**，您可以刷新应用。

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。

1. 执行以下命令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
```

预期输出：

```
lost+found
statefulset
```

2. 执行以下命令删除Pod，验证数据持久性。

```
kubectl delete pod nginx-1
```

预期输出：

```
pod"nginx-1" deleted
```

3. 过一段时间，待Pod自动重启后，验证数据持久性，证明StatefulSet应用的高可用性。

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

预期输出：

```
statefulset
```

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

8.1.3. 创建守护进程集工作负载DaemonSet

DaemonSet保证在每个节点上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用。本文介绍如何在ACK中创建守护进程集工作负载DaemonSet。

通过控制台创建DaemonSet

使用镜像创建DaemonSet

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 守护进程集**。
5. 在**守护进程集**页面右上角，单击**使用镜像创建**。
6. 设置创建DaemonSet的参数。
   i. 在**应用基本信息**配置向导页面设置应用基本信息。关于参数的详细描述，请参见使用镜像创建无状态Deployment应用。
   ii. 在**容器配置**配置向导页面设置容器信息。关于参数的详细描述，请参见使用镜像创建无状态Deployment应用。
   iii. 在**高级配置**配置向导页面完成DaemonSet的高级参数的设置。

   DaemonSet会忽略节点的不可调度状态，您可以通过设置节点亲和性、应用亲和性、调度容忍来指定Pod只运行在指定的节点上。关于参数的详细描述，请参见使用镜像创建无状态Deployment应用。
7. 单击**创建**。
   成功创建守护进程集后，在守护进程集列表中可以查看新创建的守护进程集。

使用模板创建DaemonSet

1. 在**守护进程集**页面右上角，单击**使用YAML创建资源**。
2. 在**创建**页面，在**模板**区域编辑模板信息。
3. 在**模板**区域下方，单击**创建**。
   成功创建守护进程集后，在守护进程集列表中可以查看新创建的守护进程集。

通过kubectl命令行创建DaemonSet

在使用kubectl创建DaemonSet前，您需要下载kubectl并使用kubectl命令连接集群。具体操作，请参见通过kubectl工具连接集群。

DaemonSet会忽略节点的无法调度状态，您可以设置以下参数指定Pod运行在指定的节点上：

参数	描述
nodeSelector	只调度到匹配指定标签的节点上。
nodeAffinity	设置节点亲和性。根据节点标签将Pod调度到指定节点上，但支持更丰富的节点匹配策略。
podAffinity	设置Pod亲和性。根据Pod的标签来选择节点，仅调度到满足条件的Pod所在节点。

本文以创建一个名为fluentd-elasticsearch的DaemonSet为例，说明如何使用Kubectl创建DaemonSet。

1. 保存以下YAML内容至*daemonset.yaml*文件中。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: fluentd-elasticsearch
 namespace: kube-system
 labels:
  k8s-app: fluentd-logging
spec:
 selector:
  matchLabels:
   name: fluentd-elasticsearch
 template:
  metadata:
   labels:
    name: fluentd-elasticsearch
  spec:
   tolerations:
   # this toleration is to have the daemonset runnable on master nodes
   # remove it if your masters can't run pods
   - key: node-role.kubernetes.io/master
    effect: NoSchedule
   containers:
   - name: fluentd-elasticsearch
    image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
    resources:
     limits:
      memory: 200Mi
     requests:
      cpu: 100m
      memory: 200Mi
    volumeMounts:
    - name: varlog
     mountPath: /var/log
    - name: varlibdockercontainers
     mountPath: /var/lib/docker/containers
     readOnly: true
   terminationGracePeriodSeconds: 30
   volumes:
   - name: varlog
    hostPath:
     path: /var/log
   - name: varlibdockercontainers
    hostPath:
     path: /var/lib/docker/containers
```

2. 执行以下命令创建DaemonSet。

```
kubectl create -f daemonset.yaml
```

当输出 `daemonset.apps/fluentd-elasticsearch created` 时，则说明成功创建守护进程集DaemonSet。

相关操作

创建完守护进程集后，您可以进行以下操作：

- 在守护进程集列表中单击**标签**字段，输入对应应用标签的key与 `value` ，单击**确定**后可过滤列表中的应用。
- 在守护进程集名称右侧的**操作**列中，单击**详情**查看守护进程集的基本信息，包括容器组、访问方式、事件、日志等。

- 在守护进程集名称右侧的**操作**列中，选择**更多 > 查看Yaml**、**更多 > 删除**，查看守护进程集的YAML信息和删除守护进程集。

8.1.4. 创建任务工作负载Job

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

前提条件

您已成功创建一个Kubernetes集群。参见创建Kubernetes专有版集群。

背景信息

Job负责批量处理短暂的一次性任务（short lived one-off tasks），即仅执行一次的任务，它保证批处理任务的一个或多个Pod成功结束。

Kubernetes支持以下几种Job：

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

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

> **说明** 本例中创建的任务属于固定结束次数的并行Job。

Job类型	使用示例	行为	completions	Parallelism
一次性Job	数据库迁移	创建一个Pod直至其成功结束	1	1
固定结束次数的Job	处理工作队列的Pod	依次创建一个Pod运行直至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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 任务**。
5. 在**任务**页面，单击**使用镜像创建**。
   - **应用名称**：设置应用的名称。
   - **类型**：设置类型为**任务**。

> **说明** 本例中选择任务类型，即Job。

- **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）。
- **时区同步**：容器与节点是否使用相同的时区。



6. 设置容器配置。

> **说明** 您可为应用的Pod设置多个容器。

i. 设置容器的基本配置。

- **镜像名称**：选择所需镜像。
  - 您可以单击**选择镜像**，选择所需的镜像并单击**确定**。本例中为官方Nginx镜像，可以在**搜索**页签中，选择**Docker镜像**，然后在文本框中输入*nginx*进行搜索。
    - 容器镜像服务：在**容器镜像服务**页签中，可以选择托管在容器镜像服务ACR中的镜像。需要选择镜像所属地域，以及镜像服务实例。关于ACR的更多信息，请参见什么是容器镜像服务ACR。

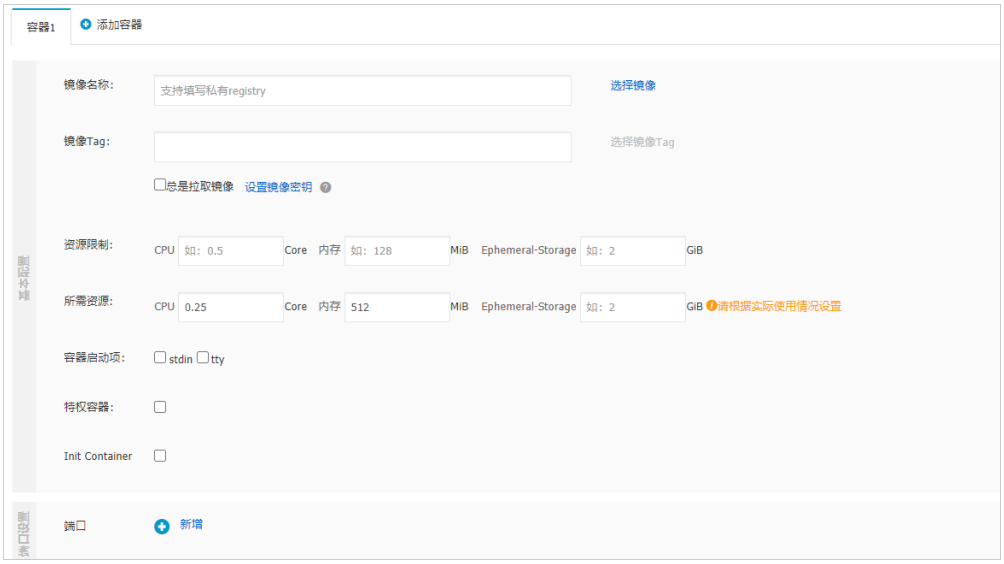
      > **说明** 您可以在**容器镜像服务**页签，根据名称搜索托管在容器镜像服务ACR中的镜像。

    - Docker官方镜像：在**Docker官方镜像**页签中，可以选择Docker官方提供的容器镜像。
    - 用户收藏：在**用户收藏**页签中，可以选择您收藏的Docker官方镜像。
    - 搜索：在**搜索**页签中，可以分别在ACR的特定地域和Docker官方镜像中搜索容器镜像。
  - 您还可以填写私有镜像。填写的格式为 `domainname/namespace/imagename:tag` 。

- **镜像Tag**：
  - 您可以单击**选择镜像Tag**选择镜像的版本。若不指定，默认为最新版。
  - ACK支持以下三种镜像拉取策略（imagePullPolicy）：
    - **优先使用本地镜像（IfNotPresent）**：如果本地有该镜像（之前拉取过该镜像至宿主机中），则使用本地镜像，本地不存在时拉取镜像。
    - **总是拉取镜像（Always）**：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。
    - **仅使用本地镜像（Never）**：仅使用本地镜像。

    > **说明** 当您选中**镜像拉取策略**时，默认不设置镜像拉取策略。

  - 单击**设置镜像密钥**，您可以实现免密拉取镜像。具体操作，请参见使用免密组件拉取容器镜像。
- **总是拉取镜像**：为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像Tag与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改Tag，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。
- **设置镜像密钥**：若您在使用私有镜像时，您可使用镜像密钥，保障镜像安全。具体配置，请参见使用镜像密钥创建应用。
- **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抢占资源，导致应用不可用。
- **容器启动项**：选择**stdin**，将控制台输入发送到容器。选择**tty**将标准输入控制台作为容器的控制台输入。
- **特权容器**：选择特权容器，则privileged=true，开启特权模式。不选择特权容器，则privileged=false，关闭特权模式。
- **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，更多信息，请参见Init Container。
- **端口**：设置主机端口和容器端口，支持TCP/UDP协议。



ii. （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，更多信息，请参见Pod variable。

iii. （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。具体参见配置生命周期。

- **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。



iv. （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。

v. （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。

7. 完成容器配置后，单击**下一步**。
8. 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的数量。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。

参数	说明
重试次数	Pod执行失败后，尝试重建Pod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）



9. 最后单击**创建**。
10. 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。

11. 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。

相关操作

在左侧导航栏单击**集群**，单击目标集群名称或目标集群**操作**列下的**详情**，选择**工作负载**，单击**任务**页签，单击目标应用名称或目标应用**操作**列下的**详情**，在应用详情页面您可以**伸缩**、**查看Yaml**、**刷新**应用。

- 伸缩：在应用详情页面单击**伸缩**，您可以修改所需容器组数量。
- 查看Yaml：在应用详情页面单击**查看Yaml**，您可以**更新**、**下载**、**另存为** *Yaml*文件。
- 刷新：在应用详情页面单击**刷新**，您可以刷新应用。

8.1.5. 创建定时任务工作负载CronJob

定时任务CronJob用于创建周期性以及重复性的任务，例如执行备份操作或者发送邮件。Job负责批量处理短暂的一次性任务，即仅执行一次的任务，而CronJob则是在Job上增加了时间调度。本文介绍如何创建定时任务CronJob。

通过控制台创建CronJob

使用镜像创建CronJob

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 定时任务**。
5. 在**定时任务**页面右上角，单击**使用镜像创建**。
6. 设置创建CronJob的参数。
   i. 在**应用基本信息**配置向导页面设置应用基本信息。关于参数的详细描述，请参见步骤五：设置应用基本信息。
   ii. 在**容器配置**配置向导页面设置容器信息。关于参数的详细描述，请参见容器配置。

iii. 在**高级配置**配置向导页面完成CronJob的高级参数的设置。

分类	参数	描述
定时任务	定时规则	您可以按天、按星期、按月，或者使用Cron表达式设置定时规则。Cron表达式是一个字符串，字符串以5或6个空格隔开，分为6或7个域，每一个域代表一个含义，Cron有以下两种语法格式： 秒（Seconds） 分（Minutes） 小时（Hours） 日期（DayofMonth） 月份（Month） 星期（DayofWeek） 年（Year） 秒（Seconds） 分（Minutes） 小时（Hours） 日期（DayofMonth） 月份（Month） 星期（DayofWeek） 关于更多Cron表达式说明，请参见Cron Expressions。
	并发策略	您可以设置以下三种并发策略： ■ Allow：允许并发运行任务，会抢占集群资源。 ■ Forbid：禁止并发运行，如果前一个还没有完成，则直接跳过下一个任务。 ■ Replace：已到新任务创建时间点，但前一个任务还未完成，新的任务会取代前一个任务。
	任务记录	可以设置保留执行成功或执行失败的任务个数，设置为0表示不保留。
任务设置	成功运行的Pod数	关于任务设置的参数描述，请参见任务配置。
	并行运行的Pod数	
	超时时间	
	重试次数	
	重启策略	
标签和注解	Pod标签	为Pod设置标签的名称和值。 **说明** 标签名称长度为1~253字符，只能包含大小写字母、数字、短划线（-）、下划线（_）和小数点（.）。
	Pod注解	为Pod设置注解的名称和值。 **说明** 注解名称长度为1~253字符，只能包含大小写字母、数字、短划线（-）、下划线（_）和小数点（.）。

7. 单击**创建**。
   成功创建定时任务后，在定时任务列表中可以查看新创建的定时任务。

使用YAML创建CrobJob

1. 在**定时任务**页面右上角，单击**使用YAML创建资源**。
2. 在**创建**页面，在**模板**区域编辑模板信息。

3. 单击**创建**。

通过kubectl命令行创建CronJob

在使用kubectl创建CronJob前，您需要下载kubectl并使用kubectl命令连接集群。具体操作，请参见通过kubectl工具连接集群。

创建CronJob涉及的主要参数如下：

参数	描述
.spec.schedule	指定任务运行周期，格式同Cron schedule。
.spec.jobTemplate	指定需要运行的任务，格式同Job patterns。
.spec.startingDeadlineSeconds	指定任务开始的截止期限。
.spec.concurrencyPolicy	指定任务的并发策略，支持Allow、Forbid和Replace三个选项： • Allow：允许并发运行任务，会抢占集群资源。 • Forbid：禁止并发运行，如果前一个还没有完成，则直接跳过下一个任务。 • Replace：已到新任务创建时间点，但前一个任务还未完成，新的任务会取代前一个任务。

本文以创建一个名为hello的CronJob为例，说明如何使用Kubectl创建CronJob。

1. 保存以下YAML内容至*cronjob.yaml*文件中。

```
apiVersion: batch/v2alpha1
kind: CronJob
metadata:
 name: hello
spec:
 schedule: "*/1 * * * *"
 jobTemplate:
  spec:
   template:
    spec:
     containers:
     - name: hello
      image: busybox
      args:
      - /bin/sh
      - -c
      - date; echo Hello from the Kubernetes cluster
     restartPolicy: OnFailure
```

2. 执行以下命令创建CronJob。

```
kubectl create -f cronjob.yaml
```

当输出 `cronjob.batch/hello created` 时，则说明成功创建定时任务CronJob。

相关操作

创建完定时任务后，您可以进行以下操作：

- 在定时任务名称右侧的**操作**列中，单击**详情**查看定时任务的基本信息，包括任务列表、事件、日志。
- 在定时任务名称右侧的**操作**列中，选择**更多 > 查看Yaml**、**更多 > 停止**、**更多 > 删除**，查看定时任务的YAML信息、停止任务、删除任务。

8.1.6. 管理容器组（Pod）

容器组（Pod）是Kubernetes中最小的可部署单元。容器组代表了Kubernetes中一个独立的应用程序运行实例，该实例可能由单个容器或者几个紧耦合在一起的容器组成。本文介绍如何通过ACK控制台查看、变更容器组（Pod）配置及如何手动伸缩容器应用。

查看容器组（Pod）

查看容器组详细信息

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。
5. 单击目标容器组右侧的**详情**，查看容器组的详细信息。

   您还可以通过以下两种方式查看容器组信息：

   - 方式一：在集群管理页左侧导航栏中，选择**工作负载 > 无状态**，单击目标应用名称，然后在**容器组**页签，单击容器组名称查看容器组信息。
   - 方式二：在集群管理页左侧导航栏中，选择**网络 > 服务**，单击目标服务名称，单击目标应用名称，然后在**容器组**页签，单击容器组名称查看容器组信息。

   > **说明** 在容器组列表页面，您可对容器组进行编辑和删除操作。对于通过部署（例如Deployment）创建的容器组，建议您通过Deployment进行管理。

   以下为容器组现状详情的说明：

类型	说明
Initialized	所有的Init容器都已成功启动。
Ready	Pod可以为请求提供服务，并且应该被添加到对应服务的负载均衡池中。
ContainersReady	Pod中所有容器都已就绪。
PodScheduled	Pod已经被调度到某节点。

   更多信息，请参见Pod的生命周期。

查看容器组日志

您可以在工作负载的应用列表中或者容器组列表中查看容器的日志。

在容器组列表中，选择目标容器组，然后单击其右侧的**日志**，查看容器的日志信息

过滤容器组

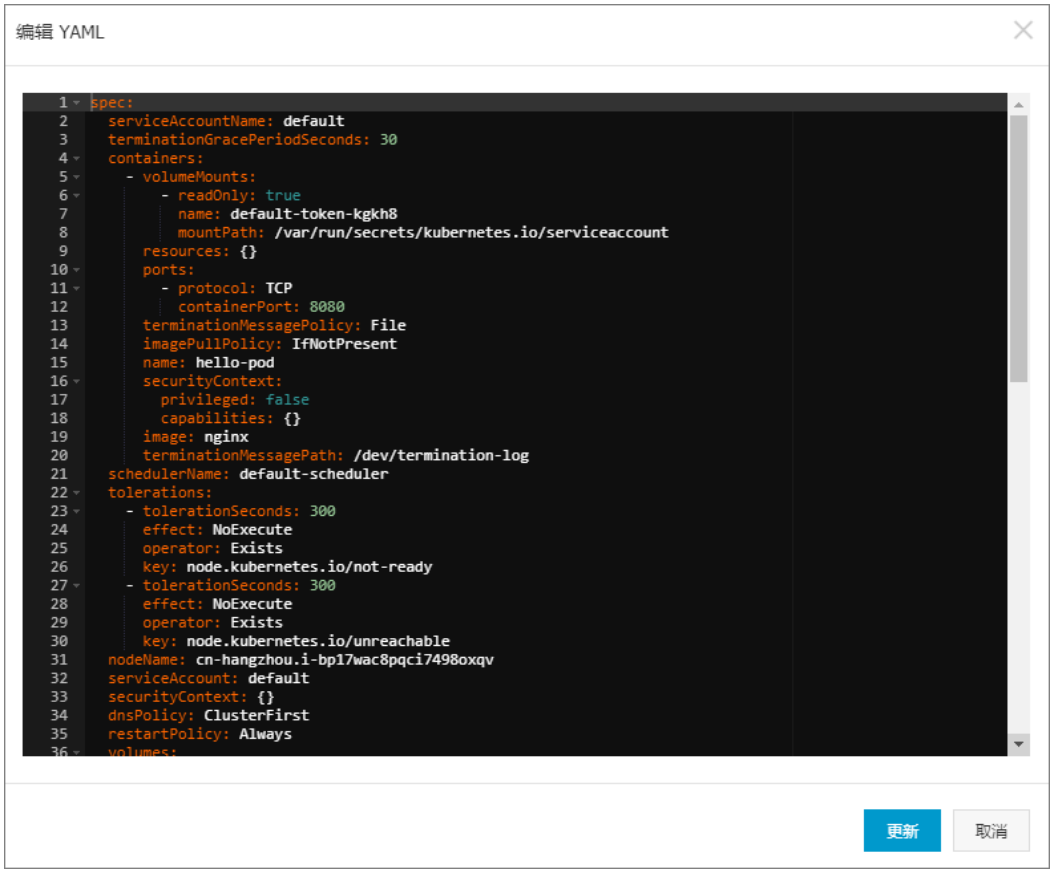
在容器组列表页面，您可以根据名称、节点、Host IP、Pod IP和标签，过滤出相应的容器组。



变更容器组配置

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。
5. 在**容器组**页面，选择所需的容器组，单击右侧的**编辑**。
6. 更新容器组的配置并单击**更新**。



手动伸缩容器组

应用创建后，您可以根据业务需求扩容或缩容服务。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 选择命名空间，选择所需的Deployment，单击目标应用右侧**操作**列的**伸缩**。
6. 在弹出的对话框中，设置所需容器组的数量，将其调整为4，然后单击**确定**。

> **说明** Kubernetes的Deployment资源默认的更新方式就是rollingUpdate，所以在更新过程中，会保证最小可服务的容器个数，该个数也可以在模板里面调整。

8.1.7. 管理自定义资源

Kubernetes集群支持通过自定义资源定义（Custom Resource Definition）扩展Kubernetes API，并允许您添加自定义资源（Custom Resources）。您可以查看集群中的所有API组和其包含的资源类型，查看各个资源类型的对象列表并进行管理操作。本文介绍如何管理自定义资源。

使用限制
部分资源类型仅支持创建等操作，无法列出对应的资源对象。

操作步骤
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页面左侧导航栏中，选择**工作负载 > 自定义资源**。
5. 管理对象浏览器。
   - 浏览API组和资源类型列表。
     对象浏览器左侧列出了集群支持的全部API组。您可以点击API组的名称展开查看组内包含的资源类型，也可以通过上方的搜索框，使用关键词搜索全部资源类型。
   - 浏览资源对象。
     在左侧API列表中选择一个资源类型，即可浏览对应的资源对象。您可以对资源对象进行YAML编辑和删除操作。

8.2. 镜像

8.2.1. kritis-validation-hook组件介绍

kritis-validation-hook组件是部署可信容器环节中进行容器镜像签名验证的关键组件。通过在部署前对容器镜像进行签名验证可以确保只部署经过可信授权方进行过签名的容器镜像，降低在您的环境中运行意外或恶意代码的风险。本文通过具体示例对kritis-validation-hook组件的工作效果进行介绍。

前提条件
已创建托管版或专有版集群。具体操作，请参见创建Kubernetes托管版集群或创建Kubernetes专有版集群。

背景信息
kritis-validation-hook组件在开源的kritis软件的基础上增加了对阿里云容器镜像服务ACR的深度集成，支持验证通过阿里云密钥管理服务KMS签名的容器镜像。kritis-validation-hook组件通过与云安全中心、KMS和ACR的深度合作，实现了全自动化的对容器镜像进行加签和验签，协助您构建更安全的集群运行环境。实现自动验证容器镜像签名，请参见使用kritis-validation-hook组件实现自动验证容器镜像签名。

配置资源访问权限
为了正常运行kritis-validation-hook组件，需要确保集群使用的RAM角色拥有以下资源访问权限：

> 注意
> - 若您使用的是托管版集群，需要确保集群使用的Worker RAM角色拥有以下资源访问权限。
> - 若您使用的是专有版集群，需要确保集群使用的Master RAM角色和Worker RAM角色拥有以下资源访问权限。

```
cr:ListInstance
cr:ListMetadataOccurrences
```

如果您的集群使用的RAM角色没有上述访问权限，您可以通过以下方法添加组件所需的资源访问权限。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

4. 在**集群信息**页面的**集群资源**页签找到集群所使用的RAM角色。
   - 对于托管版集群，单击**Worker RAM 角色**名称。
   - 对于专有版集群，先后单击**Master RAM 角色**和**Worker RAM 角色**名称。
5. 在**RAM角色管理**页面单击**权限管理**页签，然后单击**权限策略名称**列下的**k8s******RolePolicy-******。
6. 在**权限策略管理**页面单击**策略内容**页签，再单击**修改策略内容**。
7. 在**修改策略内容**面板新增以下Action策略，然后单击**确定**。

```
{
  "Action": [
    "cr:ListInstance",
    "cr:ListMetadataOccurrences"
  ],
  "Resource": [
    "*"
  ],
  "Effect": "Allow"
}
```

启用镜像签名验证示例

以下通过对当前**default**命名空间启用镜像签名验证为例，展示kritis-validation-hook组件的工作效果。

> **说明** 因为镜像签名不属于kritis-validation-hook组件的工作范畴，所以示例略过签名的步骤。本示例中涉及的签名后的信息如下：
>
> - 使用KMS签名的容器镜像：kritis-demo-registry.cn-hangzhou.cr.aliyuncs.com/kritis-demo/alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45。
> - 使用的KMS key对应的公钥信息存储在*publickey.txt*中。
> - 使用的KMS key-id：`4a2ef103-5aa3-4220-****`。

1. 配置AttestationAuthority，声明相应的可信授权方。
   i. 使用以下内容创建*AttestationAuthority.yaml*文件。

```
apiVersion: kritis.grafeas.io/v1beta1
kind: AttestationAuthority
metadata:
 name: demo-aa
spec:
 noteReference: namespaces/demo-aa
 publicKeyData: $(cat publickey.txt | base64 | tr -d '\n')
 publicKeyId: 4a2ef103-5aa3-4220-****
```

   ii. 执行以下命令配置KMS公钥信息。

```
kubectl apply -f AttestationAuthority.yaml
```

2. 配置GenericAttestationPolicy，声明验证签名策略及使用可信授权方的信息验证签名。

i. 使用以下内容创建*GenericAttestationPolicy.yaml*文件。

```
apiVersion: kritis.grafeas.io/v1beta1
kind: GenericAttestationPolicy
metadata:
 name: demo-gap
spec:
 attestationAuthorityNames:
 - demo-aa
```

ii. 执行以下命令配置验证签名策略。

```
kubectl apply -f GenericAttestationPolicy.yaml
```

3. 测试镜像签名验证功能，部署未经信任授权方签名的镜像。
   - 执行以下命令测试镜像签名验证功能。

     ```
     kubectl create deployment test-denied --image=alpine:3.11
     ```

     预期输出：

     ```
     Error from server: admission webhook "kritis-validation-hook-deployments.grafeas.io" denied the request: image alpine:3.11 is not attested
     ```

   - 执行以下命令测试镜像签名验证功能。

     ```
     kubectl create deployment test-denied --image=kritis-demo-registry.cn-hangzhou.cr.aliyuncs.com/kritis-demo/alpine:3.11
     ```

     预期输出：

     ```
     Error from server: admission webhook "kritis-validation-hook-deployments.grafeas.io" denied the request: image kritis-demo-registry.cn-hangzhou.cr.aliyuncs.com/kritis-demo/alpine:3.11 is not attested
     ```

   从预期输出可得，镜像签名验证功能可拒绝部署未经信任授权方签名的镜像。
4. 测试镜像签名验证功能，部署经信任授权方签名的镜像。

   执行以下命令测试镜像签名验证功能。

   ```
   kubectl create deployment test-allow --image=kritis-demo-registry.cn-hangzhou.cr.aliyuncs.com/kritis-demo/alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45
   ```

   预期输出：

   ```
   deployment.apps/test-allow created
   ```

   从预期输出可得，镜像签名验证功能可部署经信任授权方签名的镜像。

配置镜像验签白名单

在中间件或服务网格场景下，kritis-validation-hook组件支持通过配置镜像验签白名单的方式，解决第三方组件自动注入的Sidecar容器的镜像无法通过镜像验签，导致无法部署Pod的问题。组件不会对白名单内的镜像进行签名验证，只会验证未在白名单内的镜像。

可以通过定义 `admissionallowlists.kritis.grafeas.io` 资源的方法来配置镜像验签白名单。CRD资源定义如下：

```
apiVersion: kritis.grafeas.io/v1beta1 # 默认值，无需修改。
kind: AdmissionAllowlist          # 默认值，无需修改。
metadata:
 name: kritis-allowlist          # 资源名，在集群内唯一。
spec:
 patterns:                  # 白名单配置，可以定义多条白名单。
 - namePattern: 'registry*.*.aliyuncs.com/acs/*' # 想忽略的镜像名称，具体格式详见下方说明。
 - namePattern: 'registry-vpc.cn-beijing.aliyuncs.com/arms-docker-repo/*'
  namespace: 'default'  # [可选]，白名单配置适用于哪个命名空间，未配置时适用于所有命名空间。
```

若您需要把ACK的系统镜像加入白名单，可以通过以下操作配置。

1. 使用以下内容创建 kritis-admission-allowlist-acs.yaml 文件配置白名单。

   ```
   apiVersion: kritis.grafeas.io/v1beta1
   kind: AdmissionAllowlist
   metadata:
    name: allow-acs-images
   spec:
    patterns:
    - namePattern: 'registry*.*.aliyuncs.com/acs/*'
   ```

   namePattern配置项值支持完整匹配，且支持通过星号（*）字符实现简单的通配符匹配，具体说明如下：

   - 当配置项值不包含星号（*）字符时，将按配置的值进行完整匹配。例如， nginx:v0.1.0 只会匹配 nginx:v0.1.0 。
   - 当通过星号（*）字符实现通配符匹配时，包含以下限制：
     - 星号（*）位于末尾时，匹配除正斜线（/）外的任意字符。例如， a.com/nginx* 匹配 a.com/nginx:v0.1.0 ，但是不匹配 a.com/nginx/test:v0.1.0 。
     - 星号（*）不位于末尾时，匹配字母、数字、短横线(-)以及下划线（_）。例如， registry-vpc.cn-*.aliyuncs.com/acs/pause:3.2 既匹配 registry-vpc.cn-hangzhou.aliyuncs.com/acs/pause:3.2 ，也匹配 registry-vpc.cn-beijing.aliyuncs.com/acs/pause:3.2 。

   常见的白名单如下，您可以根据实际需求添加。

   ```
   # 容器服务ACK使用的镜像
   - namePattern: 'registry*.*.aliyuncs.com/acs/*'
   # 容器服务ACK使用的镜像（限定中国地域）
   - namePattern: 'registry*.cn-*.aliyuncs.com/acs/*'
   # ARMS使用的镜像
   - namePattern: 'registry*.*.aliyuncs.com/arms-docker-repo/*'
   # ARMS使用的镜像（限定中国地域）
   - namePattern: 'registry*.cn-*.aliyuncs.com/arms-docker-repo/*'
   ```

2. 执行以下命令配置白名单。

   ```
   kubectl apply -f kritis-admission-allowlist-acs.yaml
   ```

   预期输出：

   ```
   admissionallowlist.kritis.grafeas.io/allow-acs-images created
   ```

3. 执行以下命令验证白名单配置是否成功。

   ```
   kubectl get admissionallowlists.kritis.grafeas.io
   ```

   预期输出：

```
NAME          AGE
allow-acs-images  2m22s
```

相关文档

- kritis-validation-hook
- 使用kritis-validation-hook组件实现自动验证容器镜像签名

8.2.2. 管理镜像

本文为您介绍如何创建镜像及查看镜像。

创建镜像

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 容器镜像服务**。
3. 在**实例列表**页面单击目标企业版实例。

   若需要创建企业版实例，请参见步骤三：创建企业版实例。
4. 创建命名空间。具体操作，请参见步骤四：创建命名空间。
5. 创建镜像仓库。具体操作，请参见步骤五：创建镜像仓库。
6. 上传镜像至镜像仓库，或者从第三方代码仓库自动构建镜像。具体操作，请参见使用企业版实例推送和拉取镜像和使用容器镜像服务企业版构建镜像。

查看镜像

1. 登录容器镜像服务控制台。
2. 在顶部菜单栏，选择所需地域。
3. 在左侧导航栏，选择**实例列表**。
4. 在**实例列表**页面单击目标企业版实例。
5. 在企业版实例管理页面左侧导航栏选择**仓库管理 > 镜像仓库**，然后单击目标镜像仓库名称。
6. 在镜像仓库详情页面左侧导航栏单击**镜像版本**，查看镜像仓库中镜像的版本信息。

8.2.3. 使用免密组件拉取容器镜像

aliyun-acr-credential-helper是一个可以在ACK集群中免密拉取ACR个人版或企业版私有镜像的组件。该组件会默认安装在所有ACK集群中。本文列举四个场景介绍如何使用免密组件拉取私有镜像。

前提条件

- 创建Kubernetes托管版集群
- 通过kubectl工具连接集群

背景信息

免密组件通过读取ACK集群内的kube-system命名空间中的acr-configuration的配置，进行私有镜像拉取。当前支持针对私有镜像仓库使用以下三种权限之一的策略进行配置：

- 使用默认的ACK Worker RAM角色权限进行拉取（默认策略）。
- 使用自定义RAM角色的AccessKey ID和AccessKey Secret的权限进行拉取。
- 通过配置RAM AssumeRole权限， 使用其他用户的权限进行拉取。

实现机制

根据升级并配置免密组件的要求配置免密插件后，免密插件会在集群中创建Secrets并关联到配置中指定的ServiceAccount 上，任何使用这个ServiceAccount的Pod都会默认使用免密插件生成的Secrets进行镜像拉取，从

而达到免密拉取镜像的效果。

使用限制

使用免密组件涉及的镜像及集群限制如下：

- 镜像
  - 支持拉取容器镜像服务企业版实例和个人版实例中的私有镜像。
  - 支持拉取集群当前用户容器镜像服务中的私有镜像，通过跨账号授权或AccessKey ID和AccessKey Secret配置可以拉取其他用户的私有镜像。
  - 支持跨地域拉取容器镜像服务中的私有镜像。
- 集群
  - 支持集群多命名空间免密拉取。
  - 支持的集群类型：
    - 专有版Kubernetes集群。
    - 托管版Kubernetes集群。
  - 支持的集群版本：
    - 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。低于1.11.2版本请您手动升级，请参见升级ACK集群K8s版本。
    - 托管版Kubernetes集群：所有版本。

注意事项

- 在Kubernetes资源（例如无状态应用Deployment）模板中配置拉取凭证（imagePullSecret）会导致免密组件失效，如果需使用免密组件，请避免手工配置拉取凭证（imagePullSecret）。
- 如果部署的Kubernetes资源（例如无状态应用Deployment）使用了自定义的ServiceAccount，需先调整免密组件配置文件中Service-Account字段，使其作用于自定义的ServiceAccount，再进行部署资源操作。
- 确认Kubernetes集群所属地域与要拉取的镜像所在的地域是否一致，默认配置只可以拉取本地域的镜像。如果需要跨地域拉取镜像，请参见下面的场景三。
- 使用自定义RAM角色的AccessKey ID和AccessKey Secret进行镜像拉取的情况下，需要把访问密钥写入configMap，这样存在一定的密钥泄露风险。请确定AccessKey ID和AccessKey Secret所属的RAM角色只拥有拉取容器镜像的相关权限。
- 在集群中创建新的Service Account一段时间后，免密插件根据ACK集群默认权限生成的ACR私有镜像拉取Token才会更新到应用使用到的Service Account中， 使用Service Account的应用才会使用Token去拉取镜像。如果创建完Service Account之后立即创建应用则会出现因鉴权失败无法拉取的情况。
- 免密插件默认覆盖ACK中所有命名空间中默认的ServiceAccount中的imagePullSecret字段。被覆盖的ServiceAccount会随着对应kube-system命名空间中**acr-configuration**配置项中的service-account字段变动而变动。
- 在修改kube-system命名空间中的**acr-configuration**配置项时，请确认缩进是否与给出的场景的例子相同。建议直接复制对应场景的YAML内容到编辑器中，修改对应的值然后直接应用到集群，以保证YAML格式的正确性。

升级并配置免密组件

在使用免密组件拉取镜像前，您可能需要升级组件并对组件进行配置，操作步骤如下。

1. 升级aliyun-acr-credential-helper组件。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面，单击目标集群操作列下的**更多 > 系统组件管理**。
   iv. 在**安全**区域，找到**aliyun-acr-credential-helper**，单击**升级**。
2. 设置acr-configuration配置项。

方式一：通过控制台方式设置acr-configuration配置项。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面，单击目标集群下的**详情**。

iv. 在**集群信息**页面左侧导航栏，选择**配置管理 > 配置项**。

v. 在**配置项**页面的上方命名空间下拉框中，选择**kube-system**，然后找到配置项**acr-configuration**，然后通过以下两种方式配置acr-configuration。

- 方式一：单击其右侧的**编辑**，设置**配置项**键和值。
  如果您没有**acr-configuration**配置项，请参见创建配置项。关于如何更新配置项，请参见修改配置项。
- 方式二：单击其右侧的**YAML编辑**，设置**配置项**键和值。

acr-configuration配置项的键和值说明如下。

配置项键	配置项键说明	配置项值
service-account	使免密组件作用于指定的服务账号。	默认为**default**。 **说明** 如果要配置多个请以逗号分隔，如果设置为“*”，表示支持所有命名空间下的所有ServiceAccount。

acr-registry-info	容器镜像的实例信息数组，YAML多行字符串格式，每个实例以三元组方式配置。
	说明 实例信息三元组：
	■ instanceId：实例ID，企业版实例必须配置此项。
	■ regionId：可选，默认为本地地域。
	■ domains：可选，默认为相应实例的所有域名。若要指定个别域名，多个以逗号分隔。
		默认值为空，表示免密拉取本地地域的默认容器镜像实例仓库。
		针对企业版容器镜像实例，配置示例如下：
		`- instanceId: "cri-xxx"`
		`regionId: "cn-hangzhou"`
		`domains: "xxx.com,yyy.com"`
		针对个人版容器镜像实例，配置示例如下：
		`- instanceId: ""`
		`regionId: "cn-hangzhou"`
		`domains: "xxx.com,yyy.com"`

watch-namespace	期望能免密拉取镜像的Namespace。	默认值为**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过期阈值。	默认值为**15m**（建议使用15 min）。

方式二：通过kubectl命令行的方式设置acr-configuration配置项。

i. 执行以下命令打开acr-configuration配置项的编辑页。

```
kubectl edit cm acr-configuration -n kube-system
```

ii. 根据实际情况设置acr-configuration配置项的值。

企业版和个人版容器镜像实例下设置acr-configuration配置项的示例如下：

- 企业版

```
apiVersion: v1
data:
 acr-api-version: "2018-12-01"
 acr-registry-info: |-
  - instanceId: "cri-xxx"
    regionId: "cn-hangzhou"
 expiring-threshold: 15m
 service-account: default
 watch-namespace: all
kind: ConfigMap
metadata:
 name: acr-configuration
 namespace: kube-system
 selfLink: /api/v1/namespaces/kube-system/configmaps/acr-configuration
```

- 个人版

```
apiVersion: v1
data:
 acr-api-version: "2018-12-01"
 acr-registry-info: |-
  - instanceId: ""
    regionId: "cn-hangzhou"
 expiring-threshold: 15m
 service-account: default
 watch-namespace: all
kind: ConfigMap
metadata:
 name: acr-configuration
 namespace: kube-system
 selfLink: /api/v1/namespaces/kube-system/configmaps/acr-configuration
```

场景一：拉取个人版实例和企业实例的私有镜像

ACK支持同时拉取企业版和个人版的私有镜像，只拉取企业版的私有镜像，只拉取个人版的私有镜像。根据您的使用场景，按照以下方式修改配置项**acr-configuration**中的configMap。配置步骤请参见上述配置组件。以下为配置内容：

- 配置拉取企业版的私有镜像。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: "cri-xxx"
     regionId: "cn-hangzhou"
     domains: "xxx.com,yyy.com"
```

- 配置拉取个人版的私有镜像。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
     regionId: "cn-hangzhou"
     domains: "xxx.com,yyy.com"
```

- 配置同时拉取个人版和企业版的私有镜像。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
   - instanceId: "cri-xxxx"
```

场景二：配置当前账号拉取权限

您需要检查当前账号ACR拉取的权限。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面，单击**集群资源**页签，单击**Worker RAM角色**右侧链接。
5. 在RAM角色基本信息的**权限管理**页签，单击目标**权限策略名称**。
6. 单击**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"
}
```

> **说明** 您还需要检查Pod中镜像字段的值是否存在于ACR仓库里面。
>
> - 如果存在，但不在当前账号下，您可能需要使用跨账号镜像拉取（具体操作，请参见场景四：配置跨账号拉取权限）。
> - 如果存在，且在当前账号下，您可以提交工单。
> - 如果不存在，您需要确认该镜像是否存在于阿里云容器镜像服务上。如果不存在，则需要同步外部镜像到阿里云容器镜像服务商，或者变更改镜像为公网可拉取状态。

场景三：配置跨地域拉取镜像权限

如果需要拉取的镜像与当前ACK集群不属于同一地域的时候，需要修改配置项**acr-configuration**中的configMap。

例如，默认仓库同时拉取北京地域与杭州地域的镜像，配置如下。配置步骤请参见上述配置组件。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
     regionId: "cn-beijing"
   - instanceId: ""
     regionId: "cn-hangzhou"
```

场景四：配置跨账号拉取权限

跨账号拉取权限分为：

- **使用角色扮演进行跨账号拉取**：A用户扮演B用户的角色拉取B用户的私有镜像。
- **使用自定义RAM角色的访问密钥进行跨账号私有镜像拉取**：需要使用指定的用户的权限拉取特定的私有镜像。

使用角色扮演进行跨账号拉取

> 说明
> 配置原则如下：
> 1. B用户（RAM角色）可以拉取指定私有仓库下的私有镜像（B用户RAM角色有cr.*相关的权限）。
> 2. B用户（RAM角色）允许让A用户下的ACK Worker RAM角色扮演（信任策略）。
> 3. A用户下的ACK集群的Worker RAM角色有扮演B用户（RAM角色）的权限（AliyunAssumeRoleAccess）。
> 4. 设置A用户下的Worker RAM角色扮演B用户（configMap中的assumeRoleARN）。

1. 创建B用户的受信实体为**阿里云账号**类型的RAM角色，并确保该RAM角色拥有拉取B用户私有镜像的权限。
   i. 创建RAM角色。
      具体操作请参见创建可信实体为阿里云账号的RAM角色。

ii. 自定义RAM角色权限策略内容，确保该RAM角色拥有拉取B用户私有镜像的权限。

具体操作请参见修改自定义策略内容。

> 注意 请确保该RAM角色有cr.*的相关权限。

```
{
"Action": [
 "cr:GetAuthorizationToken",
 "cr:ListInstanceEndpoint",
 "cr:PullRepository"
],
"Resource": [
"*"
],
"Effect": "Allow"
}
```

添加字段位置如下图所示。



2. 在创建的B用户的RAM角色上配置信任策略，允许A用户（需要拉取B用户下的私有镜像）的ACK集群的Worker RAM角色来扮演B用户。

i. 查看A用户的ACK集群的Worker RAM角色的ARN信息。

查看ARN信息具体步骤，请参见如何查看RAM角色的ARN?

ii. 配置信任策略。

a. 在RAM控制台左侧导航栏，单击**RAM角色管理**，找到目标RAM角色并单击RAM角色名称。

b. 在RAM角色基本信息页，单击**信任策略管理**页签，然后使用A用户的Worker RAM角色的ARN修改信任策略。



3. 确认A用户的Worker RAM角色拥有AssumeRole权限。

权限应用范围	权限策略名称
全局	AliyunSTSAssumeRoleAccess

具体操作步骤，请参见查看权限策略基本信息。

4. 调整组件配置，新增assumeRoleARN配置。

   配置内容为B用户Worker RAM角色的ARN。查看ARN信息具体步骤，请参见如何查看RAM角色的ARN？YAML配置示例如下。配置步骤，请参见上述配置组件。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
     regionId: "cn-beijing"
     domains: "registry.cn-beijing.aliyuncs.com"
     assumeRoleARN: "acs:ram::.*:role/kubernetesworkerrole-test"
```

使用RAM用户的访问密钥拉取跨账号私有镜像

1. 创建一个RAM用户并为该用户添加cr镜像拉取权限。

   有关配置cr镜像拉取权限的具体操作步骤，请参见使用角色扮演进行跨账号拉取步骤1。

2. 配置**kube-system**命名空间中的配置项acr-configuration，并填入创建的RAM用户的访问密钥AccessKey ID和AccessKey Secret。

   此种方式可以使免密组件直接使用创建的RAM用户拉取私有镜像。查看访问密钥的具体步骤，请参见查看访问密钥信息。
   配置示例如下。配置步骤请参见上述配置组件。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
     customAccessKey: "xxxxx" // 此处填写刚创建的RAM用户的AccessKey ID。
     customAccessKeySecret: "xxxxxx" // 此处填写刚创建的RAM用户的AccessKey Secret。
```

8.2.4. 使用kritis-validation-hook组件实现自动验证容器镜像签名

本文介绍如何使用容器镜像服务ACR（Container Registry）、密钥管理服务KMS（Key Management Service）、云安全中心（Security Center）和kritis-validation-hook组件实现自动验证容器镜像签名，确保只部署可信授权方签名的容器镜像，降低在您环境中发生意外或恶意代码的风险。

步骤一：安装kritis-validation-hook组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在左侧导航栏选择**运维管理 > 组件管理**。

5. 在**组件管理**页面的**安全**区域，找到**kritis-validation-hook**并单击**安装**。
6. （可选）授予访问kritis-validation-hook组件所需的权限。

   如果您使用的是ACK托管版集群，还需要确保您的集群拥有运行kritis-validation-hook组件所需的访问权限。具体操作，请参见配置资源访问权限。

   > **说明** ACK专有版集群默认开通运行kritis-validation-hook组件所需的访问权限。

步骤二：创建用于容器镜像签名的KMS密钥

1. 登录密钥管理服务控制台。
2. 在**密钥列表**中，单击**创建密钥**。
3. 在弹出的**创建密钥**对话框，填写**别名**和**描述**。

   > **注意** 选择**密钥类型**为RSA_2048；选择**密钥用途**为SIGN/VERIFY。

4. 单击**高级选项**，选择**密钥材料来源**。
   - **阿里云KMS**：密钥材料将由KMS生成。
   - **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KMS，详情请参见导入密钥材料。

     > **说明** 此时需要选中**我了解使用外部密钥材料的方法和意义**复选框。

5. 单击**确认**。

步骤三：在云安全中心配置使用KMS密钥的证明者

1. 登录云安全中心控制台。
2. 在左侧导航栏，选择**安全运营 > 容器签名**。
3. 在**证明者**页签中，单击**创建证明者**。

   创建证明者的配置信息，请参见容器签名。
4. 单击**确定**。

步骤四：在ACR中开启容器镜像自动加签功能

1. 登录容器镜像服务控制台。
2. 在**实例列表**页面，单击目标企业版实例名称或者单击目标企业版实例的**管理**。
3. 在企业版实例管理页左侧导航栏中，选择**仓库管理 > 命名空间**。

   创建一个命名空间，用于为该命名空间内的所有容器镜像启用镜像加签功能。具体操作，请参见命名空间的基本操作。
4. 为创建的命名空间启用容器镜像自动加签功能。

   添加加签规则时选择步骤三中在云安全中心创建的证明者。

   i. 在企业版实例管理页左侧导航栏中，选择**内容可信 > 镜像加签**。

   ii. 在**镜像加签**页签中，单击**添加加签规则**。

   添加加签规则配置详情，请参见配置镜像自动加签规则。

步骤五：在云安全中心启用容器镜像验签功能

通过在云安全中心添加和启用策略的方式来为Kubernetes集群中某个命名空间启用容器镜像验签功能。

1. 登录云安全中心控制台。
2. 在左侧导航栏，选择**安全运营 > 容器签名**。

3. 在**安全策略**页签，单击**添加策略**。

   添加策略的配置说明，请参见创建安全策略。



4. 单击**确定**。

步骤六：验证容器镜像自动验签功能

> **说明** 目前只支持使用digest格式的镜像。

执行以下命令验证容器镜像自动验签功能：

- 已开启验签功能的Default命名空间将拒绝部署使用未加签的容器镜像服务。
  使用Tag格式镜像的命令如下：

  ```
  kubectl -n default create deployment not-sign --image=alpine:3.11 -- sleep 10
  ```

  ```
  Error from server: admission webhook "kritis-validation-hook-deployments.grafeas.io" denied the request: image alpine:3.11 is not attested
  ```

  使用Digest格式镜像的命令如下：

  ```
  kubectl -n default create deployment not-sign --image=alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45 -- sleep 10
  ```

  ```
  Error from server: admission webhook "kritis-validation-hook-deployments.grafeas.io" denied the request: image alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45 is not attested
  ```

- 向已开启自动加签功能的容器镜像服务命名空间中推送一个镜像，然后验证可以使用这个加签的容器镜像部署服务。

  ```
  docker push kritis-demo-registry-vpc.cn-hongkong.cr.aliyuncs.com/kritis-demo/alpine:3.11
  ```

  ```
  The push refers to repository [kritis-demo-registry-vpc.cn-hongkong.cr.aliyuncs.com/kritis-demo/alpine]
  5216338b40a7: Pushed
  3.11: digest: sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45 size: 528
  ```

  放行已自动加签容器镜像的命令如下：

  ```
  kubectl -n default create deployment is-signed --image=kritis-demo-registry-vpc.cn-hongkong.cr.aliyuncs.com/kritis-demo/alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45 -- sleep 10
  ```

```
deployment.apps/is-signed created
```

相关文档

相关链接

kritis-validation-hook组件介绍，请参见kritis-validation-hook组件介绍。
kritis-validation-hook组件变更记录，请参见kritis-validation-hook。

8.3. 配置项及保密字典

8.3.1. 管理配置项

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。您可以使用配置项保存不需要加密的配置信息和配置文件。本文介绍如何管理配置项。

前提条件

已成功创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。

创建配置项

通过配置项菜单创建

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在**配置项**页面，选择该配置项所属的**命名空间**，然后单击**创建**。
6. 输入配置项名称，然后单击**+添加**，填写配置项**名称**和**值**，最后单击**确定**。
   - **配置项名称**：指定配置项的文件名，名称可以包含小写字母、数字、连字符（-）或者半角句号（.），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**：填写配置项名称和配置项的值。您也可以单击**从文件导入**创建配置项。
7. 单击**确定**。
   您可以在配置项列表中看到aliyun-config配置文件。



使用YAML创建资源

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 在**无状态**页面，单击右上角的**使用YAML创建资源**。
5. 在使用模版部署的页面，设置配置文件的信息并单击**创建**。

参数	描述
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。

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

查看配置项

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在**配置项**页面单击目标配置项名称。
6. 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。

修改配置项

修改配置文件会影响使用该配置文件的应用。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在**配置项**页面找到目标配置项，单击右侧的**编辑**。
6. 在**编辑**面板中，修改配置项的**名称**和**值**，然后单击**确定**。

> **说明** 您也可以在**配置项**页面单击**YAML编辑**，在**查看YAML**面板中里编写配置项并单击**确定**。

删除配置项

您可以删除不再使用的配置项。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
5. 在**配置项**页面找到目标配置项，单击右侧的**删除**。
6. 在**确认删除**对话框，单击**确定**。

8.3.2. 在容器组中使用配置项

本文为您介绍如何在Pod中使用配置项。

背景信息

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。

YAML示例模板如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  SPECIAL_LEVEL: very
  SPECIAL_TYPE: charm
```

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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用YAML创建资源**。
6. 选择示例模板或自定义，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
   ```

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. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用YAML创建资源**。
6. 选择示例模板或自定义，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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用YAML创建资源**。
6. 选择示例模板或自定义，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
```

在数据卷中使用配置项

1. 在控制台左侧导航栏中，单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 在**无状态**页面中，单击**使用YAML创建资源**。
5. 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
```

8.3.3. 管理保密字典

本文为您介绍如何在Kubernetes集群中管理保密字典。

前提条件

您已成功创建一个Kubernetes集群，请参见创建Kubernetes托管版集群。

背景信息

若您需要在Kubernetes集群中使用一些敏感的配置，例如密码、证书等信息时，建议使用保密字典（Secret），即密钥。

保密字典有多种类型，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类型的保密字典。Opaque类型的数据是一个map类型，要求value是base64编码格式。ACK提供一键创建保密字典的功能，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。

创建保密字典

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 保密字典**。
5. 在**保密字典**页面，单击右上角的**创建**。
6. 在**创建**面板中，配置新的保密字典。

   需要配置以下参数：

参数	说明
名称	输入保密字典的名称。名称长度为1~253字符，只能包含小写字母、数字、短划线（-）和半角句号（.）。
命名空间	选择保密字典的命名空间。
类型	包含Opaque、私有镜像仓库登录密钥以及TLS证书。
Opaque	若类型选择为Opaque，则需要配置以下参数： ○ （可选）若您输入保密字典的明文数据，请选中**对数据值进行base64编码**。 ○ 配置保密字典的数据。单击**+ 添加**，在**名称**和**值**文本框中，输入保密字典名称和值。
私有镜像仓库登录密钥	若类型选择为私有镜像仓库登录密钥，则需要配置以下参数： ○ 镜像仓库地址：输入保密字典所在镜像仓库地址。 ○ 用户名：输入镜像仓库的用户名。 ○ 密码：输入镜像仓库的密码。

参数	说明
TLS证书	若类型选择为TLS证书，则需要配置以下参数： ○ Cert：输入TLS证书。 ○ Key：输入TLS证书Key。

查看保密字典

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 保密字典**。
5. 在**保密字典**页面，找到要查看的目标保密字典，单击其右侧**操作**列的**详情**。

   进入保密字典的详情页面，您可查看该保密字典的基本信息和详细信息。

   > **说明** 单击**详细信息**区域的图标，可查看数据明文。

修改保密字典

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 保密字典**。
5. 在**保密字典**页面，找到目标保密字典，单击其右侧**操作**列下的**编辑**。
6. 在编辑页面对数据进行编辑，然后单击**确定**。

删除保密字典

> **说明** 集群创建过程中生成的保密字典请勿删除。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**配置管理 > 保密字典**。
5. 在**保密字典**页面，找到目标保密字典，单击其右侧**操作**列下的**删除**，在**确认删除**对话框中，单击**确定**。

8.3.4. 在容器组中使用保密字典

若需要在Kubernetes集群中使用一些敏感的配置，例如密码、证书等信息时，建议使用保密字典。本文介绍如何在控制台创建保密字典，通过命令行或控制台方式配置Pod数据卷及环境变量。

前提条件

- 您在Pod里使用保密字典时，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，请参见通过kubectl工具连接集群。

背景信息

您可以在Pod中使用保密字典，有多种使用场景，主要包括：

- 使用保密字典配置Pod的数据卷
- 使用保密字典设置Pod的环境变量

更多关于保密字典的信息，请参见保密字典。

创建保密字典

本示例创建保密字典secret-test。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用YAML创建资源**。
6. 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示例模板创建保密字典。

```
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界面创建保密字典，请参见管理保密字典。

使用保密字典配置Pod数据卷

以下两种配置方法您可任选其一进行配置。

通过命令行进行配置

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

1. 创建并复制以下内容到*example0.yaml*中。

```
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
```

2. 执行以下命令，配置保密字典。

```
kubectl apply -f example0.yaml
```

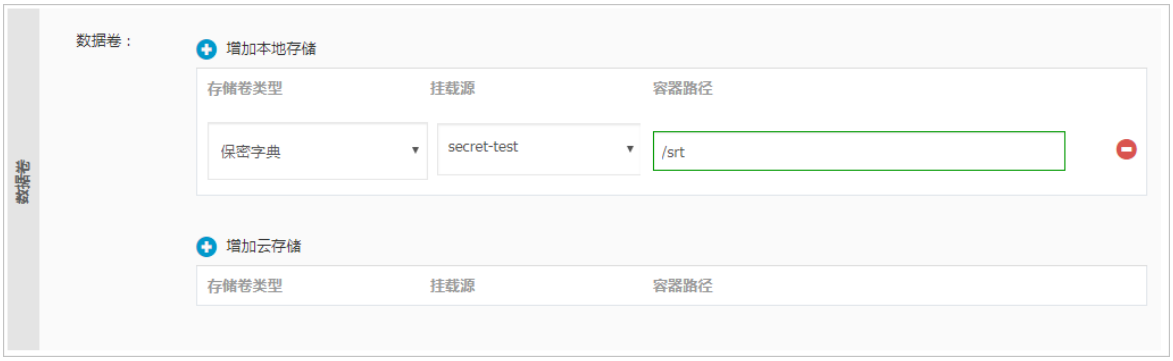
> **说明** 此处的*example0.yaml*需要替换成实际YAML文件的名称。

通过容器服务控制台进行配置

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用镜像创建**。

   详细说明，请参见创建无状态工作负载Deployment。
6. 在**应用基本信息**页签中，进行相应配置，然后单击**下一步**。
7. 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*，挂载源为创建保密字典中创建好的保密字典，容器路径为在容器中访问的路径。配置好相应信息后，单击**下一步**。

   本例中，配置示例如下所示。



8. 在**高级配置**页签中，进行相应配置，然后单击**创建**。

使用保密字典设置Pod的环境变量

以下两种配置方法您可任选其一进行配置。

通过命令行进行配置

本例中， secret-test保密字典中user和password设置为Pod的环境变量。

1. 创建并拷贝以下内容到*example1.yaml*中。

```
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
```

2. 执行如下命令，配置环境变量。

```
kubectl apply -f example1.yaml
```

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

通过容器服务控制台进行配置

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用镜像创建**。

   更多信息，请参见创建无状态工作负载Deployment。
6. 在**应用基本信息**页签中，进行相应配置，然后单击**下一步**。
7. 在**容器配置**页签，在**环境变量**区域，单击 新增，类型为**保密字典**，变量/变量引用为创建保密字典中创建好的保密字典，在分别选择使用的key并输入变量名称。

   本例中，配置示例如下所示。



8. 在**高级配置**页签中，进行相应配置，然后单击**创建**。

8.4. 应用调度部署

8.4.1. 调度应用Pod至指定节点

由于业务场景需要，有时您需要将服务部署到特定节点，或者将某些服务部署到具有SSD盘的节点上。通过容器服务管理控制台，您可以快速地将Pod调度到指定的节点上。本文介绍如何通过设置节点标签、配置模板等操作，将Pod调度到指定节点上。

前提条件

创建Kubernetes托管版集群。

背景信息

您可通过设置节点标签，然后通过配置 `nodeSelector` 强制约束Pod调度，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。

步骤一：设置节点标签

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在页面右上角单击**标签与污点管理**，进入**标签与污点管理**页面。
6. 在**标签**页签中，选择目标节点，然后单击左下角的**添加标签**。
7. 在弹出的**添加**对话框中，输入标签的**名称**和**值**，然后单击**确定**。
   - **名称**：标签名称由字母、数字、短划线（-）、下划线（_）、小数点（.）组成，且必须以字母或者数字开头和结尾。
   - **值**：标签值可以为空字符串或由字母、数字、短划线（-）、下划线（_）、小数点（.）组成，且必须以字母或者数字开头和结尾。

步骤二：部署Pod到指定节点

1. 在控制台左侧导航栏中，单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 在**无状态**页面，单击页面右上角的**使用YAML创建资源**。
5. 对模板进行相关配置，部署一个Pod。
   - **命名空间**：选择资源对象所属的命名空间，本示例中设置命名空间为default。
   - **示例模板**：本示例选择自定义模板。

   本示例的YAML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: {}
```

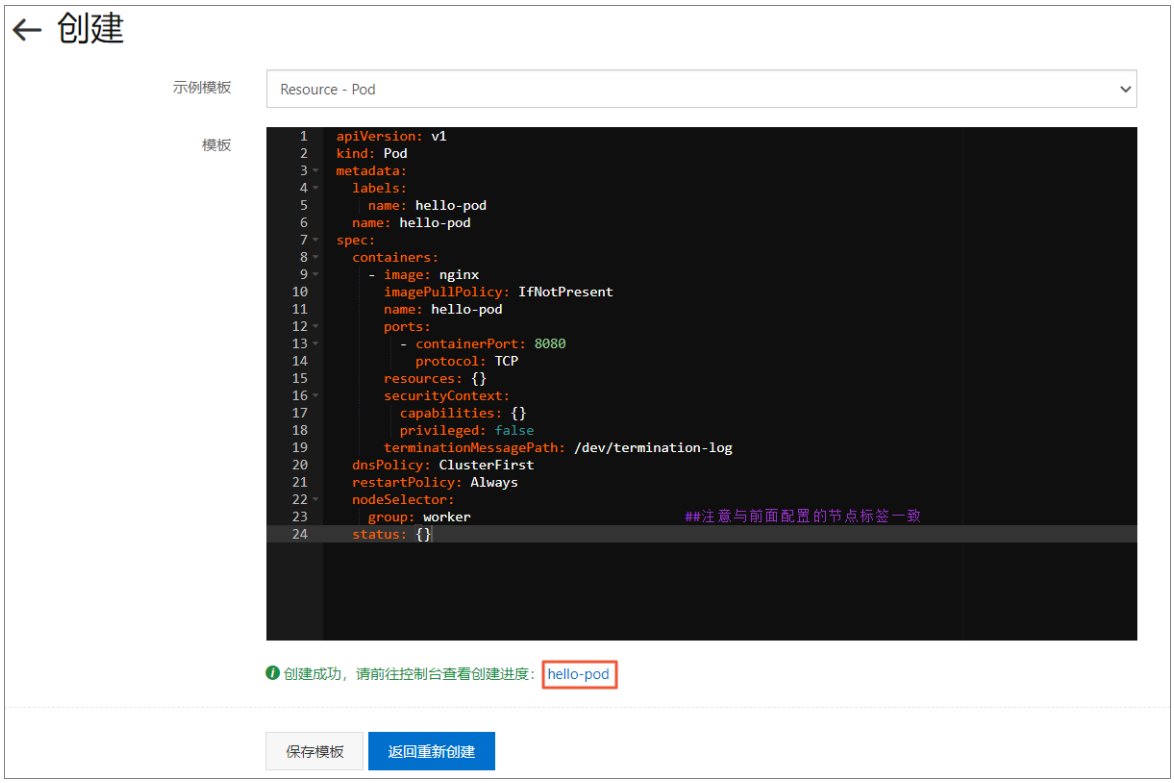
6. 单击**创建**后，会提示部署状态信息。

验证结果

您可以选择以下任意一种方式查看Pod是否已成功部署到指定节点上。

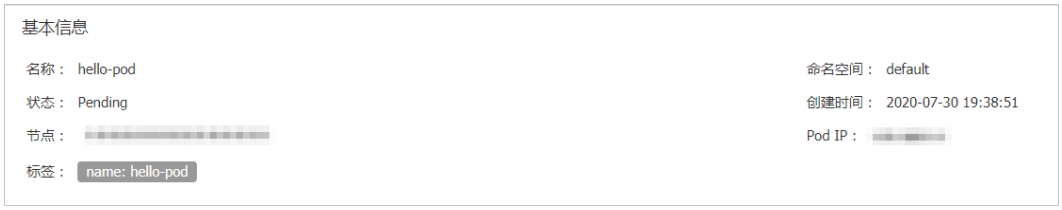
方式一：部署成功后单击页面Pod名称

1. 部署成功后，单击页面底部提示信息中的Pod名称，进入**容器组**页面。



2. 在**容器组**页面，单击目标Pod名称，进入该Pod的详情页。

   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。



方式二：容器组页面查找目标容器

1. 在控制台左侧导航栏中，单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。
4. 在**容器组**页面，单击目标Pod名称，进入该Pod的详情页。

   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。

8.4.2. 使用Descheduler组件对Pod进行调度优化

为了解决实际运行中集群资源无法充分利用或浪费的问题，可以使用Descheduler组件对集群的Pod进行调度优化，使部分不合理Pod能够重新调度到合适的节点上。本文介绍如何安装和使用Descheduler组件。

前提条件

- 已创建ACK集群（版本不低于1.14）。具体操作，请参见创建Kubernetes托管版集群。
- 已通过kubectl工具连接kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

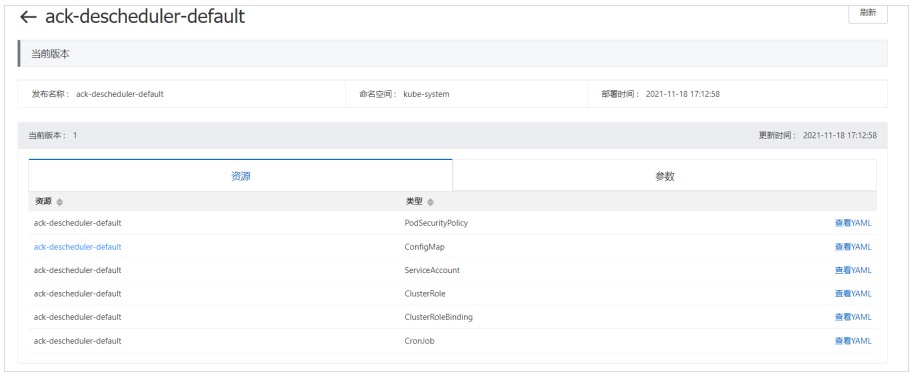
安装ack-descheduler组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，选中ack-descheduler应用。

   阿里云应用包含较多应用，您可在页面左上角搜索**ack-descheduler**，支持关键字搜索。

4. 在**详情**页面右侧**创建**区域，选择目标集群，然后单击**创建**。

   安装完成后，默认会在 `kube-system` 的 `namespace` 下部署执行周期为两分钟的Cronjob。创建成功后，会自动跳转到目标集群的**ack-descheduler-default**页面，检查安装结果。若下图中所有资源创建成功，则说明组件安装成功。



通过ack-descheduler优化Pod调度

1. 查看配置项（ConfigMap）文件，了解当前的调度策略（DeschedulerPolicy）。

```
kubectl describe cm ack-descheduler-default -n kube-system
```

预期输出：

```
Name:         descheduler
Namespace:    kube-system
Labels:       app.kubernetes.io/instance=descheduler
              app.kubernetes.io/managed-by=Helm
              app.kuberne
tes.io/name=descheduler
              app.kubernetes.io/version=0.20.0
              helm.sh/chart=descheduler-0.20.0
Annotations:  meta.helm.sh/release-name: descheduler
              meta.helm.sh/release-namespace: kube-system
Data
====
policy.yaml:
----
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
 "RemoveDuplicates":
   enabled: true
 "RemovePodsViolatingInterPodAntiAffinity":
   enabled: true
 "LowNodeUtilization":
   enabled: true
   params:
    nodeResourceUtilizationThresholds:
      thresholds:
       "cpu" : 20
       "memory": 20
       "pods": 20
      targetThresholds:
       "cpu" : 50
       "memory": 50
       "pods": 50
 "RemovePodsHavingTooManyRestarts":
   enabled: true
   params:
    podsHavingTooManyRestarts:
     podRestartThreshold: 100
     includingInitContainers: true
Events:  <none>
```

当前的调度策略如下表所示。关于 `strategies` 中策略的更多信息，请参见Descheduler。

策略	描述
RemoveDuplicates	删除重复的Pod，确保只有一个Pod与同一节点上运行的ReplicaSet、Replication Controller、StatefulSet或者Job关联。
RemovePodsViolatingInterPodAntiAffinity	此策略可确保从节点中删除违反Pod间反亲和性的Pod。

策略	描述
LowNodeUtilization	此策略会找到未充分利用的节点，并尽可能从其他节点上驱逐Pod，以便ack-descheduler重新将这些被驱逐的Pod调度到未充分利用的节点上。该策略的参数可以通过 `nodeResourceUtilizationThresholds` 字段进行配置。
RemovePodsHavingTooManyRestarts	此策略可确保从节点中删除重启次数过多的Pod。

2. 查看调度策略修改前的调度效果。
   i. 部署测试Deployment。

   nginx.yaml的命令示例如下所示。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
 name: nginx-deployment-basic
 labels:
  app: nginx
spec:
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9 #替换为实际的容器镜像，格式为：<image_name:tags>。
    ports:
    - containerPort: 80
```

   执行以下命令，部署Deployment *nginx.yaml*。

```
kubectl apply -f nginx.yaml
```

   预期输出：

```
deployment.apps/nginx-deployment-basic created
```

ii. 等待两分钟后，执行以下命令，查看Pod分布节点。

```
kubectl get pod -o wide | grep nginx
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE   IP         NODE              NOMINATED NODE  READIN
ESS GATES
nginx-deployment-basic-**1  1/1   Running  0     36s  172.25.XXX.XX1  cn-hangzhou.172.16.XXX.XX2  <n
one>      <none>
nginx-deployment-basic-**2  1/1   Running  0     11s  172.25.XXX.XX2  cn-hangzhou.172.16.XXX.XX3  <n
one>      <none>
nginx-deployment-basic-**3  1/1   Running  0     36s  172.25.XXX.XX3  cn-hangzhou.172.16.XXX.XX3  <n
one>      <none>
```

从以上预期输出可以得出， `nginx-deployment-basic-**2` 及 `nginx-deployment-basic-**3` 都被调度到 `cn-hangzhou.172.16.XXX.XX3` 节点。

> **说明** 默认ConfigMap配置下的调度结果取决于实际的集群环境，此处的结果为多种情况之一，仅作示例。

3. 修改调度策略。

为了防止多个调度策略影响调度效果，修改步骤1中的ConfigMap，只保留RemoveDuplicates策略。

> **说明** RemoveDuplicates策略是指尽可能保障有副本控制器的Pod，可以水平分布到不同的节点上。

修改后的ConfigMap文件命名为*newPolicy.yaml*，内容示例如下所示。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: descheduler
 namespace: kube-system
 labels:
  app.kubernetes.io/instance: descheduler
  app.kubernetes.io/managed-by: Helm
  app.kubernetes.io/name: descheduler
  app.kubernetes.io/version: 0.20.0
  helm.sh/chart: descheduler-0.20.0
 annotations:
  meta.helm.sh/release-name: descheduler
  meta.helm.sh/release-namespace: kube-system
data:
 policy.yaml: |-
  apiVersion: "descheduler/v1alpha1"
  kind: "DeschedulerPolicy"
  strategies:
   "RemoveDuplicates": #仅保留removeDuplicates策略。
    enabled: true
```

4. 查看调度策略修改后的调度效果。

i. 执行以下命令，部署新的调度策略。

```
kubectl apply -f newPolicy.yaml
```

预期输出：

```
configmap/descheduler created
```

ii. 等待两分钟后，执行以下命令，查看Pod分布节点。

```
kubectl get pod -o wide | grep nginx
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE   IP        NODE              NOMINATED NODE  READI
NESS GATES
nginx-deployment-basic-**1  1/1   Running  0     8m26s  172.25.XXX.XX1  cn-hangzhou.172.16.XXX.XX2
<none>      <none>
nginx-deployment-basic-**2  1/1   Running  0     8m1s   172.25.XXX.XX2  cn-hangzhou.172.16.XXX.XX1  <
none>      <none>
nginx-deployment-basic-**3  1/1   Running  0     8m26s  172.25.XXX.XX3  cn-hangzhou.172.16.XXX.XX3
<none>      <none>
```

从以上预期输出可以得出， `nginx-deployment-basic-**2` 被Descheduler重新调度到 `cn-hangzhou.172.16.XXX.XX1` 节点，三个Pod被分散到三个不同的Node节点，实现了副本之间在不同节点的均衡分布。

8.4.3. 使用应用触发器重新部署应用

阿里云容器服务ACK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。本文介绍如何使用应用触发器重新部署应用。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建Kubernetes托管版集群。
- 您已成功创建一个应用，用于创建应用触发器，并测试触发器的作用。本例中创建一个nginx应用。具体操作，请参见创建无状态工作负载Deployment。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 找到nginx应用，单击**操作**列下的**详情**。
6. 在nginx应用详情页面中，单击**触发器**页签，然后单击**创建触发器**。
7. 在弹出的**创建触发器**对话框中，选择**触发器行为**为**重新部署**，然后单击**确定**。

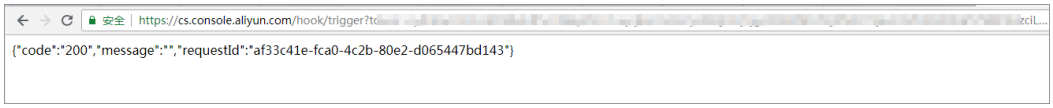


> **说明** 目前触发器行为仅支持重新部署功能。

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。



8. 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。



9. 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
```

8.4.4. 使用Helm简化应用部署

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

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。具体操作，请参见创建Kubernetes托管版集群。
  在Kubernetes集群创建的同时，Tiller将会被自动部署到集群之中，并且在所有的master节点上自动安装Helm CLI以及配置指向阿里云的Chart存储库。
- 查看您集群中Kubernetes的版本。
  仅支持Kubernetes版本1.8.4及以上的集群。对于1.8.1版本的集群，您可以在集群列表中进行**集群升级**操作。

背景信息

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，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。

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Kubernetes构建的应用，它包含以下几个基本概念。

- 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：

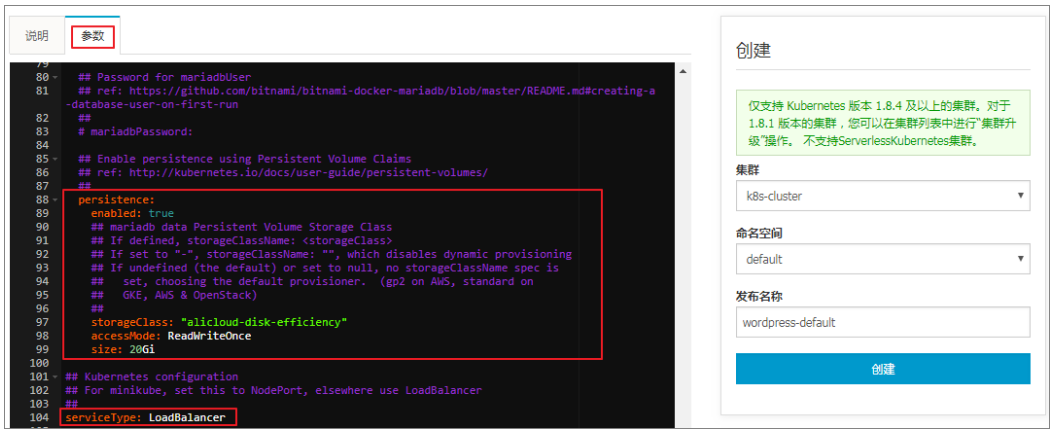
- Helm CLI是Helm客户端，可以在Kubernetes集群的master节点或者本地执行。
- Tiller是服务器端组件，在Kubernetes集群上运行，并管理Kubernetes应用程序的生命周期。
- Repository是Chart存储库，Helm客户端通过HTTP协议来访问存储库中Chart的索引文件和压缩包。

通过控制台界面部署应用

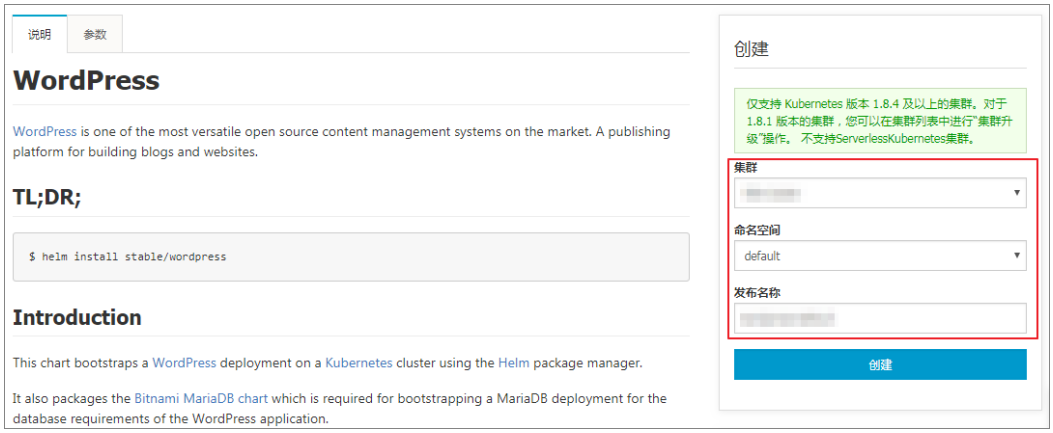
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**阿里云应用**页签中，选择一个Chart（本示例选择**ack-wordpress-sample**），单击该Chart，进入Chart详情页面。
4. 单击**参数**页签，对配置进行修改。

   本示例中使用云盘的动态数据卷绑定一个PVC。具体操作，请参见使用云盘动态存储卷。

   > **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。



5. 在页面右侧，填写部署的基本信息，然后单击**创建**。
   - **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。



6. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
7. 在服务列表中，找到对应的服务，获取HTTP和HTTPS外部端点的地址，然后单击外部端点的访问地址，进入WordPress博客发布页面。

> **说明** 访问外部端点前，请确保您已在安全组中添加外部端点的端口。

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（Helm CLI已自动安装并已配置Repository）进行操作，请参见通过SSH连接ACK专有版集群的Master节点。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。

1. 安装配置kubectl和Helm CLI。
   i. 在本地计算机上安装和配置kubectl。

      参见通过kubectl工具连接集群。
      若要查看Kubernetes目标集群的信息，键入命令 `kubectl cluster-info` 。
   ii. 在本地计算机上安装Helm。

      安装方法，参见Install Helm。
2. 部署WordPress。

   下面我们将利用Helm，来部署一个WordPress博客网站。
   i. 输入以下命令。

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

   > **说明** ACK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。

   得到以下的结果。

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

   ii. 执行以下命令查看WordPress的发布版本和Service。

   ```
   helm list
   kubectl get svc
   ```

iii. 执行以下命令查看WordPress相关的Pod，并等待其状态变为Running。

```
kubectl get pod
```

iv. 执行以下命令获得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，利用以下命令获得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)
```

v. 如需彻底删除WordPress应用，可输入以下命令。

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

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，您可以参见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。

8.4.5. 通过控制台部署、发布和监控应用

阿里云容器服务控制台提供丰富的集群及应用的管理运维功能。本文将介绍如何通过控制台部署Nginx服务、配置Ingress以及查询日志。

前提条件

- 创建Kubernetes集群。更多信息，请参见创建Kubernetes托管版集群。
- 启用日志服务组件Logtail功能。更多信息，请参见通过日志服务采集Kubernetes容器日志。

步骤一：部署Nginx应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面单击**使用YAML创建资源**。
6. 对模板进行相关配置，完成配置后单击**创建**。
   - 在页面顶部设置**命名空间**为**default**。
   - 设置**示例模板**为**Resource - basic Deployment**。
   - 配置日志采集。有关更多日志参数的配置信息，请参见通过日志服务采集Kubernetes容器日志。

在模板中的 spec 字段下添加以下内容。

> **说明** aliyun_logs_log-nginxvarlog 表示创建一个名为log-nginxvarlog的Logstore，日志采集路径为stdout的配置，从而将容器的标准输出采集到Logstore中。

```
env:
- name: aliyun_logs_log-nginxvarlog
  value: stdout
```

```
16      spec:
17      #  nodeSelector:
18      #    env: test-team
19        containers:
20        - name: nginx
21          image: nginx:1.7.9 # replace it with your exactly <image_name:tags>
22          ports:
23          - containerPort: 80
24          env:
25          - name: aliyun_logs_log-nginxvarlog
26            value: stdout
```

- 将Nginx镜像版本改为 1.9.1。

```
16      spec:
17      #  nodeSelector:
18      #    env: test-team
19        containers:
20        - name: nginx
21          image: nginx:1.9.1 # replace it with your exactly <image_name:tags>
22          ports:
23          - containerPort: 80
24          env:
25          - name: aliyun_logs_log-nginxvarlog
26            value: stdout
```

创建成功后，可以在无状态页签下看到Nginx应用。



步骤二：发布Nginx应用

1. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
2. 在**服务**页面单击**创建**。
3. 在**创建服务**对话框中配置服务，然后单击**创建**。

参数	说明
名称	设置服务名称。
类型	设置**类型**为**虚拟集群IP**。
关联	设置关联对象为nginx-deployment-basic。
端口映射	设置端口名称，**服务端口**为80，**容器端口**为80，**协议**为TCP。
注解	为该服务添加一个注解，配置负载均衡的参数，例如设置注解为 service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth:20 ，表示将该服务的带宽峰值设置为20 Mbit/s，从而控制服务的流量。更多信息，请参见通过Annotation配置负载均衡。

参数	说明
标签	为服务添加一个标签，标识该服务。

步骤三：创建Ingress服务

1. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
2. 在**路由**页面，单击**创建**。
3. 在**创建**对话框中配置路由，然后单击**创建**。

   以下为重点配置项，其他配置项的说明请参见路由配置说明。

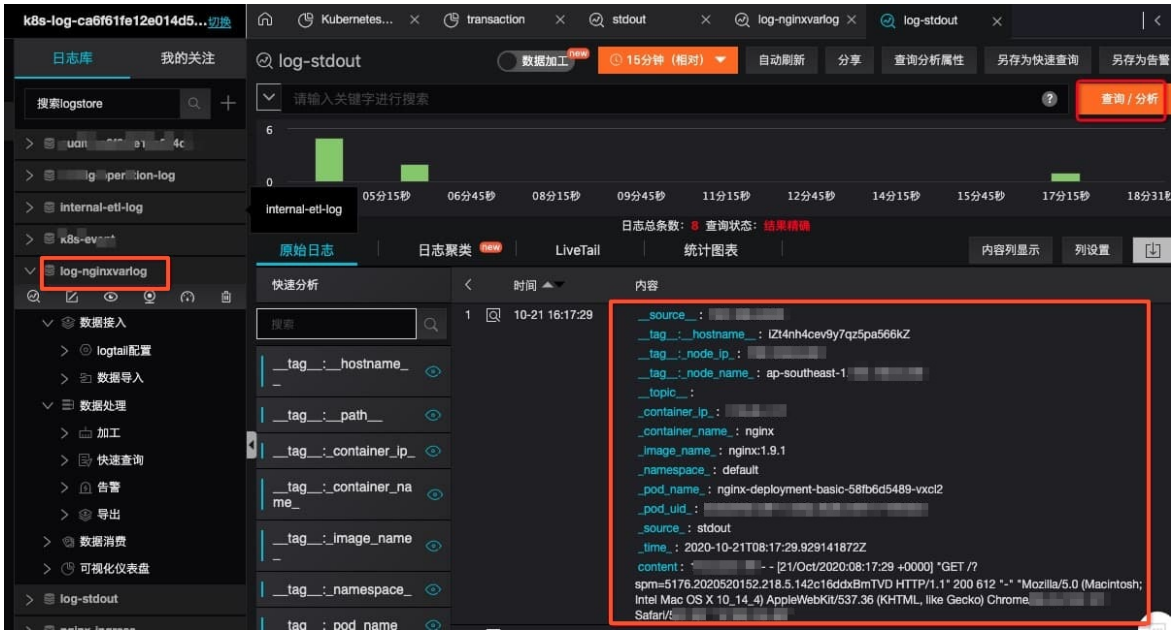
   - 名称：自定义路由名称，本例设置为ingress-demo。
   - 规则：自定义**域名**，设置**路径**为*/nginx*，设置服务名称为步骤二：发布Nginx应用的服务名称，**端口**自动绑定为80。

   在**路由**页面获取ingress-demo**端点**列的IP，在浏览器中输入*端点IP*，显示以下页面，说明路由创建成功。



步骤四：查看Nginx Access日志

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页面左侧导航栏中单击**集群信息**。
5. 在集群信息页面单击**集群资源**页签，单击**日志服务Project**的ID，跳转至日志库。
6. 在日志库中，选择log-nginxvarlog，查看最近访问日志。



8.5. 使用AHAS对应用进行高可用防护

对于部署在容器服务Kubernetes版ACK（Container Service for Kubernetes）中的Java应用，您可以使用应用高可用服务AHAS（Application High Availability Service）来对系统进行高可用防护，实现服务间限流、降级、应用系统保护等操作，保障应用24小时高可用。

前提条件

- 创建Kubernetes托管版集群
- 开通AHAS

背景信息

AHAS是一款阿里云应用高可用服务相关产品。只要为部署在ACK中的Java应用安装AHAS应用防护组件后，您无需修改任何代码，就能借助AHAS对Java应用进行全方位系统防护，针对性的对系统进行流量管控、服务降级等操作。更多信息，请参见什么是应用防护。

步骤一：安装AHAS应用防护组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的搜索框中搜索**ack-ahas-sentinel-pilot**，然后单击**ack-ahas-sentinel-pilot**。
4. 在右侧**创建**面板中选择集群，然后单击**创建**。

步骤二：为Java应用开启AHAS

1. 在容器服务管理控制台左侧导航栏单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 为Java应用开启AHAS。

   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AHAS应用防护，请按以下步骤操作。
     a. 单击右上角的**使用YAML创建资源**。

b. 选择**示例模板**，并在模板中将以下 `annotations` 添加到 `spec>template>metadata` 层级下。

```
annotations:
 #是否开启AHAS应用防护插件，on、true表示开启，off、false表示关闭。
 ahasPilotAutoEnable: "on"
 #AHAS应用名。
 ahasAppName: "<your-deployment-name>"
 #AHAS license。公网环境需配置AHAS license，VPC环境无需配置，AHAS license可以在AHAS控制台>应用防护>新应用接入中获取。
 #ahasLicense: "<your-license>"
```

完整YAML示例模板如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: agent-foo
 labels:
  name: agent-foo
spec:
 replicas: 1
 selector:
  matchLabels:
   name: agent-foo
 template:
  metadata:
   labels:
    name: agent-foo
   annotations:
    ahasPilotAutoEnable: "on"
    ahasAppName: "K8sFooTest"
    ahasNamespace: "default"
  spec:
   containers:
   - name: foo
    image: registry.cn-hangzhou.aliyuncs.com/sentinel-docker-repo/foo:0.1.1
    imagePullPolicy: Always
```

- 如需为现有应用开启AHAS应用防护，请按以下步骤操作。
  a. 选择**无状态**或**有状态**，然后在目标应用右侧**操作**列中选择**更多 > 查看YAML**。
  b. 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**。

```
annotations:
 ahasPilotAutoEnable: "on"
 ahasAppName: "<your-deployment-name>"
 #ahas namespace, 默认default
 #ahasNamespace: "default"
 #ahas license, 公网需要
 #ahasLicense: "<your-license>"
```

> **说明** 请将 `<your-deployment-name>` 替换为您的应用名称。

以下表格介绍了YAML中部分配置项和默认值。

参数	描述	默认值
image.imageTag	Pilot镜像Tag。	0.1.1
image.imagePullPolicy	镜像拉取策略，必须是Always、IfNotPresent、Never三者中的一个。	Always
controller.logLevel	Pilot日志级别，1表示INFO，2表示DEBBUG。	1
controller.region_id	目标集群所在的region，如cn-hangzhou、cn-beijing、cn-shenzhen、cn-shanghai，如果是公网，则填写为cn-public。	cn-hangzhou

后续步骤

完成上述步骤后，您就为部署在ACK中的应用开启了AHAS服务。在**无状态**或**有状态**页面，单击目标应用**操作**列的**应用流控**，进入AHAS控制台，为应用设置流量防护规则。

参考文档

为应用设置流量防护规则的具体步骤，请参见以下文档：

- 配置流控规则
- 配置隔离规则
- 配置熔断规则

8.6. 应用FAQ

本文主要为您介绍Kubernetes集群中应用的常见问题。

- 容器服务ACK应用故障排查
- 如何手动升级Helm的版本？
- 如何支持私有镜像？
- Cloud Controller Manager（CCM）组件升级检查失败
- 在Kubernetes集群中如何支持私有镜像编排容器
- 容器镜像服务源码绑定失败问题排查
- 容器镜像仓库构建服务失败问题排查
- 如何在国外地域的ACK集群中使用国内地域的容器镜像服务企业版的镜像？

如何手动升级Helm的版本？

1. 登录到Kubernetes集群。更多信息，请参见通过kubectl工具连接集群。
2. 执行以下命令安装Tiller。

   其中镜像地址可使用对应地域的VPC域名，例如，杭州地域的机器替换为 `registry-vpc.cn-hangzhou.aliyuncs.com/acs/tiller:v2.11.0` 。

   ```
   helm init --tiller-image registry.cn-hangzhou.aliyuncs.com/acs/tiller:v2.11.0 --upgrade
   ```

3. 等待tiller健康检查通过，执行 `helm version` 命令查看版本升级情况。

> **说明**
> - 这里只会升级Helm服务端版本，客户端可以通过直接下载对应的Client Binary使用。
> - 请下载阿里云支持的最新客户端版本Helm client 2.11.0。

4. Helm客户端和服务端版本都升级完毕后，执行以下命令查看Helm版本。

```
helm version
```

预期输出：

```
Client: &version.Version{SemVer:"v2.11.0", GitCommit:"2e55dbe1fdb5fdb96b75ff144a339489417b****", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.11.0", GitCommit:"2e55dbe1fdb5fdb96b75ff144a339489417b****", GitTreeState:"clean"}
```

如何支持私有镜像？

1. 执行以下命令创建Secret。

```
kubectl create secret docker-registry regsecret --docker-server=registry-internal.cn-hangzhou.aliyuncs.com --docker-username=abc****@aliyun.com --docker-password=**** --docker-email=abc****@aliyun.com
```

> **说明**
> - `regsecret`：指密钥的键名称，可自定义。
> - `—docker-server`：指Docker仓库地址。
> - `—docker-username`：指Docker仓库用户名。
> - `—docker-password`：指Docker仓库登录密码。
> - （可选）`—docker-email`：指邮件地址。

您可以通过以下两种方法操作：

- 手动配置私有镜像
  YAML文件加入密钥参数。

```
containers:
  - name: foo
    image: registry-internal.cn-hangzhou.aliyuncs.com/abc/test:1.0
imagePullSecrets:
  - name: regsecret
```

> **说明**
> - `imagePullSecrets` 是声明拉取镜像时需要指定密钥。
> - `regsecret` 必须和上面生成密钥的键名一致。
> - `image` 中的Docker仓库名称必须和 `--docker-server` 中的Docker仓库名一致。

  更多信息，请参见使用私有仓库。

- 自动配置私有镜像实现无密钥编排

> **说明** 为了避免每次使用私有镜像部署时，都需要引用密钥，您可将Secret添加到Namespace的Default Service Account中。更多信息，请参见Add ImagePullSecrets to a service account。

a. 执行以下命令，查看创建的拉取私有镜像的Secret。

```
kubectl get secret regsecret
```

预期输出：

```
NAME     TYPE                DATA    AGE
regsecret  kubernetes.io/dockerconfigjson  1     13m
```

本例中采用手动配置的方式，修改命名空间的默认服务账号**Default**，从而将此Secret作为imagePullSecret。

b. 创建一个*sa.yaml*配置文件，将服务账号Default的配置导入到该文件中。

```
kubectl get serviceaccounts default -o yaml > ./sa.yaml
```

c. 执行以下命令查看*sa.yaml*文件详情。

```
cat  sa.yaml
```

预期输出：

```
apiVersion: v1
kind: ServiceAccount
metadata:
 creationTimestamp: 2015-08-07T22:02:39Z
 name: default
 namespace: default
 resourceVersion: "243024"         #注意该项selfLink: /api/v1/namespaces/default/serviceaccounts/default。
 uid: 052fb0f4-3d50-11e5-b066-42010af0****
secrets:
- name: default-token-uudgeoken-uudge
```

d. 执行`vim sa.yaml`命令，删除resourceVersion，并增加拉取镜像的密钥配置项imagePullSecrets。修改后的配置如下所示：

```
apiVersion: v1
kind: ServiceAccount
metadata:
 creationTimestamp: 2015-08-07T22:02:39Z
 name: default
 namespace: default
 selfLink: /api/v1/namespaces/default/serviceaccounts/default
 uid: 052fb0f4-3d50-11e5-b066-42010af0****
secrets:
- name: default-token-uudge
imagePullSecrets:          #增加该项。
- name: regsecret
```

e. 执行以下命令将*sa.yaml*配置文件替换Default的服务账号配置。

```
kubectl replace serviceaccount default -f ./sa.yaml
```

预期输出：

```
serviceaccount "default" replaced
```

f. 创建Tomcat示例应用。
Tomcat编排示例*tomcat.yaml*文件如下所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tomcat-deployment
 labels:
  app: tomcat
spec:
 replicas: 1
 selector:
  matchLabels:
   app: tomcat
 template:
  metadata:
   labels:
    app: tomcat
  spec:
   containers:
   - name: tomcat
    image: registry-internal.cn-hangzhou.aliyuncs.com/abc/test:1.0        #替换为您自己的私有镜像地址P
orts。
    - containerPort: 8080
```

执行以下命令创建Tomcat应用。

```
kubectl create -f tomcat.yaml
```

g. Pod启动成功后执行以下命令，可看到预期的配置。

```
kubectl get pod tomcat-**** -o yaml
```

预期输出：

```
spec:
 imagePullSecrets:
- nameregsecretey
```

如何在国外地域的ACK集群中使用国内地域的容器镜像服务企业版的镜像？

在此种场景下，您需要在国内地域购买标准版和高级版的容器镜像服务企业版，在国外地域购买基础版的容器镜像服务企业版。

完成购买后，您需要使用同步实例的方法将国内地域的镜像同步到国外地域，具体操作，请参见同账号同步实例。在国外地域的容器镜像服务企业版中获取镜像地址，然后在国外地域的ACK集群中使用镜像地址创建应用。

如果您使用容器镜像服务个人版，同步镜像的速度将非常慢。如果您使用的是自建仓库，您需要购买GA加速。

> **说明** 自建仓库和购买GA加速的成本比购买容器镜像服务企业版高，推荐您使用容器镜像服务企业版。

关于容器镜像服务企业版的计费方式，请参见计费说明。

9.组件

9.1. 组件概述

容器服务ACK提供多种类型的组件，您可以根据业务需求部署、升级、卸载组件。本文从功能维度列举容器服务ACK管理的集群组件。

组件类型

容器服务ACK管理的集群组件类型包括系统组件和可选组件：

- 系统组件：创建ACK集群时，默认安装的组件。
- 可选组件：创建ACK集群时，可选择性安装的组件，用于扩展集群功能。

核心组件

组件名称	组件类型	描述
Kube API Server	系统组件	Kubernetes集群的总线和入口网关。
Kube Controller Manager	系统组件	Kubernetes集群内部资源的管理器。
Cloud Controller Manager	系统组件	提供Kubernetes与阿里云基础产品的对接能力，例如CLB、VPC等。

应用管理

组件名称	组件类型	描述
appcenter	可选组件	提供统一管理多集群应用部署和应用生命周期的应用中心组件。
progressive-delivery-tool	可选组件	提供应用渐进式灰度发布的组件。

日志与监控

组件名称	组件类型	描述
alicloud-monitor-controller	系统组件	ACK提供对接云监控的系统组件。
metrics-server	系统组件	ACK基于社区开源监控组件进行改造和增强的监控采集和离线组件，并提供Metrics API进行数据消费，提供HPA的能力。
ack-node-problem-detector	可选组件	ACK基于社区开源项目进行改造和增强的集群节点异常事件监控组件，以及对接第三方监控平台功能的组件。
ags-metrics-collector	可选组件	为基因计算客户使用的监控服务组件，可以通过该组件监控基因工作流中各个节点资源使用的详细信息。
ack-arms-prometheus	可选组件	使用ARMS Prometheus实现容器服务集群监控。
logtail-ds	可选组件	使用日志服务采集Kubernetes容器日志。
logtail-windows	可选组件	ACK集群上使用的容器日志收集插件，用于在阿里云上配合SLS服务对Windows容器进行日志收集。

存储

组件名称	组件类型	描述
csi-plugin	可选组件	支持数据卷的挂载、卸载功能。 创建集群时，如果选择CSI插件实现阿里云存储的接入能力的话，默认安装该组件。
csi-provisioner	可选组件	支持数据卷的自动创建能力。 创建集群时，如果选择CSI插件实现阿里云存储的接入能力的话，默认安装该组件。
storage-operator	可选组件	用于管理存储组件的生命周期。
alicloud-disk-controller	可选组件	支持自动创建云盘卷。
Flexvolume	可选组件	Kubernetes社区较早实现的存储卷扩展机制。Flexvolume支持数据卷的挂载、卸载功能。 创建集群时，如果选择Flexvolume插件实现阿里云存储的接入能力的话，默认安装该组件。

网络

组件名称	组件类型	描述
CoreDNS	系统组件	ACK集群中默认采用的DNS服务发现插件，其遵循Kubernetes DNS-Based Service Discovery规范。
Nginx Ingress Controller	系统组件	Nginx Ingress Controller解析Ingress的转发规则。Ingress Controller收到请求，匹配Ingress转发规则转发到后端Service。
managed-kube-proxy-windows	系统组件	ACK托管版集群上使用的容器化kube-proxy，用于管理Windows节点上Service的访问入口，包括集群内Pod到Service的访问和集群外访问Service。
Terway	可选组件	阿里云开源的基于专有网络VPC的容器网络接口CNI（Container Network Interface）插件，支持基于Kubernetes标准的网络策略来定义容器间的访问策略。您可以通过使用Terway网络组件实现Kubernetes集群内部的网络互通。 创建集群时，如果选择Terway网络插件实现集群内部网络互通的话，默认安装该组件。
Flannel	可选组件	一种容器网络接口CNI（Container Network Interface）插件，在阿里云上使用的Flannel网络模式采用阿里云VPC模式。 创建集群时，如果选择Flannel网络插件实现集群内部网络互通的话，默认安装该组件。
ACK NodeLocal DNSCache	可选组件	基于社区开源项目NodeLocal DNSCache的一套DNS本地缓存解决方案。
kube-flannel-ds-windows	可选组件	ACK托管版集群上使用的容器网络插件，用于构建适合Windows容器通讯的L2Bridge集群网络。

安全

组件名称	组件类型	描述
aliyun-acr-credential-helper	系统组件	一个可以在ACK集群中免密拉取ACR默认版或企业版私有镜像的组件。

组件名称	组件类型	描述
gatekeeper	可选组件	帮助您方便地管理和应用集群内的Open Policy Agent（OPA）策略，实现命名空间标签管理等功能。
kritis-validation-hook	可选组件	部署可信容器环节中进行容器镜像签名验证的关键组件。
security-inspector	可选组件	实现安全巡检功能的关键组件。
ack-kubernetes-webhook-injector	可选组件	一款可以从多种阿里云产品白名单中动态加入或移出Pod IP的K8s组件，免去手动配置Pod IP到云产品白名单的操作。

其他

组件名称	组件类型	描述
ack-arena	可选组件	对开源Arena的安装做进一步简化，能够实现在控制台一键安装Arena的目标。
ack-cost-exporter	可选组件	容器服务ACK成本分析功能进行数据处理的插件。
ack-kubernetes-cronhpa-controller	可选组件	使用ack-kubernetes-cronhpa-controller实现应用负载定时伸缩。
ack-virtual-node	可选组件	基于社区开源项目Virtual Kubelet，扩展了对Aliyun Provider的支持，并做了大量优化，实现Kubernetes与弹性容器实例ECI的无缝连接。
Intel SGX AESM	可选组件	Intel® SGX Architectural Enclave Service Manager （Intel® SGX AESM）是Intel® SGX的系统组件，主要提供了SGX Enclave启动支持，密钥配置、远程认证等服务。
aliyun-acr-acceleration-suite	可选组件	提供镜像按需加载加速能力的客户端插件，以DaemonSet形式部署在Worker节点上。
migrate-controller	可选组件	基于开源项目Velero开发的一个Kubernetes应用迁移的组件。
resource-controller	可选组件	实现动态控制Pod资源的关键组件，使用ACK Pro集群的CPU拓扑感知调度需要安装此组件。
sandboxed-container-controller	可选组件	安全沙箱运行时提供的专用控制器组件，旨在增强和扩展安全沙箱的基本功能。
sandboxed-container-helper	可选组件	为安全沙箱提供诊断和运维的组件。
sgx-device-plugin	可选组件	由阿里云容器服务团队和蚂蚁金服安全计算团队针对Intel SGX联合开发的Kubernetes Device Plugin，可以让您更容易的在容器中使用SGX。
directx-device-plugin-windows	可选组件	ACK集群上使用的DirectX设备插件。

相关文档

- 管理组件

9.2. 管理组件

本文为您介绍如何升级、安装和卸载组件。

前提条件

创建Kubernetes托管版集群或创建ACK Pro版集群

背景信息

很多时候集群已经是最新版本，但某些组件需要进行更小粒度的版本操作。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 系统组件管理**。
4. 在**组件管理**页面，您可以进行以下操作：
   - 选择需要安装的组件，单击**安装**。
   - 选择需要卸载的组件，单击**卸载**。
   - 选择需要升级的组件，单击**升级**。
   - 选择需要修改管控面参数的组件，单击**配置**。

> **说明** 仅Pro版集群支持自定义托管组件Kube API Server和Kube Controller Manager（KCM）的参数。更多信息，请参见自定义ACK Pro集群的管控面参数。

9.3. metrics-server组件常见问题

metrics-server是离线监控数据组件，提供查看集群离线监控数据功能。本文介绍使用metrics-server组件时遇到的常见问题。

kubectl top pod/node全部无数据

请按照以下方式进行预检查。

1. 执行以下命令，检查metrics-server的API Service是否正常。

   ```
   kubectl get apiservices
   ```

   ```
   v1beta1.flowcontrol.apiserver.k8s.io   Local                  True   24d
   v1beta1.metrics.k8s.io                 kube-system/metrics-server   True   24d
   v1beta1.networking.k8s.io              Local                  True   24d
   v1beta1.node.k8s.io                    Local                  True   24d
   ```

   返回结果中 `v1beta1.metrics.k8s.io` 显示 `True`，说明metrics-server的API Service是正常的。

2. （可选）如果metrics-server的API Service不正常，在metrics-server所在的Node节点上执行以下命令，检查metrics-server的443端口与8082端口是否可以在集群中正常访问。

   ```
   curl -v 127.0.0.1:8082/apis/metrics/v1alpha1/nodes
   ```

   执行以上命令，能正常返回数据，说明metrics-server的443端口与8082端口可以在集群中正常访问。

3. （可选）如果metrics-server的443端口与8082端口无法在集群中正常访问，重启metrics-server。

   您可以通过删除metrics-server的Pod的方式重启metrics-server。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

v. 在**无状态**页面顶部设置**命名空间**为kube-system，单击metrics-server。

vi. 在**容器组**页签下选择metrics-server的Pod**操作**列下的**更多 > 删除**。

vii. 在**提示**对话框单击**确定**。

按上述说明检查后，没有发现问题，请按照以下工单模板提交工单。

工单模板

1. API Service是否正常？
   是
2. metrics-server 443与8082端口是否可达？
   是
3. 提供集群ID。

kubectl top pod/node部分无数据

请按照以下方式进行预检查。

- 检查是特定的Node上所有Pod无数据，还是特定的Pod无数据。如果是特定的Node上所有Pod无数据，请检查节点是否存在时区漂移，可以通过NTP服务器的date命令进行时区校验。
- 检查metrics-server Pod到特定的Node的10255端口的网络连通性。

按上述说明检查后，没有发现问题。请按照以下工单模板提交工单。

工单模板

1. 单个Node上的Pod是否全部无数据？
   是
2. 节点时区是否有漂移？
   无
3. metrics-server到指定节点的连通性是否可达？
   是

HPA无法获取metrics数据

请按照以下方式进行预检查。

检查对应的Pod执行 `kubectl top pod pod-id` 的结果。如果数据异常，请参考上述kubectl top pod/node部分无数据和kubectl top pod/node全部无数据的检查方法进行检查。

按上述说明检查后，没有发现问题。请按照以下工单模板提交工单。

工单模板

1. 监控数据是否有异常？
   无
2. 执行 `kubectl describe hpa hpa-name` ，提交元数据信息。

滚动发布时HPA额外弹出多余的Pod

请按照以下方式进行预检查。

检查metrics-server是否升级到了最新的版本。如果版本没有问题，在kube-system命名空间下的metrics-server配置启动参数。

```
--metric-resolution=15s
--enable-hpa-rolling-update-skipped=true
```

按上述说明检查后，没有发现问题。请按照以下工单模板提交工单。

工单模板

1. 检查metrics-server的版本是否为最新？
   是
2. 检查配置参数是否已经增加防误弹能力？
   是
3. 执行 `kubectl describe hpa hpa-name` ，提交HPA的描述。

9.4. ack-arms-prometheus组件常见问题

ack-arms-prometheus组件是ARMS Prometheus监控在ACK集群中的监控组件。本文介绍安装ack-arms-prometheus组件时遇到的常见问题。

提示Component Not Installed后继续安装ack-arms-prometheus组件，安装失败

- 检查是否已经安装ack-arms-prometheus组件。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 在集群管理页面左侧导航栏选择**应用 > Helm**。
  在**Helm**页面检查是否已有ack-arms-prometheus。
  - 若**Helm**页面已有ack-arms-prometheus，您需要进行以下操作：
    您需要在**Helm**页面删除ack-arms-prometheus，并在**组件管理**页面重新安装ack-arms-prometheus。关于安装ack-arms-prometheus的具体操作，请参见管理组件。
  - 若**Helm**页面没有ack-arms-prometheus，您需要进行以下操作：
    a. 若没有ack-arms-prometheus，说明删除ack-arms-prometheus helm有资源残留，需要手动清理。
    关于删除ack-arms-prometheus残留资源的具体操作，请参见ARMS Prometheus监控常见问题。
    b. 在**组件管理**页面安装ack-arms-prometheus。关于安装ack-arms-prometheus的具体操作，请参见管理组件。
    c. 执行以上步骤，仍无法安装ack-arms-prometheus，请提交工单。
- 检查arms-prometheus-ack-arms-prometheus的日志是否有报错。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 在集群管理页面左侧导航栏选择**工作负载 > 无状态**。
  v. 在**无状态**页面顶部设置**命名空间**为**arms-prom**，然后单击arms-prometheus-ack-arms-prometheus。
  vi. 单击**日志**页签，查看日志中是否有报错。
  若日志中出现报错，请提交工单。
- 检查Agent是否安装报错。
  i. 登录ARMS控制台。
  ii. 在顶部菜单栏，选择地域。
  iii. 在控制台左侧导航栏单击**Prometheus监控**。
  iv. 在**Prometheus监控**页面，单击K8s集群名称。
  v. 在左侧导航栏单击**设置**，在右侧页面单击**设置**页签。
  在**设置**页签下可以查看健康检查结果。若有报错，请提交工单。

安装ack-arms-prometheus组件时报错xxx in use

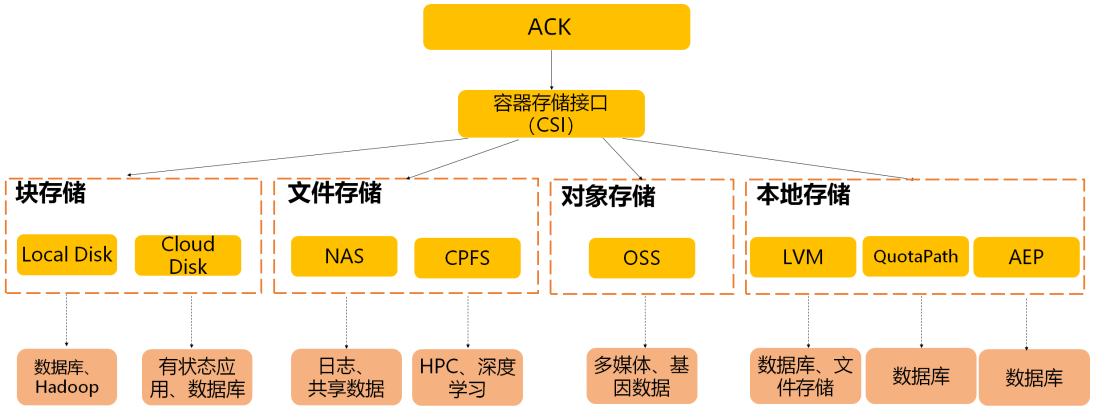
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页面左侧导航栏选择**应用 > Helm**。
   在**Helm**页面检查是否已有ack-arms-prometheus。
   - 若**Helm**页面已有ack-arms-prometheus，您需要进行以下操作：
     您需要在**Helm**页面删除ack-arms-prometheus，并在**组件管理**页面重新安装ack-arms-prometheus。关于安装ack-arms-prometheus的具体操作，请参见管理组件。
   - 若**Helm**页面没有ack-arms-prometheus，您需要进行以下操作：
     a. 若没有ack-arms-prometheus，说明删除ack-arms-prometheus helm有资源残留，需要手动清理。关于删除ack-arms-prometheus残留资源的具体操作，请参见ARMS Prometheus监控常见问题。
     b. 在**组件管理**页面安装ack-arms-prometheus。关于安装ack-arms-prometheus的具体操作，请参见管理组件。
     c. 执行以上步骤，仍无法安装ack-arms-prometheus，请提交工单。

10.存储-CSI

10.1. 存储CSI概述

阿里云容器服务ACK的容器存储功能基于Kubernetes容器存储接口（CSI），深度融合阿里云存储服务云盘EBS，文件存储NAS / CPFS，以及对象存储OSS，本地盘等，并完全兼容Kubernetes原生的存储服务，例如EmptyDir、HostPath、Secret、ConfigMap等存储。本文介绍ACK存储CSI的概览、ACK存储CSI支持的功能、CSI使用授权、CSI使用限制等。

ACK容器存储概览



容器服务ACK支持Pod自动绑定阿里云云盘、NAS、 OSS、CPFS、本地卷等存储服务，每种存储卷的主要特点及应用场景如下表。

阿里云存储	静态数据卷	动态数据卷	默认部署	主要特点	应用场景
阿里云云盘	支持	支持存储类alicloud-disk-available，alicloud-disk-efficiency，alicloud-disk-essd，alicloud-disk-ssd，alicloud-disk-topology	是	非共享存储，每个云盘只能在单个节点挂载。	• 高IO、低延时场景。 云盘块存储设备适用于对IO、延迟要求较高的应用场景，例如：数据库、中间件等服务。 • 非共享场景 云盘卷是独享模式，同时只能给一个Pod使用，在非共享数据业务场景可以考虑使用云盘卷。 更多信息，请参见云盘存储卷概述。

阿里云存储	静态数据卷	动态数据卷	默认部署	主要特点	应用场景
阿里云NAS	支持	支持 存储类 alibabacloud-cnfs-nas	是	共享存储，可提供高性能、高吞吐存储服务。	• 共享数据场景 NAS支持多个Pod同时访问一份数据，如果您有共享数据需求，推荐使用NAS。 • 大数据分析场景 NAS提供了较高的数据吞吐能力，能满足大批量作业对共享存储访问的需求。 • Web应用场景 为Web应用、内容管理系统提供存储支撑。 • 保存日志场景 如果您希望将日志保存到存储卷，推荐使用NAS数据卷。 更多信息，请参见NAS存储卷概述。
阿里云OSS	支持	不支持	是	共享存储，用户态文件系统。	• 媒体、图片等媒体文件只读场景 您可以通过OSS数据卷读取上述文件类型。 • 网页、应用配置文件只读场景 OSSFS网络性能欠佳，可以支撑一些小文件的读场景。 **说明** OSS数据卷通过OSSFS挂载，是一种通过FUSE方式模拟实现的用户态文件系统。对于写场景稳定性欠佳，因此在写场景中推荐使用其他存储卷类型。 更多信息，请参见OSS存储卷概述。
阿里云CPFS	支持	支持	否	高性能、高带宽、共享存储。	• 基因计算、大数据分析场景 CPFS提供的高吞吐能力，满足大规模集群对存储超高性能的需求。 • 数据缓存访问场景 您可以将低速存储数据先下载到CPFS卷，然后供应用高速访问。 更多信息，请参见CPFS静态卷和CPFS动态卷。
本地卷	支持	支持	否	低延迟、高IO，非高可用。	• 本地临时文件存储 如果您期望将非高可用数据保存在本地，可以通过LVM卷进行存储管理。 • IO要求高、自带高可用场景 有些业务对IO要求很高，且应用自带高可用方案，例如：MySQL；可以使用LVM卷。 • 大数据分析场景 Spark等类型的大数据分析场景，可以用LVM卷替代共享本地盘的方案。 更多信息，请参见本地存储卷概述。

> **说明** ACK部署的CSI插件支持静态存储卷挂载和动态存储卷挂载2种方式，在静态存储卷挂载的方式中，通常需要手动编辑和创建一个PV和PVC进行挂载。当需要PV和PVC数量很大的时候，您可以创建动态存储卷进行挂载。PV和PVC的概念如下：
>
> *PersistentVolume（PV）*
> 集群内的存储资源。PV独立于Pod的生命周期，可根据不同的StorageClass类型创建不同类型的PV。
>
> *PersistentVolumeClaim（PVC）*
> 集群内的存储请求。例如，PV是Pod使用的节点资源，PVC则声明使用PV资源。当PV资源不足时，PVC可动态创建PV。

ACK容器存储功能

ACK不同集群形态与各存储服务能力的对应关系如下表。

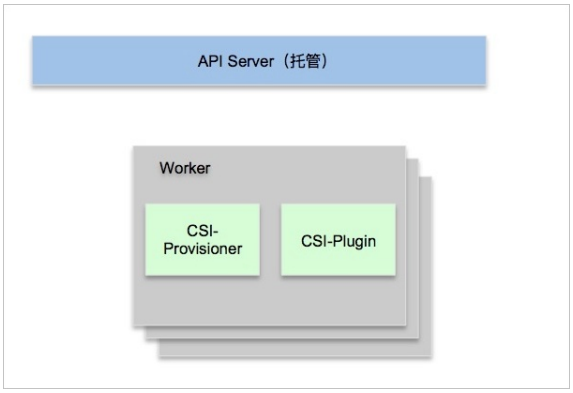
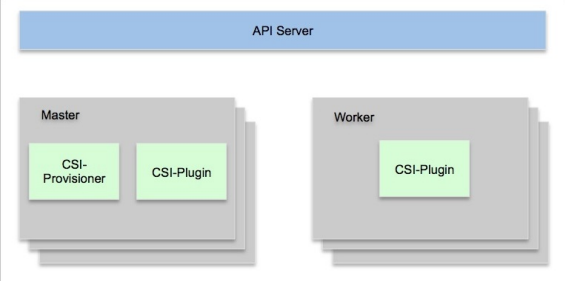
存储分类	存储能力	ACK集群（Linux）	ASK集群	注册集群（混合云/多云）	ACK@Edge	ACK（Windows）	ACK专有云	ACK安全容器（Sandbox）
块存储	云盘挂载和卸载	✔	✔	✘	✘	✔	✔	✔
	在线扩容（Resize）	✔	✘	✘	✘	✘	✔	✘
	快照（Snapshot）	✔	✔	✘	✘	✘	✔	✔
	容器IO监控	✔	✘	✘	✘	✘	✔	✘
	文件系统	支持XFS、Ext4、dBFS	支持XFS、Ext4	✘	✘	支持NTFS	支持XFS、Ext4	支持XFS、Ext4
	块、裸设备	✔	✘	✘	✘	✘	✔	✘
	快照恢复	✔	✔	✘	✘	✘	✔	✔
	云盘队列设置	✔	✘	✘	✘	✘	✔	✘
	数据加密CMK和BYOK	✔	✔	✘	✘	✔	✔	✔
	多可用区感知	✔	✘	✘	✘	✔	✔	✔
	用户自定义标签	✔	✔	✘	✘	✔	✔	✔
	跨主机迁移	✔	✔	✘	✘	✔	✔	✔
	NAS创建，挂载，卸载	✔	✔	✔	✔	✘	✔	✔
	Samba挂载卸载	✘	✘	✔	✘	✔	✘	✘

存储分类	存储能力	ACK集群（Linux）	ASK集群	注册集群（混合云／多云）	ACK@Edge	ACK（Windows）	ACK专有云	ACK安全容器（Sandbox）
文件存储	高吞吐并行文件系统（CPFS）	✔	✘	✔	✘	✘	✘	✘
	回收站（CNFS）	✔	✘	✘	✘	✘	✘	✘
	动态数据卷-子目录／共享目录（CNFS）	✔	✘	✔	✔	✔	✔	✔
	数据加密CMK（CNFS，极速NAS）	✔	✘	✘	✘	✘	✔	✔
	容量控制Quota（CNFS）	✔，只支持托管版	✘	✘	✘	✘	✘	✘
	容量／IO监控（CNFS）	✔	✘	✔	✘	✘	✘	✘
	在线扩容（CNFS）	✔	✘	✔	✘	✘	✘	✘
对象存储	OSS挂载和卸载	✔	✔	✔	✔	✘	✔	✔
	数据加密BYOK	✔	✔	✔	✔	✘	✔	✔
本地存储	块存储LVM管理	✔	✘	✔	✔	✘	✔	✘
	VolumeGroup自动化配置	✔	✘	✔	✔	✘	✔	✘
	节点LVM容量感知调度	✔	✘	✔	✔	✘	✔	✘
	PMEM Direct Mem	✔	✘	✘	✘	✘	✘	✘
	PMEM LVM卷	✔	✘	✘	✘	✘	✘	✘

CSI部署架构

CSI插件包括CSI-Plugin和CSI-Provisioner两部分。ACK托管版集群和ACK专有版集群中CSI的部署架构图如下表。

> **说明** ACK托管版和专有版集群默认部署CSI插件。针对ASK集群和边缘集群，您需自行部署CSI插件。

ACK托管版集群	ACK专有版集群
 ACK托管版集群中CSI插件CSI-Provisioner和CSI-Plugin都运行在Worker节点中。	 ACK专有版集群中CSI-Provisioner部署在Master节点上。CSI-Plugin以DaemonSet负载方式自动部署在Master、Worker中。

CSI使用授权

CSI插件在数据卷的挂载、卸载、创建、删除等操作上都需要您授予访问其他产品资源的权限。CSI插件支持直接使用AccessKey授权，也支持使用RAM角色授权的方式（默认）进行授权。两种授权方式的具体内容如下表。

使用账户AccessKey信息授权	使用RAM角色授权
• 您可以直接在CSI部署模板中写入AccessKey信息。 • 您也可以通过Secret方式将AccessKey信息传入环境变量。	ACK集群存储插件CSI使用**AliyunCSManagedCsiRole**角色来访问您在其他云产品中的资源。更多信息，请参见AliyunCSManagedCsiRole。关于如何为RAM角色授权，请参见为RAM角色授权。 • 托管集群 CSI使用的RAM角色权限的令牌被保存在名为**addon.csi.token**的Secret中，CSI插件挂载该Secret即可以实现RAM角色授权，并访问OpenAPI。 • 专有集群 CSI使用其Pod所在的ECS节点的RAM角色。

CSI插件使用限制

您在使用ACK存储CSI插件时需要注意CSI插件本身的使用限制，也需要注意使用CSI插件时涉及的阿里云存储产品的使用限制。

- **阿里云存储产品使用限制**

阿里云存储产品	使用限制

阿里云存储产品	使用限制
云盘数据卷	○ 每个节点最多挂载15个云盘数据卷。 ○ 每个云盘数据卷只能给一个Pod使用，不能共享。 ○ 不是每种云盘类型都能挂载所有ECS实例。更多信息，请参见实例规格族。 ○ 包年包月的云盘不能进行挂载、卸载操作，不能用于容器数据卷场景。 ○ 云盘只能挂载到相同可用区的ECS实例，不可以跨可用区挂载。 ○ 建议使用StatefulSet方式使用云盘卷，不建议使用Deployment。 **说明** Deployment是无状态服务，当Pod在重启时，无法保证新Pod启动时间和老Pod结束时间之间的时间叠加，且Deployment多副本模式不能为每个Pod配置独立的存储卷。 ○ 每个卷的最小容量是20 GiB。
NAS数据卷	○ NAS卷只能挂载到相同VPC的ECS实例，不同VPC网络不能挂载。 ○ 每个用户的NAS卷有数量限制，如果您要申请较多数量NAS盘，请提交工单至文件存储团队。
OSS数据卷	不推荐您使用OSS数据卷进行数据写操作，写数据场景请选择其他存储。 **说明** OSS数据卷通过OSSFS挂载，是一种通过FUSE方式模拟实现的用户态文件系统。对于写场景稳定性欠佳，因此在写场景中推荐使用其他存储卷类型。
CPFS数据卷	○ CPFS驱动对操作系统内核有较强的版本匹配需求，部署完CPFS环境后，请不要升级内核。 ○ 当前只支持为没有CPFS驱动的节点安装CPFS驱动，而不支持升级CPFS驱动。 **说明** CSI-CPFS插件属于Kubernetes CSI组件，负责挂载CPFS盘到Kubernetes Pod，供应用使用。 CPFS驱动：是CPFS存储产品对应的客户端驱动，为内核层面负责CPFS存储协议的存储实现。 CPFS CSI 插件和CPFS驱动之间的关联： ○ 当节点没有安装CPFS驱动时，如果部署CSI-CPFS插件，会默认安装CPFS驱动。 ○ 当节点已经安装CPFS驱动时，如果部署CSI-CPFS插件，不会安装和升级CPFS驱动。

- **ACK存储CSI插件使用限制**
  阿里云CSI插件已在社区开源，主要的支撑场景为ACK集群。对于非ACK集群（非阿里环境集群、阿里云自建集群），由于集群配置、权限管理、网络差异等，插件无法无缝支撑。如果您是这样的集群环境，可以通过阅读源码、修改配置等方式进行适配。更多信息，请参见alibaba-cloud-csi-driver。

版本说明

使用CSI插件要求Kubernetes版本为1.14及以上，且kubelet运行参数 `--enable-controller-attach-detach` 需要为 `true` 。

CSI插件安装与升级

关于CSI存储插件的安装与升级，请参见安装与升级CSI组件。

CSI和Flexvolume存储插件的区别

插件名称	插件特性	参考文档
Flexvolume	Flexvolume插件是Kubernetes社区较早实现的存储卷扩展机制。ACK从上线起，即支持Flexvolume类型数据卷服务。Flexvolume插件包括以下三部分。 • Flexvolume：负责数据卷的挂载、卸载功能。ACK默认提供云盘、NAS、OSS三种存储卷的挂载能力。 • Disk-Controller：负责云盘卷的自动创建能力。 • Nas-Controller：负责NAS卷的自动创建能力。	有关Flexvolume的详细概述，请参见Flexvolume概述。 有关如何升级Flexvolume存储插件，请参见管理组件。
CSI	CSI插件是当前Kubernetes社区推荐的插件实现方案。ACK集群提供的CSI存储插件兼容社区的CSI特性。CSI插件包括以下两部分： • CSI-Plugin：实现数据卷的挂载、卸载功能。ACK默认提供云盘、NAS、OSS三种存储卷的挂载能力。 • CSI-Provisioner：实现数据卷的自动创建能力，目前支持云盘、NAS两种存储卷创建能力。	有关CSI的详细概述，请参见CSI概述和alibaba-cloud-csi-driver。 有关如何升级CSI存储插件，请参见CSI存储插件升级指南。

> **说明**
> - 在创建集群的时候确定插件类型。
> - 不支持CSI和Flexvolume插件在同一个集群中使用。
> - 不支持Flexvolume转变到CSI插件。

使用推荐

- 针对新建集群，推荐您使用CSI插件。ACK会跟随社区持续更新CSI插件的各种能力。
- 针对已经创建的集群，仍然使用已经安装的存储插件类型。ACK会持续支持Flexvolume插件。

如何判断集群的存储插件模式

- 方式一：通过控制台查看节点注释
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
  v. 在目标节点**操作**列选择**更多 > 详情**。
  vi. 在**基本信息**中查看节点注释。
  如果有 `volumes.kubernetes.io/controller-managed-attach-detach: true` ，表示集群存储插件为CSI；若没有 `volumes.kubernetes.io/controller-managed-attach-detach: true` ，表示集群存储插件为Flexvolume。
- 方式二：通过命令查看kubelet参数
  执行以下命令查看kubelet参数。

```
ps -ef | grep kubelet
```

预期输出：

```
--enable-controller-attach-detach=true
```

若 `--enable-controller-attach-detach` 的值为 `true` ，则表示集群存储插件为CSI；若 `--enable-controller-attach-detach` 的值为 `false` ，则表示集群存储插件为Flexvolume。

相关文档

- 存储基础知识
- 云盘存储卷概述
- NAS存储卷概述
- OSS存储卷概述
- CPFS静态卷
- CPFS动态卷
- 本地存储卷概述
- AEP非易失性数据卷概述

10.2. 存储基础知识

阿里云容器服务ACK使用Kubernetes编排系统作为集群、应用、存储、网络等模块的管理平台。本文为您介绍ACK容器存储相关的基础知识，以便在使用容器服务的存储能力时，了解相应模块的基础知识和使用原则。

数据卷（Volume）

因为容器中的文件在磁盘上是临时存放的，所以Kubernetes定义了数据卷（Volume）以解决容器中的文件因容器重启而丢失的问题。数据卷（Volume）是Pod与外部存储设备进行数据传递的通道，也是Pod内部容器间、Pod与Pod间、Pod与外部环境进行数据共享的方式。

数据卷（Volume）定义了外置存储的细节，并内嵌到Pod中作为Pod的一部分。其实质是外置存储在Kubernetes系统的一个资源映射，当负载需要使用外置存储的时候，可以从数据卷（Volume）中查到相关信息并进行存储挂载操作。

> **说明** 数据卷（Volume）生命周期和Pod一致，即Pod被删除的时候，数据卷（Volume）也一起被删除（Volume中的数据是否丢失取决于Volume的具体类型）。

Kubernetes提供了非常丰富的Volume类型，您在ACK可以使用的常用的Volume类型如下：

数据卷（Volume）分类	描述
本地存储	适用于本地存储的数据卷，例如HostPath、emptyDir等。本地存储卷的特点是数据保存在集群的特定节点上，并且不能随着应用漂移，节点停机时数据即不再可用。
网络存储	适用于网络存储的数据卷，例如Ceph、GlusterFS、NFS、iSCSI等。网络存储卷的特点是数据不在集群的某个节点上，而是在远端的存储服务上，使用存储卷时需要将存储服务挂载到本地使用。
Secret和ConfigMap	Secret和ConfigMap是特殊的数据卷，其数据是集群的一些对象信息，该对象数据以卷的形式被挂载到节点上供应用使用。
PVC	一种数据卷定义方式，将数据卷抽象成一个独立于Pod的对象，这个对象定义（关联）的存储信息即存储卷对应的真正存储信息，供Kubernetes负载挂载使用。

> **说明** CSI和Flexvolume是数据卷的两种扩展方式。每种扩展方式都可再细化成不同的存储类型。关于存储CSI的更多信息，请参见存储CSI概述。

数据卷（Volume）使用原则

- 一个Pod可以挂载多个数据卷（Volume）。

- 一个Pod可以挂载多种类型的数据卷（Volume）。
- 每个被Pod挂载的Volume卷，可以在不同的容器间共享。
- Kubernetes环境推荐使用PVC和PV方式挂载数据卷（Volume）。
- 虽然单Pod可以挂载多个数据卷（Volume），但是并不建议给一个Pod挂载过多数据卷。

PV和PVC

并非所有的Kubernetes数据卷（Volume）具有持久化特征，为了实现持久化的实现，容器存储需依赖于一个远程存储服务。为此Kubernetes引入了PV和PVC两个资源对象，将存储实现的细节从其如何被使用中抽象出来，并解耦存储使用者和系统管理员的职责。PV和PVC的概念如下：

PV

PV是PersistentVolume的缩写，译为持久化存储卷。PV在Kubernetes中代表一个具体存储类型的卷，其对象中定义了具体存储类型和卷参数。即目标存储服务所有相关的信息都保存在PV中，Kubernetes引用PV中的存储信息执行挂载操作。

> **说明** PV是一个集群级别的概念，其对象作用范围是整个Kubernetes集群，而不是一个节点。PV可以有自己的独立生命周期，不依附于Pod。

PVC

PVC是PersistentVolumeClaim的缩写，译为存储声明。PVC是在Kubernetes中一种抽象的存储卷类型，代表了某个具体类型存储的数据卷表达。其设计意图是分离存储与应用编排，将存储细节抽象出来并实现存储的编排。这样Kubernetes中存储卷对象独立于应用编排而单独存在，在编排层面使应用和存储解耦。

PV和PVC使用说明

- **PVC和PV的绑定**
  PVC与PV是一一对应关系，不能一个PVC挂载多个PV，也不能一个PV挂载多个PVC。为应用配置存储时，需要声明一个存储需求声明（PVC），而Kubernetes会通过最佳匹配的方式选择一个满足PVC需求的PV，并与之绑定。所以从职责上PVC是应用所需要的存储对象，属于应用作用域。PV是存储平面的存储对象，属于整个存储域。
  PVC只有绑定了PV之后才能被Pod使用，而PVC绑定PV的过程即是消费PV的过程，这个过程是有一定规则的，以下规则都满足的PV才能被PVC绑定：
  - VolumeMode：被消费PV的VolumeMode需要和PVC一致。
  - AccessMode：被消费PV的AccessMode需要和PVC一致。
  - StorageClassName：如果PVC定义了此参数，PV必须有相关的参数定义才能进行绑定。
  - LabelSelector：通过标签（labels）匹配的方式从PV列表中选择合适的PV绑定。
  - Size：被消费PV的capacity必须大于或者等于PVC的存储容量需求才能被绑定。
- **PV和PVC定义中的size字段**
  PVC和PV里面的size字段作用如下：
  - PVC、PV绑定时，会根据各自的size进行筛选。
  - 通过PVC、StorageClass动态创建PV时，有些存储类型会参考PVC的size创建相应大小的PV和后端存储。
  - 对于支持Resize操作的存储类型，PVC的size作为扩容后PV、后端存储的容量值。

  一个PVC、PV的size值只是在执行一些PVC和PV管控操作的时候，作为配置参数来使用。
  真正的存储卷数据流写数据的时候，不会参考PVC和PV的size字段，而是依赖底层存储介质的实际容量。

数据卷（Volume）使用方式

常见的容器存储数据卷使用方式如下：

- **静态存储卷**
  静态存储卷一般是指由管理员创建的PV。所有的数据卷（Volume）都支持创建静态存储卷。

静态存储卷先由集群管理员分析集群中存储需求，并预先分配一些存储介质（例如云盘、NAS盘等），同时创建对应的PV对象。创建好的PV对象等待PVC来消费。如果负载中定义了PVC需求，Kubernetes会通过相关规则实现PVC和匹配的PV进行绑定，这样就实现了应用对存储服务的访问能力。

- **动态存储卷**
  动态存储卷一般是指通过存储插件自动创建的PV。
  动态卷一般由集群管理员配置好后端的存储池，并创建相应的模板（StorageClass），当有PVC需要消费PV的时候，根据PVC定义的需求，并参考StorageClass的存储细节，由存储插件动态创建一个PV。StorageClass的定义如下：
  - StorageClass
    当您声明一个PVC时，如果在PVC中添加了StorageClassName字段，意味着当PVC在集群中找不到匹配的PV时，会根据StorageClassName的定义触发相应的Provisioner插件创建合适的PV供绑定，即创建动态数据卷。
    动态数据卷由Provisioner插件创建，并通过StorageClassName与PVC进行关联。
    StorageClass可译为存储类，表示为一个创建PV存储卷的模板。在PVC触发自动创建PV的过程中，即使用StorageClass对象中的内容进行创建。其内容如下：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-disk-topology
parameters:
 type: cloud_ssd
provisioner: diskplugin.csi.alibabacloud.com
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

参数	描述
reclaimPolicy	用来指定创建PV的persistentVolumeReclaimPolicy字段值，支持*Delete*和*Retain*。 ■ *Delete*表示动态创建的PV，在销毁的时候也会自动销毁。 ■ *Retain*表示动态创建的PV，不会自动销毁，而是由管理员处理。
allowVolumeExpansion	定义由此存储类创建的PV是否运行动态扩容，默认为 `false` 。是否能动态扩容是由底层存储插件来实现的，这里只是一个开关。
volumeBindingMode	表示动态创建PV的时间，支持*Immediate*和*WaitForFirstConsumer*，分别表示立即创建和延迟创建。

- **延迟绑定**
  针对某类存储（例如阿里云云盘）在挂载属性上有所限制，只能挂载相同可用区的数据卷和节点，不在同一个可用区不可以挂载。这种类型的存储卷通常遇到如下问题：
  - 创建了A可用区的数据卷，但是A可用区的节点资源已经耗光，导致Pod启动无法完成挂载。
  - 集群管理员在规划PVC、PV的时候不能确定在哪些可用区创建多个PV来备用。

  StorageClass中的volumeBindingMode字段正是用来解决此问题，如果将volumeBindingMode配置为*WaitForFirstConsumer*值，则表示存储插件在收到PVC Pending的时候不会立即进行数据卷创建，而是等待这个PVC被Pod消费的时候才执行创建流程。

  > **说明**
  > 延迟绑定实现原理如下：
  > Provisioner在收到PVC Pending状态的时候不会立即进行数据卷创建，而是等待这个PVC被Pod消费。如果有Pod消费此PVC，调度器发现PVC是延迟绑定模式，则继续完成调度功能，且调度器会将调度结果通过patch命令到PVC的metadata中。当Provisioner发现PVC中写入了调度信息时，会根据调度信息获取创建目标数据卷的位置信息（区域和节点信息），并触发PV的创建流程。
  > 在多可用区集群环境中，更推荐使用延迟绑定的动态卷方案，目前阿里云ACK集群已经支持上述配置方案。

动态存储卷的优势

- 动态存储卷让Kubernetes实现了PV的自动化生命周期管理，PV的创建、删除都通过Provisioner完成。
- 自动化创建PV对象，减少了配置复杂度和系统管理员的工作量。
- 动态卷可以实现PVC对存储的需求容量和Provisioner出来的PV容量一致，实现存储容量规划最优。

- **挂载SubPath卷**
  Kubernetes提供了 `volumeMounts.subPath` 属性配置，可用于指定所引用的卷内的子路径，而不是其根路径。

  > **说明** SubPath在Kubernetes的一些版本中存在挂载、卸载不充分的问题，故不建议使用。如果您想要挂载卷的子目录（如NAS、OSS子目录），可以通过阿里云存储CSI原生支持的能力进行挂载。

存储卷配额说明

- 部分存储类型在每个节点的挂载数量是有限制的，例如：阿里云云盘类型在一个节点上最多挂载16块数据盘。对于这样的配额限制，ACK提供了相应的调度功能，当某个节点上挂载了某种存储满额的卷后，调度器就不会再把这个类型的卷调度到这个节点。
- ACK CSI存储插件中可以通过MAX_VOLUMES_PERNODE字段配置每个节点的数据卷挂载数量。关于存储CSI的更多信息，请参见存储CSI概述。

相关文档

- 关于Kubernetes存储的更多信息，请参见Storage。
- 关于存储CSI的更多信息，请参见存储CSI概述。

10.3. 安装与升级CSI组件

CSI组件包括CSI-Plugin和CSI-Provisioner两部分。本文介绍在容器服务Kubernetes版Container Service for Kubernetes（ACK）中，如何管理与升级CSI-Plugin和CSI-Provisioner存储组件。

前提条件

- 您已创建一个大于1.14版本的ACK集群，且存储插件选择为CSI。具体操作，请参见创建Kubernetes托管版集群。
- 您已通过kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

安装CSI-Plugin和CSI-Provisioner

在创建ACK托管版和专有版集群时除非您选择安装flexvolume组件，否则控制台将默认安装CSI-Plugin和CSI-Provisioner组件。

验证安装

验证CSI-Plugin和CSI-Provisioner组件是否成功部署。

- 执行以下命令，查看CSI-Plugin组件是否成功部署。

```
kubectl get pod -n kube-system | grep csi-plugin
```

- 执行以下命令，查看CSI-Provisioner组件是否成功部署。

```
kubectl get pod -n kube-system | grep csi-provisioner
```

升级CSI-Plugin和CSI-Provisioner

CSI-Plugin和CSI-Provisioner组件支持在控制台升级。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理详情页左侧导航栏，选择**运维管理 > 组件管理**。
5. 单击**存储**页签，在**csi-plugin**及**CSI-Provisioner**组件区域单击**升级**。
6. 在**提示**对话框中确认版本信息后单击**确定**。
   升级成功后，对应组件区域会提示升级成功，且可查看组件当前版本。

若控制台升级失败或前置检查失败，可以按照以下场景操作。

- CSI-Plugin组件前置检查失败。
  - 若您的集群没有使用云盘、NAS或OSS类型数据卷，请手动升级CSI-Plugin组件。具体操作，请参见CSI-Plugin升级。
  - 若您的集群使用了云盘、NAS或OSS类型数据卷，且集群为测试环境，需要选择手动升级。具体操作，请参见CSI-Plugin升级。
  - 若您的集群使用了云盘、NAS或OSS类型数据卷，且包含关键业务数据，请联系我们申请手动升级保障。
- CSI-Plugin组件前置检查通过，但是升级失败。
  检查集群中节点是否都是Ready状态，如果存储在NotReady节点，需要先解决节点问题。
  若无法找到具体原因，请联系我们申请手动升级保障。
- 控制台可以看到CSI-Plugin组件，但看不到CSI-Provisioner组件。
  因为使用了早期的CSI-Provisioner的StatefulSet部署模式，请联系我们申请手动升级保障。
- CSI-Provisioner组件前置检查失败。
  - 若您的集群没有使用通过StorageClass创建的云盘或NAS类型的动态数据卷，请手动升级CSI-Provisioner组件。具体操作，请参见CSI-Provisioner升级。
  - 若您的集群使用通过StorageClass创建的云盘或NAS类型的动态数据卷，且集群为测试环境，需要选择手动升级。具体操作，请参见CSI-Provisioner升级。
  - 若您的集群使用通过StorageClass创建的云盘或NAS类型的动态数据卷，且包含关键业务数据，请联系我们申请手动升级保障。
- CSI-Provisioner组件前置检查通过，但是升级失败，请联系我们申请手动升级保障。

联系我们

若您需要申请手动升级保障，请您使用钉钉搜索钉钉群号35532895加入钉钉群咨询。

10.4. 云盘存储卷

10.4.1. 云盘存储卷概述

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，包括静态存储卷和动态存储卷。本文介绍了阿里云云盘存储卷的功能介绍、存储规格、适用场景、使用限制及计费说明。

功能介绍

云盘是阿里云为云服务器ECS提供的数据块级别的块存储产品，具有低时延、高性能、持久性、高可靠等特点。云盘采用分布式三副本机制，为ECS实例提供数据可靠性保证。云盘支持在可用区内自动复制您的数据，防止意外硬件故障导致的数据不可用，保护您的业务免于组件故障的威胁。

根据性能分类，云盘包含以下几类产品：

- ESSD云盘：基于新一代分布式块存储架构的超高性能云盘产品，结合25GE网络和RDMA技术，单盘可提供高达100万的随机读写能力和更低的单路时延能力。更多详情，请参见ESSD云盘。
  建议在大型OLTP数据库、NoSQL数据库和ELK分布式日志等场景中使用。
- SSD云盘：具备稳定的高随机读写性能、高可靠性的高性能云盘产品。
  建议在I/O密集型应用、中小型关系数据库和NoSQL数据库等场景中使用。
- 高效云盘：具备高性价比、中等随机读写性能、高可靠性的云盘产品。
  建议在开发与测试业务和系统盘等场景中使用。
- 普通云盘：属于上一代云盘产品，已经逐步停止售卖。

存储规格

各类型云盘的性能比较如下表所示。

性能类别	ESSD云盘				SSD云盘	高效云盘	普通云盘
	PL3	PL2	PL1	PL0			
单盘容量范围（GiB）	1261~32768	461~32768	20~32768	40~32768	20~32768	20~32768	5~2000
最大IOPS	1000000	100000	50000	10000	25000	5000	数百
最大吞吐量（MB/s）	4000	750	350	180	300	140	30~40
单盘IOPS性能计算公式	min{1800+50*容量, 1000000}	min{1800+50*容量, 100000}	min{1800+50*容量, 50000}	min{1800+12*容量, 10000 }	min{1800+30*容量, 25000}	min{1800+8*容量, 5000}	无
单盘吞吐量性能计算公式（MB/s）	min{120+0.5*容量, 4000}	min{120+0.5*容量, 750}	min{120+0.5*容量, 350}	min{100+0.25*容量, 180}	min{120+0.5*容量, 300}	min{100+0.15*容量, 140}	无
单路随机写平均时延（ms），Block Size=4K	0.2			0.3~0.5	0.5~2	1~3	5~10
API参数取值	cloud_essd				cloud_ssd	cloud_efficiency	cloud

云盘更多性能介绍请参见块存储性能。

适用场景

根据业务需求，您可以对云盘做以下操作：

业务需求	参考链接
存储应用数据	具体操作，请参见： • 使用云盘静态存储卷 • 使用云盘动态存储卷
系统盘或数据盘容量不足	更多信息，请参见扩容概述。 具体操作，请参见： • 在线扩容云盘数据卷 • 手动扩容云盘数据卷 • 自动扩容云盘数据卷（公测）
备份云盘数据	具体操作，请参见使用云盘存储快照。
加密存储在云盘上的数据	更多信息，请参见加密概述。 具体操作，请参见加密云盘存储卷。

关于ACK使用云盘快速入门，请参见以下视频：

注意事项

-
- 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，且不能保证挂载、卸载的优先顺序。使用Deployment时由于升级策略，可能出现重启Pod时新的Pod一直无法挂载，故不推荐使用Deployment。
-
-
- 云盘类型和ECS类型需要匹配才可以挂载，否则会挂载失败。关于云盘类型和ECS类型的匹配关系，请参考实例规格族。
- 每个节点最多可挂载16块云盘，单块云盘容量最大32 TiB。

计费说明

- 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。
- 云盘的具体价格信息，请参见详细定价。

更多信息，请参见计费。

存储类（StorageClass）

如果PVC请求多，就需要创建很多PV，对于运维人员来说维护成本很高，Kubernetes提供一种自动创建PV的机制，叫StorageClass。

StorageClass

容器服务Kubernetes版（ACK）集群默认提供了以下几种StorageClass：

- 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SSD云盘；如果SSD云盘售尽，则创建高效云盘。

> 注意
> - 对于alicloud-csi-provisioner v1.14.8.39-0d749258-aliyun之前的版本，优先创建ESSD云盘。
> - 如果ESSD云盘售尽，则创建SSD云盘。
> - 如果SSD云盘售尽，则创建高效云盘。

- alicloud-disk-topology：使用延迟绑定的方式创建云盘。

以上5种StorageClass中，前4种可以在单可用区使用，最后一种更适合在多可用区使用。

通过StorageClass创建云盘，可用区选择规则如下：

- StorageClass配置volumeBindingMode: WaitForFirstConsumer，则取PVC的Pod所在可用区为创建云盘的可用区。
- StorageClass配置volumeBindingMode: Immediate，且配置zoneId参数（一个可用区），则选择此可用区为创建云盘的可用区。
- StorageClass配置volumeBindingMode: Immediate，且配置zoneId参数（多个可用区），则每次选择轮询配置的多个可用区中的一个作为创建云盘的可用区。
- StorageClass配置volumeBindingMode: Immediate，且没有配置zoneId参数，则选择csi-provisioner所在的可用区作为创建云盘的可用区。

综上，如果您是多可用区集群，推荐您使用WaitForFirstConsumer模式的StorageClass。您可以根据需要的云盘类型自行创建StorageClass。

Default StorageClass

Kubernetes提供Default StorageClass机制，您在PVC不指定StorageClass的情况下，可以通过Default StorageClass创建数据卷，请参见Default StorageClass。

> 说明
> - 由于Default StorageClass会对所有PVC起作用，对于具备不同类型存储卷能力的集群，需要小心使用。例如，您想生成一个NAS类型PVC、PV，并绑定PVC和PV，但可能因为有Default StorageClass而自动创建了云盘PV。基于上述原因，ACK集群没有提供Default StorageClass，如果您期望使用Default StorageClass，可以参考以下配置。
> - 一个集群中最多配置一个Default StorageClass，否则默认能力将不起作用。

1. 配置Default StorageClass。

   执行以下命令将StorageClass （alicloud-disk-ssd）配置为一个Default StorageClass。

   ```
   kubectl patch storageclass alicloud-disk-ssd -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
   ```

   这时查询集群中的StorageClass可以看到alicloud-disk-ssd名字后面加了(default)字样。

   ```
   kubectl get sc
   ```

   返回结果如下：

   ```
   NAME                 PROVISIONER                  AGE
   alicloud-disk-ssd (default)  diskplugin.csi.alibabacloud.com  96m
   ```

2. 使用Default StorageClass。

i. 使用以下模板创建一个没有配置StorageClass的PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: disk-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 20Gi
```

集群会自动创建一个云盘卷（PV），且配置了Default StorageClass（alicloud-disk-ssd）。

```
kubectl get pvc
```

返回结果如下：

```
NAME     STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS     AGE
disk-pvc  Bound   d-bp18pbai447qverm3ttq  20Gi     RWO       alicloud-disk-ssd  49s
```

您可以通过以下命令取消默认存储类型配置。

```
kubectl patch storageclass alicloud-disk-ssd -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

10.4.2. 使用云盘静态存储卷

云盘是阿里云为云服务器ECS提供的数据块级别的块存储产品，具有低时延、高性能、持久性、高可靠等特点。ACK支持使用CSI插件创建云盘静态存储卷和动态存储卷。本文介绍如何在CSI插件中挂载云盘静态存储卷及使用云盘静态存储卷如何实现持久化存储。

前提条件

- 已创建Kubernetes集群，并且在该集群中安装CSI插件。具体操作，请参见创建Kubernetes托管版集群。
- 已创建按量付费的云盘并记录云盘ID为 `d-wz92s6d95go6ki9x****` 。具体操作，请参见创建云盘。

> **说明**

- 步骤二：选择集群凭证类型。

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，不会因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。

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。

通过控制台的方式使用云盘静态存储卷

步骤一：创建PV。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。
5. 在**存储卷**页面，单击右上角的**创建**。
6. 在**创建存储卷**对话框中配置参数。

参数	描述
存储卷类型	支持云盘、NAS、OSS三种云存储类型。本文中选择为云盘。
存储驱动	支持Flexvolume和CSI。本文中选择为CSI。
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。

7. 参数配置完成后，单击**创建**。

步骤二：创建PVC。

1. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
2. 在**存储声明**页面，单击右上角的**创建**。
3. 在**创建存储声明**对话框中，配置参数。

参数	描述
存储声明类型	支持云盘、NAS、OSS三种云存储类型。本文中选择云盘。
名称	创建的数据卷的名称，数据卷名在命名空间内必须唯一。
分配模式	本文中选择已有存储卷。 **说明** 若未创建存储卷，您可以设置**分配模式**为**创建存储卷**，配置创建存储卷参数。具体操作，请参见创建PV。
已有存储卷	单击**已有存储卷**，在目标存储卷右侧操作列单击**选择**，选择存储卷。
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云盘容量。
访问模式	默认为ReadWriteOnce。

4. 单击**创建**。

   创建成功后可以在列表中看到存储声明，并且已绑定相应的存储卷。

步骤三：创建应用

1. 在集群管理页左侧导航栏中，选择**工作负载 > 有状态**。
2. 在**有状态**页面中，单击**使用镜像创建**。
3. 配置创建应用的参数信息。

   以下主要为您介绍数据卷的配置。关于其他参数的描述，请参见创建有状态工作负载StatefulSet。
   ACK数据卷支持配置本地存储和云存储，本示例需要配置**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。

通过kubectl命令行的方式使用云盘静态存储卷

步骤一：创建PV

1. 使用以下内容，创建*pv-static.yaml*文件。

```
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: "<your-disk-id>"
 nodeAffinity:
  required:
   nodeSelectorTerms:
   - matchExpressions:
     - key: topology.diskplugin.csi.alibabacloud.com/zone
       operator: In
       values:
       - "<your-node-zone-id>"
```

参数	说明
name	PV的名称。
labels	设置PV的标签。
storage	云盘的可使用量。
accessModes	设置访问模式。
persistentVolumeReclaimPolicy	PV的回收策略。
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和PVC所属的区域信息。 通过定义该参数，可以将PV和PVC所在的Pod调度到对应的区域上。

2. 执行以下命令，创建PV。

```
kubectl create -f pv-static.yaml
```

3. 查看创建的PV。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏选择**存储 > 存储卷**。

   在**存储卷**页面可查看创建的PV。

步骤二：创建PVC

1. 使用以下内容，创建*pvc-static.yaml*文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: csi-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 25Gi
 selector:
  matchLabels:
   alicloud-pvname: static-disk-pv
```

参数	说明
name	PVC的名称。
accessModes	设置访问模式。

参数	说明
storage	声明应用使用量，不能大于存储卷的总量。
matchLabels	通过标签关联PV，与PV标签保存一致。

2. 执行以下命令，创建PVC。

```
kubectl create -f pvc-static.yaml
```

3. 在集群管理页左侧导航栏选择**存储 > 存储声明**。

   在**存储声明**页面可看到创建的PVC。

步骤三：创建应用。

本文以Web应用为例，在应用中挂载PVC。

1. 使用以下内容，创建*web.yaml*文件。

```
apiVersion: apps/v1
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
```

   - mountPath：云盘在容器中挂载的位置。
   - claimName：PVC的名称，用于绑定PVC。

2. 执行以下命令，创建一个挂载了云盘静态存储卷的应用并挂载PVC。

```
kubectl apply -f web.yaml
```

3. 在集群管理页左侧导航栏选择**工作负载 > 有状态**。

   在**有状态**页面可看到创建的Web应用。

验证静态云盘的持久化存储

1. 查看部署Web应用的Pod和云盘文件。
   i. 执行以下命令，查看部署的Web应用所在Pod的名称。

   ```
   kubectl get pod | grep web
   ```

   预期输出：

   ```
   web-1**** 1/1 Running 0 32s
   ```

   ii. 执行以下命令，查看*/data*路径下是否挂载了新的云盘。

   ```
   kubectl exec web-1**** df | grep data
   ```

   预期输出：

   ```
   /dev/vdf 20511312 45080 20449848 1% /data
   ```

   iii. 执行以下命令，查看*/data*路径下的文件。

   ```
   kubectl exec web-1**** ls /data
   ```

   预期输出：

   ```
   lost+found
   ```

2. 执行以下命令，在*/data*路径下创建文件*static*。

   ```
   kubectl exec web-1**** touch /data/static
   ```

3. 执行以下命令，查看*/data*路径下的文件。

   ```
   kubectl exec web-1**** ls /data
   ```

   预期输出：

   ```
   lost+found
   static
   ```

4. 执行以下命令，删除名称为 `web-1****` 的Pod。

   ```
   kubectl delete pod web-1****
   ```

   预期输出：

   ```
   pod "web-1****" deleted
   ```

5. 验证删除Pod后，云盘里创建的文件是否还存在。
   i. 执行以下命令，查看重建的Pod名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME    READY  STATUS   RESTARTS  AGE
   web-2**** 1/1    Running  0       14s
   ```

ii. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec web-2**** ls /data
```

预期输出：

```
lost+found
static
```

*static*文件仍然存在，则说明静态云盘的数据可持久化保存。

10.4.3. 使用云盘动态存储卷

云盘是阿里云为云服务器ECS提供的数据块级别的块存储产品，具有低时延、高性能、持久性、高可靠等特点。ACK支持使用CSI插件创建云盘动态存储卷。本文介绍如何使用云盘动态存储卷，及如何验证云盘动态存储卷的持久化存储特性。

前提条件

- 已创建Kubernetes集群，并且在该集群中安装CSI插件。具体操作，请参见创建Kubernetes托管版集群。
- 已通过Kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

动态云盘的使用场景：没有提前购买云盘，在应用部署时自动购买云盘的情况。

动态云盘的使用方式：

- 手动创建PVC，在PVC中声明StorageClass。
- 部署应用时通过StorageClass自动创建PV。

有关StorageClass的详细说明，请参见存储类（StorageClass）。

通过控制台的方式使用云盘动态存储卷

步骤一：创建StorageClass

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储类**。
5. 在**存储类**页面，单击右上角的**创建**。
6. 在**创建**对话框，配置StorageClass的相关参数。

   部分参数的说明如下所示：

参数	说明
名称	StorageClass的名称。 名称必须以小写字母开头，只能包含小写字母、数字、小数点（.）和短划线（-）。
存储卷类型	可选择**云盘**或**NAS**。本示例选择**云盘**。
存储驱动	默认为**CSI**。

参数	说明
参数	默认参数type，其值为cloud_essd。表示云盘类型，支持cloud_efficiency、cloud_ssd、cloud_essd、available四种参数及除available外其他三种参数的任意组合。例如，`type: cloud_efficiency, cloud_ssd, cloud_essd`，这种配置方式支持对指定类型云盘的依次创建，直到创建成功。其中available会对SSD、高效依次尝试创建，直到创建成功。 **说明** 部分ECS机型不支持ESSD云盘挂载。更多信息，请参见块存储FAQ。 可添加自定义参数。例如，配置zoneId，表示自动创建云盘所在的区域。单可用区集群，与集群所在区域相同。例如，`cn-beiiing-a`：多可用区集群，zoneId可同时配置多个。例如，`cn-beijing-a, cn-beijing-b`
回收策略	云盘的回收策略，默认为Delete，支持Retain。 ○ Delete模式：删除PVC的时候，PV和云盘会一起删除。 ○ Retain模式：删除PVC的时候，PV和云盘数据不会被删除，需要您手动删除。 如果数据安全性要求高，推荐使用Retain方式以免误删数据。
绑定模式	云盘的绑定模式。默认为Immediate，支持WaitForFirstConsumer。 ○ Immediate：表示先创建云盘再创建Pod。 ○ WaitForFirstConsumer：延迟绑定，即调度器先调度Pod，并根据Pod的可用区信息创建云盘。

7. 参数配置完成后，单击**创建**。
   创建成功后在**存储类**列表中可看到刚创建的StorageClass。

步骤二：创建PVC

1. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
2. 在**存储声明**页面，单击右上角的**创建**。
3. 在**创建存储声明**对话框中，配置参数。

参数	描述
存储声明类型	支持云盘、NAS、OSS三种云存储类型。本文中选择云盘。
名称	创建的存储声明名称在命名空间内必须唯一。
分配模式	本文中选择**使用存储类动态创建**。
已有存储类	单击**选择存储类**，在**选择存储类**对话框目标存储类右侧**操作**列单击**选择**。
总量	所创建存储卷的容量。
访问模式	默认为ReadWriteOnce，也可选择ReadOnlyMany或ReadWriteMany。

4. 单击**创建**。
   创建成功后在**存储声明**列表中可看到创建的存储声明，并且已绑定相应的存储卷。

步骤三：创建应用

1. 在集群管理页左侧导航栏中，选择**工作负载 > 有状态**。

2. 在**有状态**页面中，单击**使用镜像创建**。
3. 配置创建应用的参数信息。

   以下主要为您介绍数据卷的配置。关于其他参数的描述，请参见创建有状态工作负载StatefulSet。
   ACK数据卷支持配置本地存储和云存储，本示例需要配置**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。

通过kubectl命令行的方式使用云盘动态存储卷

步骤一：创建StorageClass

在多可用区集群场景下，您可以根据不同的场景通过以下两种方式创建StorageClass。

方式一：使用Topology（延迟绑定）方式创建StorageClass

延迟绑定可以优化ECS和云盘不在一个可用区的问题。以下通过部署名为*storage-class-csi-topology.yaml*文件为例，创建StorageClass。

1. 使用以下内容，创建*storage-class-topology.yaml*文件。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-disk-topology-essd
provisioner: diskplugin.csi.alibabacloud.com
parameters:
  type: cloud_essd
  resourceGroupId: "default"
  regionId: cn-beijing
  zoneId: cn-beijing-a
  fstype: ext4
  readonly: "true"
  mkfsOptions: "-O project,quota"
  diskTags: "a:b,b:c"
  encrypted: "false"
  performanceLevel: PL1
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain
allowVolumeExpansion: true
```

参数	说明
name	StorageClass的名称。

参数	说明
provisioner	配置为 `diskplugin.csi.alibabacloud.com` 。表示使用阿里云云盘Provisioner插件创建StorageClass。
type	表示云盘类型，支持cloud_efficiency、cloud_ssd、cloud essd、available四种参数及除available外其他三种参数的任意组合。例如， `type: cloud_efficiency, cloud_ssd, cloud_essd` ，这种配置方式支持对指定类型云盘的依次创建，直到创建成功。其中available会对SSD、高效依次尝试创建，直到创建成功。 **说明** 部分ECS机型不支持ESSD云盘挂载。更多信息，请参见块存储FAQ。
resourceGroupId	可选，定义云盘的资源组。默认为 `default` 。
（可选）regionId	可选，自动创建云盘所在的地域，与集群的地域相同。
（可选）zoneId	可选，自动创建云盘所在的区域。 ○ 单可用区集群，与集群所在区域相同。 ○ 多可用区集群，zoneId可同时配置多个，例如： `zoneId: cn-hangzhou-a,cn-hangzhou-b,cn-hangzhou-c`
（可选）fstype	可选，自动创建云盘所使用的文件系统，默认为*ext4*。
（可选）readonly	可选，挂载自动创建云盘的权限是否为可读。 ○ *true*：云盘具有只读权限。 ○ *false*：云盘具有可读可写权限。 默认为*false*。
（可选）mkfsOptions	可选，云盘格式化所用的参数。例如， `mkfsOptions: "-O project,quota"` 。
（可选）diskTags	可选，自定义云盘Tag。例如， `diskTags: "a:b,b:c"` 。
（可选）encrypted	可选，表示创建的云盘是否加密。默认为false，创建的云盘不加密。
（可选）performanceLevel	可选，值为*PL0*PL1、*PL2*或PL3。更多信息，请参见容量范围与性能级别的关系。
reclaimPolicy	云盘的回收策略，默认为Delete，支持Retain。 ○ Delete模式：删除PVC的时候，PV和云盘会一起删除。 ○ Retain模式：删除PVC的时候，PV和云盘数据不会被删除，需要您手动删除。 如果数据安全性要求高，推荐使用Retain方式以免误删数据。
volumeBindingMode	云盘的绑定模式。默认为Immediate，支持WaitForFirstConsumer。 ○ Immediate：表示先创建云盘再创建Pod。 ○ WaitForFirstConsumer：延迟绑定，即调度器先调度Pod，并根据Pod的可用区信息创建云盘。
allowVolumeExpansion	配置为true时，可以实现云盘的自动扩容。

2. 执行以下命令创建StorageClass。

```
kubectl apply -f storage-class-topology.yaml
```

3. 查看创建的StorageClass。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**存储 > 存储类**。
   在**存储类**页面查看创建的StorageClass。

方式二：先创建云盘再创建Pod的方式创建StorageClass

1. 使用以下内容，创建*storage-class-csi.yaml*文件。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-disk-ssd-b
provisioner: diskplugin.csi.alibabacloud.com
parameters:
 type: cloud_ssd
 regionId: cn-beijing
 zoneId: cn-beijing-b
 encrypted: false
reclaimPolicy: Retain
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

参数	说明
provisioner	配置为 `diskplugin.csi.alibabacloud.com` 。表示使用阿里云云盘Provisioner插件创建StorageClass。
type	表示云盘类型，支持cloud_efficiency、cloud_ssd、cloud essd、available四种参数及除available外其他三种参数的任意组合。例如， `type: cloud_efficiency, cloud_ssd, cloud_essd` ，这种配置方式支持对指定类型云盘的依次创建，直到创建成功。其中available会对SSD、高效依次尝试创建，直到创建成功。 **说明** 部分ECS机型不支持ESSD云盘挂载。更多信息，请参见块存储FAQ。
（可选）regionId	可选，期望创建云盘的区域。
（可选）zoneId	可选，期望创建云盘的可用区。
（可选）encrypted	可选，标识创建的云盘是否加密。默认情况是false，创建的云盘不加密。
reclaimPolicy	云盘的回收策略，默认为Delete，支持Retain。 ○ Delete模式：删除PVC的时候，PV和云盘会一起删除。 ○ Retain模式：删除PVC的时候，PV和云盘数据不会被删除，需要您手动删除。 如果数据安全性要求高，推荐使用Retain方式以免误删数据。
allowVolumeExpansion	配置为true时，可以实现云盘的自动扩容。

参数	说明
volumeBindingMode	云盘的绑定模式。默认为Immediate，支持WaitForFirstConsumer。 ○ Immediate：表示先创建云盘再创建Pod。 ○ WaitForFirstConsumer：延迟绑定，即调度器先调度Pod，并根据Pod的可用区信息创建云盘。

2. 执行以下命令，创建StorageClass。

```
kubectl apply -f storage-class-csi.yaml
```

3. 查看创建的StorageClass。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**存储 > 存储类**。
   在**存储类**页面查看创建的StorageClass。

步骤二：创建PVC

1. 使用以下内容，创建*pvc-ssd.yaml*文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: disk-pvc
spec:
 accessModes:
 - ReadWriteOnce
 volumeMode: Filesystem
 resources:
  requests:
   storage: 25Gi
 storageClassName: alicloud-disk-ssd
```

参数	说明
name	PVC的名称。
accessModes	配置访问模式。
（可选）volumeMode	可选，挂载云盘的格式，为Filesystem或Block。 默认为文件系统挂载。
storageClassName	StorageClass的名称，用于绑定StorageClass。
storage	申请的云盘大小，最小为20 GiB。

2. 执行以下命令，创建PVC。

```
kubectl create -f pvc-ssd.yaml
```

3. 查看创建的PVC。

在集群管理页左侧导航栏选择**存储 > 存储声明**。可以在**存储声明**页面可以看到创建的PVC。

步骤三：创建应用

1. 创建*pvc-dynamic.yaml*文件。

   创建一个名为**nginx-dynamic**的应用，并挂载PVC。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: nginx-dynamic
spec:
 selector:
  matchLabels:
   app: 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-ssd
```

参数	说明
mountPath	云盘挂载的位置。
claimName	PVC的名称，用于绑定PVC。

2. 执行以下命令，创建应用并挂载PVC。

```
kubectl create -f pvc-dynamic.yaml
```

3. 查看创建的应用。

   在集群管理页左侧导航栏选择**工作负载 > 有状态**。您可以在**有状态**页面看到创建的应用。

验证动态云盘的持久化存储

云盘提供了持久化存储服务，当某个Pod删除时，重新部署的Pod将自动同步之前Pod的所有数据。

根据以下示例验证动态云盘的持久化存储特性：

1. 查看**nginx-dynamic**应用所在的Pod和云盘文件。

i. 执行以下命令，查看**nginx-dynamic**应用所在Pod的名称。

```
kubectl get pod | grep dynamic
```

预期输出：

```
nginx-dynamic-1****  1/1   Running   0      3m
```

ii. 执行以下命令，查看*/data*路径下是否挂载了新的云盘。

```
kubectl exec nginx-dynamic-1**** df | grep data
```

预期输出：

```
/dev/vdh     20511312   45080 20449848   1% /data
```

iii. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec nginx-dynamic-1**** ls /data
```

预期输出：

```
lost+found
```

2. 在云盘里创建文件。

i. 执行以下命令，在*/data*路径下创建文件*dynamic*。

```
kubectl exec nginx-dynamic-1**** touch /data/dynamic
```

ii. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec nginx-dynamic-1**** ls /data
```

预期输出：

```
dynamic
lost+found
```

3. 执行以下命令，删除名为 `nginx-dynamic-1****` 的Pod。

```
kubectl delete pod nginx-dynamic-1****
```

预期输出：

```
pod "nginx-dynamic-1****" deleted
```

4. 验证删除Pod后，云盘中创建的文件是否还存在。

i. 执行以下命令，查看重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME              READY  STATUS    RESTARTS  AGE
nginx-dynamic-2****     1/1   Running   0      2m
```

ii. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec nginx-dynamic-2**** ls /data
```

预期输出：

```
dynamic
lost+found
```

*dynamic*文件仍然存在，说明动态云盘的数据可持久保存。

10.4.4. 在线扩容云盘数据卷

在Kubernetes 1.16版中，云盘数据卷在线扩容已经进入Beta阶段，ACK支持在Kubernetes 1.16及以上版本中实现CSI云盘在线扩容。本文介绍应用在线时如何动态扩容云盘，从而增加云盘空间。

背景信息

云盘数据卷扩容包括**云盘扩容**和**文件系统扩容**两部分，云盘和文件系统都支持应用在线扩容（设备挂载到目录状态下扩容），但是从文件系统扩容最佳实践角度，最稳定的扩容方案是先停止应用层服务、解除挂载目录，再进行数据卷扩容。

相关概念

- 在线扩容
  指在业务Pod不暂停的情况下，实现云盘及其文件系统的扩容。
- 离线扩容
  指在业务Pod暂停的情况下，完成云盘数据卷与文件系统扩容后，再启动业务Pod。

从Kubernetes 1.16开始默认实现了Pod不重启即可以实现数据卷的扩容。

使用说明

- **使用限制**
  仅支持在线扩容2000 GiB以下容量的云盘。
- **数据备份**
  扩容数据卷前，先备份云盘快照，以防扩容过程异常导致数据出现问题。
- **适用范围**
  - 只有动态存储卷才可以进行数据卷动态扩容，即配置了StorageClassName的PVC。
  - 不支持扩容InlineVolume类型（非PV、PVC方式）的云盘数据卷。
  - 不支持动态扩容普通云盘类型。
  - StorageClass需要配置为**AllowVolumeExpansion: True**。ACK集群默认提供的StorageClass为True，自建的StorageClass需要自行配置。
- **插件版本**
  升级您使用Flexvolume或者CSI插件到最新版本。

为RAM用户添加ResizeDisk权限

实现云盘在线扩容前，您需要为集群的**RAM角色**添加ResizeDisk权限。针对不同集群和插件类型，添加ResizeDisk权限的步骤如下：

专有集群（CSI插件）

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选中目标集群，并在目标集群右侧**操作**列下，单击**详情**。
4. 在集群管理页左侧导航栏中，单击**集群信息**。

5. 单击**集群资源**页签，单击**Master RAM角色**链接。
6. 在RAM控制台中，添加ResizeDisk权限。添加权限具体步骤，请参见修改自定义策略内容。

```
"Action": [
  "ecs:AttachDisk",
  "ecs:DetachDisk",
  "ecs:DescribeDisks",
  "ecs:CreateDisk",
  "ecs:CreateSnapshot",
  "ecs:DeleteDisk",
  "ecs:ResizeDisk",
  "ecs:CreateNetworkInterface",
  "ecs:DescribeNetworkInterfaces",
  "ecs:AttachNetworkInterface",
  "ecs:DetachNetworkInterface",
  "ecs:DeleteNetworkInterface",
  "ecs:DescribeInstanceAttribute",
  "ecs:AssignPrivateIpAddresses",
  "ecs:UnassignPrivateIpAddresses",
  "ecs:DescribeInstances"
```

专有集群（Flexvolume插件）、托管集群（所有插件类型）

重复上述前4个步骤，然后在集群管理页面单击**Worker RAM 角色**链接。

不重启容器实现在线扩容

1. 使用命令行工具连接Kubernetes集群，请参见步骤二：选择集群凭证类型。

   本文假设应用的当前状态如下。
   执行以下命令获取Pod信息。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   web-0     1/1   Running  0      42s
   ```

   执行以下命令查看Pod的挂载详情。

   ```
   kubectl exec web-0 df /data
   ```

   预期输出：

   ```
   Filesystem    1K-blocks  Used  Available Use% Mounted on
   /dev/vdb      20511312  45080 20449848  1%  /data
   ```

   执行以下命令获取PVC信息。

   ```
   kubectl get pvc
   ```

   预期输出：

   ```
   NAME        STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS        AGE
   disk-ssd-web-0  Bound   d-wz9hpoifm43yn9zi****  20Gi     RWO       alicloud-disk-available  57s
   ```

   执行以下命令获取PV信息。

   ```
   kubectl get pv
   ```

   预期输出：

   ```
   NAME            CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM           STORAGECLASS
   REASON  AGE
   d-wz9hpoifm43yn9zi****  20Gi     RWO       Delete      Bound     default/disk-ssd-web-0  alicloud-disk-avail
   able       65s
   ```

2. 在符合使用说明的各个条件下，执行以下命令进行数据卷扩容。

   ```
   kubectl patch pvc disk-ssd-web-0 -p '{"spec":{"resources":{"requests":{"storage":"30Gi"}}}}'
   ```

   等待一定时间（一分钟以内）后扩容完成，检查状态如下。

执行以下命令查看PV信息。

```
kubectl get pv d-wz9hpoifm43yn9zi****
```

预期输出：

```
NAME                CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM                STORAGECLASS        RE
ASON  AGE
d-wz9hpoifm43yn9zi****  30Gi    RWO       Delete      Bound   default/disk-ssd-web-0  alicloud-disk-availabl
e       5m23s
```

执行以下命令查看PVC信息。

```
kubectl get pvc
```

预期输出：

```
NAME        STATUS  VOLUME              CAPACITY  ACCESS MODES  STORAGECLASS        AGE
disk-ssd-web-0  Bound   d-wz9hpoifm43yn9zi****  30Gi    RWO       alicloud-disk-available  5m10s
```

执行以下命令查看Pod的挂载详情。

```
kubectl exec web-0 df /data
```

预期输出：

```
Filesystem   1K-blocks  Used  Available  Use% Mounted on
/dev/vdb     30832548  45036 30771128  1%   /data
```

您只需要执行一条命令即可完成所有的扩容操作。

10.4.5. 手动扩容云盘数据卷

在容器服务Kubernetes 1.16之前的旧版本中，目前无法实现云盘自动扩容，需要手动扩容。本文介绍如何手动扩容云盘，从而增加云盘空间。

背景信息

相关概念

- 自动扩容
  指只需要修改PVC的Size，而不需手动进行具体扩容操作，即可完成云盘数据卷及文件系统的扩容。
- 手动扩容
  指需要手动扩容云盘数据卷，且手动执行 `resize2fs` 实现文件系统的扩容。
- 在线扩容
  指在业务Pod不暂停的情况下，实现云盘及其文件系统的扩容。
- 离线扩容
  指在业务Pod暂停的情况下，完成云盘卷、文件系统扩容后，再启动业务Pod。

云盘数据卷扩容包括以下三部分：

- 扩容云盘存储空间：需要在云盘控制台操作。
- 扩容文件系统：需要所挂载的云盘在ECS节点手动操作。
- 更新PV及PVC的Size（暂不支持）。

> **说明**
> 由于以下原因，建议在集群升级到高版本后再通过自动扩容方式修改PV及PVC的大小，低版本不修改。
> - 不同版本K8s集群更新PV、PVC对象Size值的操作不同。
> - PV、PVC的Size值不会影响底层存储空间的使用，即：虽然PV、PVC存储空间大小为20 GiB，对应云盘与文件系统存储空间为30 GiB，但是Pod依然可以使用30 GiB的存储空间。

基于稳定性需求，提供以下两种扩容的方式：

- 不暂停应用手动在线扩容：文件系统在线扩容时，如果IO流量高，文件系统存在出现IO报错的风险。但应用可以不重启。
- 暂停应用手动扩容：应用停止后，磁盘IO会暂停，文件系统扩容数据更安全。但应用会暂停一段时间。

使用说明

- **使用限制**
  仅支持扩容2000 GiB以下容量的云盘。
- **数据备份**
  扩容数据卷前，先执行云盘快照备份数据，以防扩容过程异常导致数据丢失。
- **适用范围**
  不支持自动扩容的场景：
  - K8s集群版本小于1.16。
  - PV类型为静态云盘数据卷。

示例应用

本文以Web有状态应用为例，介绍两种扩容方式。执行以下操作，查看云盘使用信息。

- 执行以下命令查看Web应用的Pod。

  ```
  kubectl get pod | grep web
  ```

  预期输出：

  ```
  NAME            READY  STATUS  RESTARTS  AGE
  web-0           1/1   Running  0     11h
  web-1           1/1   Running  0     11h
  ```

- 执行以下命令查看Web应用的PVC。

  ```
  kubectl get pvc | grep web
  ```

  预期输出：

  ```
  NAME       STATUS  VOLUME    CAPACITY  ACCESS MODES  STORAGECLASS     AGE
  disk-ssd-web-0  Bound  d-0jlhaq***  20Gi    RWO       alicloud-disk-essd  11h
  disk-ssd-web-1  Bound  d-0jl0j5***  20Gi    RWO       alicloud-disk-essd  11h
  ```

- 执行以下命令查看Web应用的PV。

  ```
  kubectl get pv | grep web
  ```

  预期输出：

  ```
  NAME     CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM          STORAGECLASS    REASON  AG
  E
  d-0jl0j5***  20Gi    RWO      Delete      Bound   default/disk-ssd-web-1  alicloud-disk-essd      11h
  d-0jlhaq***  20Gi    RWO      Delete      Bound   default/disk-ssd-web-0  alicloud-disk-essd      11h
  ```

从以上预期输出可得：示例应用Web使用了20 GiB的 d-0jl0j5*** 和 d-0jlhaq*** 云盘，分别挂载在2个Pod上。

关于如何部署Web有状态应用，请参见创建有状态工作负载StatefulSet。

方式一：不暂停应用手动在线扩容

根据PV信息找到对应的云盘，手动对云盘扩容，然后在云盘对应挂载节点手动执行文件系统扩容。以下示例把2个云盘扩容到30 GiB。

步骤一：云盘数据卷扩容

1. 登录ECS管理控制台。
2. 在左侧导航栏，选择**存储与快照 > 云盘**。
3. 找到Web应用对应的云盘*d-0jl0j5****及*d-0jlhaq****，查看云盘状态，并分别在**操作**列单击**更多 > 云盘扩容**。
4. 在**磁盘扩容**页面选中**在线扩容**，并输入扩容后容量，例如30 GiB。

   > **说明** 设置的扩容后容量不允许小于当前容量。

5. 确认费用，阅读并选中《云服务器ECS服务条款》后，单击**确认扩容**。

   具体操作，请参见在线扩容云盘（Linux系统）。

步骤二：文件系统扩容

扩容云盘后需要完成对文件系统的扩容，否则容器内存储空间依然是扩容前的大小。

> **注意** 该步骤是针对于未分区云盘，是K8s中云盘的标准使用方法，不建议在K8s系统中使用分区云盘。
> - 若PV使用未分区云盘，则不能手动执行分区操作，否则将会对当前文件系统产生破坏，造成数据丢失。
> - 若PV使用分区云盘，则需要扩容分区设备后扩容文件系统。具体操作，请参见步骤三：查看云盘分区情况及步骤四：扩容分区。

1. 查看当前云盘挂载节点。
   i. 登录ECS管理控制台。
   ii. 在左侧导航栏，选择**存储与快照 > 云盘**。
   iii. 找到Web应用对应的云盘*d-0jl0j5****及*d-0jlhaq****，分别单击云盘名称。
   iv. 在**云盘详情**页面的**云盘挂载信息**列表下单击**所挂载的实例**。
   v. 在**实例详情**页签查看实例**网络信息**。

   > **说明** 也可通过在K8s中查看该云盘对应Pod所在的节点。具体操作，请参见查看容器组（Pod）。

2. 连接到挂载云盘的ECS实例节点上，获取云盘设备符。

   关于如何连接到ECS实例节点，请参见连接方式概述。
   可通过以下方法获取云盘设备符。

   - 获取云盘设备符。
   - 执行以下命令获取云盘*d-0jlhaq****的设备符。

     ```
     # 查询 {pv-name}
     mount |grep d-0jlhaq***
     ```

     预期输出：

     ```
     /dev/vdc on /var/lib/kubelet/plugins/kubernetes.io/csi/pv/d-0jlhaq***/globalmount type ext4 (rw,relatime)
     /dev/vdc on /var/lib/kubelet/pods/a26d174f-***/volumes/kubernetes.io~csi/d-0jlhaq***/mount type ext4 (r
     w,relatime)
     ```

从以上预期输出可得：所挂载的云盘 d-0jlhaq*** 在当前节点的设备符为*/dev/vdc*。

3. 执行以下命令扩容文件系统。

```
resize2fs /dev/vdc
```

> **说明** */dev/vdc*为步骤2获取的云盘设备符。

预期输出：

```
resize2fs 1.43.5 (04-Aug-2017)
Filesystem at /dev/vdc is mounted on /var/lib/kubelet/plugins/kubernetes.io/csi/pv/d-0jlhaq***/globalmount; on-line resizing required
old_desc_blocks = 3, new_desc_blocks = 4
The filesystem on /dev/vdc is now 7864320 (4k) blocks long.
```

4. 执行以下命令验证文件系统扩容结果。

```
lsblk /dev/vdc
```

预期输出：

```
NAME MAJ:MIN  RM SIZE RO TYPE MOUNTPOINT
vdc  254:32   0  30G  0 disk /var/lib/kubelet/pods/a26d174f-***/volumes/kubernetes.io~csi/d-0jlhaq***/mount
```

从以上预期输出可得，*vdc*文件系统存储空间为30 GiB。

方式二：暂停应用手动扩容

通过删除StatefulSet或者将Replica配置为0的方式暂停应用，手动对每个依赖数据盘扩容，然后再启动应用。通过以下示例把2个云盘扩容到30 GiB。

步骤一：删除应用Pod

1. 执行以下命令将Pod缩容为0个副本。

```
kubectl scale sts web --replicas=0
```

预期输出：

```
statefulset.apps/web scaled
```

2. 执行以下命令查看Pod是否删除。

```
kubectl get pod | grep web
```

无输出，代表已删除Web应用。

步骤二：云盘数据卷扩容

1. 登录ECS管理控制台。
2. 在左侧导航栏，选择**存储与快照 > 云盘**。
3. 找到Web应用对应的云盘*d-0jl0j5****及*d-0jlhaq****，查看云盘状态，并分别在**操作**列选择**更多 > 云盘扩容**。
4. 在**磁盘扩容**页面配置扩容方式及扩容后容量。
   - 若云盘处于**待挂载**状态，在**磁盘扩容**页面无需选中**在线扩容**，输入扩容后容量，例如30 GiB。
   - 若云盘处于**使用中**状态，在**磁盘扩容**页面选中**在线扩容**，并输入扩容后容量。

> **说明** 设置的扩容后容量不允许小于当前容量。

5. 确认费用，阅读并选中《云服务器ECS服务条款》后，单击**确认扩容**。

   具体操作，请参见在线扩容云盘（Linux系统）。

步骤三：文件系统扩容

扩容云盘后需要完成对文件系统的扩容，否则容器内存储空间依然是扩容前的大小。

> **注意** 该步骤是针对于未分区云盘，是K8s中云盘的标准使用方法，不建议在K8s系统中使用分区云盘。
> - 若PV使用未分区云盘，则不能手动执行分区操作，否则将会对当前文件系统产生破坏，造成数据丢失。
> - 若PV使用分区云盘，则需要扩容分区设备后扩容文件系统。具体操作，请参见步骤三：查看云盘分区情况及步骤四：扩容分区。

1. （可选）将待挂载云盘挂载到某个ECS节点。

   > **说明** 需要将云盘挂载到ECS节点上才可实现文件系统扩容。

   i. 登录ECS管理控制台。

   ii. 在左侧导航栏，选择**存储与快照 > 云盘**。

   iii. 对于**待挂载**状态的云盘，在**操作**列单击**更多 > 挂载**。

   iv. 在**挂载云盘**对话框，选择目标实例并设置云盘相关释放行为。

参数	说明
目标实例	选择需要挂载的ECS实例。
云盘随实例释放	选中此选项，释放实例时会自动释放此云盘。 如果未选中，当实例被释放时该云盘会被保留下来。 **说明** 如果您挂载的是从其他实例卸载的系统盘，**云盘随实例释放**中的实例指系统盘被卸载前的源ECS实例，并非当前操作的实例。
自动快照随云盘释放	选中此选项，当云盘释放时该云盘创建的自动快照都会一起释放。建议您不要选择该选项，以便保留备份数据。

   v. 单击**执行挂载**。

   如果该云盘的状态变为**使用中**，表示挂载成功。

2. 连接到挂载云盘的ECS实例节点上，获取云盘设备符。

   关于如何连接到ECS实例节点，请参见连接方式概述。

   - 执行以下命令获取云盘设备符。

```
for device in `ls /sys/block | grep vd`; do
 cat /sys/block/$device/serial | grep 0jlhaq*** && echo $device;
done
```

   > **说明** `0jlhaq***` 为扩容云盘ID `d-0jlhaq***` 中 `d-` 后的字符串。

   - （可选）若上述操作无法获取到云盘设备符，请按以下操作获取。

     a. 将目标云盘卸载后执行 `ls /dev/vd*` 命令查看云盘列表。

     b. 将目标云盘挂载后执行 `ls /dev/vd*` 命令查看云盘列表。

c. 比较上述获取的云盘列表，多出来的云盘即为目标云盘。

3. 执行以下命令扩容文件系统。

```
resize2fs /dev/vdb
```

> **说明** */dev/vdb*为步骤2获取的云盘设备符。

预期输出：

```
resize2fs 1.43.5 (04-Aug-2017)
Resizing the filesystem on /dev/vdb to 7864320 (4k) blocks.
The filesystem on /dev/vdb is now 7864320 (4k) blocks long.
```

4. 验证文件系统扩容结果。

i. 执行以下命令创建临时文件夹*/mnt/disk/*，并将云盘挂载到临时文件夹上。

```
mkdir /mnt/disk
mount /dev/vdb /mnt/disk/
```

ii. 执行以下命令查看文件系统大小。

```
df /mnt/disk/
```

预期输出：

```
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdb       30832548 45036  30771128   1% /mnt/disk
```

从以上预期输出可得：临时文件夹*/dev/vdb*的内存为30 GiB，说明已经扩容成功。

iii. 执行以下命令将云盘从临时文件夹卸载。

```
umount /mnt/disk
```

步骤四：重启应用

1. 执行以下命令将Pod扩容为2个副本。

```
kubectl scale sts web --replicas=2
```

预期输出：

```
statefulset.apps/web scaled
```

2. 执行以下命令查看Pod是否删除。

```
kubectl get pod | grep web
```

预期输出：

```
NAME    READY  STATUS   RESTARTS  AGE
web-0   1/1    Running  0         74s
web-1   1/1    Running  0         42s
```

3. 执行以下命令查看文件系统大小。

```
kubectl exec web-0 df /data
```

预期输出：

```
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdb       30832548 45036  30771128   1% /data
```

从以上预期输出可得，*/dev/vdb*文件系统存储空间为30 GiB。

常见问题

问题现象：

在执行 `resize2fs` 命令之后出现如下错误提示。

```
resize of device /dev/xxx failed: exit status 1 resize2fs output: resize2fs xxx(version)
Please run `e2fsck -f /dev/xxx` first
```

问题原因：

因为不满足文件系统的一致性导致IO崩溃。

解决方案：

请按照提示执行 `e2fsck -f /dev/xxx` 后再尝试扩容。

相关文档

- 在线扩容云盘数据卷

10.4.6. 自动扩容云盘数据卷（公测）

通过定义一种或多种扩容策略，在数据卷的使用率高于某个阈值时自动扩容卷。本文档主要介绍如何实现云盘数据卷的自动扩容。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已安装storage-operator组件，且组件版本不低于v1.18.8.28-18cca7b-aliyun。更多信息，请参见storage-operator。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。

注意事项

- 只支持云盘扩容，不支持缩容。
- 只有处于挂载状态的云盘才支持扩容。
- 依赖云盘的Resize能力。更多信息，请参见ResizeDisk。
- 触发扩容的最大时间间隔为2 min，云盘扩容时间为1 min，请确保不要在3 min内将云盘写满。
- 云盘容量最大为32 TiB。

使用限制

目前支持在马来西亚（吉隆坡）及华北2（北京）地域配置自动扩容云盘数据卷。

步骤一：配置云盘自动扩容策略

1. 创建StorageClass。

   若已创建StorageClass，则请跳过此步骤。

i. 使用以下内容，创建*storageclass.yaml*文件。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-disk-essd
provisioner: diskplugin.csi.alibabacloud.com
parameters:
 type: cloud_essd
reclaimPolicy: Delete
allowVolumeExpansion: true
```

ii. 执行以下命令创建StorageClass。

```
kubectl create -f storageclass.yaml
```

2. 通过CRD（Custom Resource Definitions）创建自动扩容策略。

i. 使用以下内容，创建*StorageAutoScalerPolicy.yaml*文件。

```
apiVersion: storage.alibabacloud.com/v1alpha1
kind: StorageAutoScalerPolicy
metadata:
 name: hybrid-expand-policy
spec:
 pvcSelector:
  matchLabels:
   app: mysql
 namespaces:
  - default
  - mysql
 conditions:
  - name: condition1
   key: volume-capacity-used-percentage
   operator: Gt
   values:
    - "80"
 actions:
  - name: action1
   type: volume-expand
   params:
    scale: 50Gi
    limits: 100Gi
  - name: action2
   type: volume-expand
   params:
    scale: 50%
    limits: 300Gi
```

参数	说明
pvcSelector	选择目标PVC，通过Label-Selector方式。本示例为mysql。
namespaces	表示目标PVC所在的命名空间，多个命名空间为或逻辑。若不配置，默认为default。本示例为default及mysql。

参数	说明
conditions	表示触发规则的条件，多个condition为与逻辑。每个condition包含以下参数： ■ key：定义一个Metric的类型。 ■ volume-capacity-used-percentage：表示容量使用百分比。 ■ operator：定义规则，包含Gt（大于）、Lt（小于）、Eq（等于）或Ne（不等于），不限制英文字母大小写。 ■ values：规则的具体数值。 本示例表示当PVC容量的使用率高于80%时会触发action。
actions	表示满足上述conditions时执行的操作，可以是多个操作。包含以下参数： ■ type：表示行为方式，目前只支持扩容。 ■ scale：表示扩容的大小，单位为GiB，或可使用百分比。 ■ limits：表示PVC在此action中的最大限制。 若actions中有多个action时，则从首个action开始匹配，执行首个满足条件的action，其余跳过。 ■ 本示例的action1表示云盘容量在100 GiB内以50 GiB为单位扩容，最大扩容到100 GiB。 ■ 本示例的action2表示当云盘容量大于100 GiB小于300 GiB时，以当前容量的50%扩容，即每次扩容后的总容量为扩容前容量的150%，最大扩容到300 GiB。

ii. 执行以下命令创建自动扩容策略。

```
kubectl create -f StorageAutoScalerPolicy.yaml
```

3. 创建工作负载StatefulSet。

i. 使用以下内容，创建*StatefulSet.yaml*文件。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
  matchLabels:
   app: mysql
 serviceName: "nginx"
 replicas: 3
 template:
  metadata:
   labels:
    app: mysql
  spec:
   containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
       - containerPort: 80
         name: web
      volumeMounts:
       - name: www
         mountPath: /usr/share/nginx/html
 volumeClaimTemplates:
  - metadata:
     name: www
     labels:
      app: mysql
    spec:
     accessModes: [ "ReadWriteOnce" ]
     storageClassName: "alicloud-disk-essd"
     resources:
      requests:
       storage: 25Gi
```

ii. 执行以下命令创建工作负载。

```
kubectl create -f StatefulSet.yaml
```

步骤二：验证云盘的自动扩容

1. 向挂载目录写入数据，使云盘容量使用率高于80%。

   i. 执行以下命令向挂载目录写入数据。

   ```
   dd if=<数据路径> of=<挂载路径>
   ```

   ii. 执行以下命令查看云盘容量详情。

   ```
   df -h | grep d-****1
   ```

   预期输出：

   ```
   Filesystem   Size  Used  Avail  Use%   Mounted on
   /dev/vde     25G   24G   1.5G   95%    /var/lib/kubelet/plugins/kubernetes.io/csi/pv/d-****1/globalmount
   ```

2. 执行以下命令查看扩容事件。

   当云盘容量使用率高于80%时会触发扩容，此时action1满足条件，则使用action1作为扩容策略。

```
kubectl get events
```

预期输出：

```
101s Warning StartExpand persistentvolumeclaim/www-web-1 Start to expand of pvc www-web-1 from 25Gi to 75Gi, usedCapacityPercentage:94%, freeSize:1472MB.
101s Warning ExternalExpanding persistentvolumeclaim/www-web-1 Ignoring the PVC: did't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
101s Normal Resizing persistentvolumeclaim/www-web-1 External resizer is resizing volume d-****1
97s Normal FileSystemResizeRequired persistentvolumeclaim/www-web-1 Require file system resize of volume on node
96s Warning SkipExpand persistentvolumeclaim/www-web-1 Pvc www-web-1 is expanding status from 25Gi to 75Gi, this action action2 will skip.
```

3. 执行以下命令查看PVC的容量。

```
kubectl get pvc
```

预期输出：

```
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
www-web-0 Bound d-****0 25Gi RWO alicloud-disk-essd 22m
www-web-1 Bound d-****1 75Gi RWO alicloud-disk-essd 21m
www-web-2 Bound d-****2 25Gi RWO alicloud-disk-essd 21m
```

从预期输出可得，云盘 d-****1 已扩容到75 GiB。

4. 根据步骤1，再次使云盘容量使用率高于80%，进行第二次扩容。
执行以下命令查看扩容事件。

```
kubectl get events
```

预期输出：

```
6m4s Warning StartExpand persistentvolumeclaim/www-web-1 Start to expand of pvc www-web-1 from 75Gi to 100Gi, usedCapacityPercentage:95%, freeSize:3732MB.
5m2s Warning ExternalExpanding persistentvolumeclaim/www-web-1 Ignoring the PVC: did't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
2m4s Normal Resizing persistentvolumeclaim/www-web-1 External resizer is resizing volume d-****1
3m4s Normal FileSystemResizeRequired persistentvolumeclaim/www-web-1 Require file system resize of volume on node
5m59s Warning SkipExpand persistentvolumeclaim/www-web-1 Pvc www-web-1 is expanding status from 75Gi to 100Gi, this action action2 will skip.
```

由于action1的limit为100 GiB，从预期输出可得，此次只能将云盘从75 GiB扩容到100 GiB。

5. 根据步骤1，再次使云盘容量使用率高于80%，进行第三次扩容。
执行以下命令查看扩容事件。

```
kubectl get events
```

预期输出：

```
7m22s Warning StartExpand persistentvolumeclaim/www-web-1 Start to expand of pvc www-web-1 from 100Gi to 150Gi, usedCapacityPercentage:95%, freeSize:3732MB.
5m2s Warning ExternalExpanding persistentvolumeclaim/www-web-1 Ignoring the PVC: did't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
2m4s Normal Resizing persistentvolumeclaim/www-web-1 External resizer is resizing volume d-****1
3m4s Normal FileSystemResizeRequired persistentvolumeclaim/www-web-1 Require file system resize of volume on node
5m59s Warning SkipExpand persistentvolumeclaim/www-web-1 Pvc www-web-1 is expanding status from 100Gi to 150Gi, this action action1 will skip.
```

由于action1的limit为100 GiB，不满足扩容条件。从预期输出可得，使用action2策略将云盘从100 GiB扩容到150 GiB。

6. 根据步骤1，再次使云盘容量使用率高于80%，进行第四次扩容。
   执行以下命令查看扩容事件。

   ```
   kubectl get events
   ```

   预期输出：

   ```
   0s Warning StartExpand persistentvolumeclaim/www-web-1 Start to expand of pvc www-web-1 from 150Gi to 225Gi, usedCapacityPercentage:94%, freeSize:7637MB.
   0s Warning ExternalExpanding persistentvolumeclaim/www-web-1 Ignoring the PVC: did't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
   0s Normal Resizing persistentvolumeclaim/www-web-1 External resizer is resizing volume d-****1
   0s Normal FileSystemResizeRequired persistentvolumeclaim/www-web-1 Require file system resize of volume on node
   0s Warning SkipExpand persistentvolumeclaim/www-web-1 Pvc www-web-1 is expanding status from 150Gi to 225Gi, this action action1 will skip.
   ```

   从预期输出可得，使用action2策略将云盘从150 GiB扩容到225 GiB。

7. 根据步骤1，再次使云盘容量使用率高于80%，进行第五次扩容。
   执行以下命令查看扩容事件。

   ```
   kubectl get events
   ```

   预期输出：

   ```
   0s Warning StartExpand persistentvolumeclaim/www-web-1 Start to expand of pvc www-web-1 from 225Gi to 300Gi, usedCapacityPercentage:94%, freeSize:7637MB.
   0s Warning ExternalExpanding persistentvolumeclaim/www-web-1 Ignoring the PVC: did't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
   0s Normal Resizing persistentvolumeclaim/www-web-1 External resizer is resizing volume d-****1
   0s Normal FileSystemResizeRequired persistentvolumeclaim/www-web-1 Require file system resize of volume on node
   0s Warning FileSystemResizeSuccessful persistentvolumeclaim/www-web-1 MountVolume.NodeExpandVolume succeeded for volume "d-****1"
   ```

   由于action2的limit为300 GiB，从预期输出可得，此次只能将云盘从225 GiB扩容到300 GiB。
   当云盘容量再次高于80%时，不再触发扩容。

10.4.7. 使用云盘存储快照

容器服务Kubernetes版ACK（Container Service for Kubernetes）的云盘存储快照特性可以帮助您实现应用数据的备份和恢复。本文介绍ACK的存储快照的基本概念、使用流程，及说明如何动态、静态创建快照。

前提条件

- ACK发布的Kubernetes 1.18及之上版本默认提供了存储快照的功能，因此使用云盘存储快照时，请确保您创建的ACK集群Kubernetes版本等于或大于1.18。具体操作，请参见创建Kubernetes托管版集群。
- 访问指定地域的ECS控制台，确认已开启云盘快照服务。具体操作，请参见开通快照。

背景信息

在阿里云ACK集群中部署有状态服务通常使用云盘数据卷存储数据，虽然云盘本身提供了数据的备份（快照）恢复机制，但是如何将这种机制和Kubernetes服务集成并灵活地提供给应用使用？为了解决这个问题，Kubernetes使用以下两个特性来实现备份恢复能力：

- 通过VolumeSnapshot资源实现云盘的备份（快照功能）。
- 通过PVC的DataSource功能实现数据的恢复。

使用说明

为了实现快照相关功能，ACK通过CRD定义了以下3个相关的资源类型。

资源类型名称	描述
VolumeSnapshotContent	存储后端的快照实例，由系统管理员创建维护，无NameSpace。类似PV概念。
VolumeSnapshot	声明一个快照实例，由用户创建维护，属于特定NameSpace。类似PVC概念。
VolumeSnapshotClass	定义一个快照类，描述创建快照使用的参数、Controller。类似StorageClass概念。

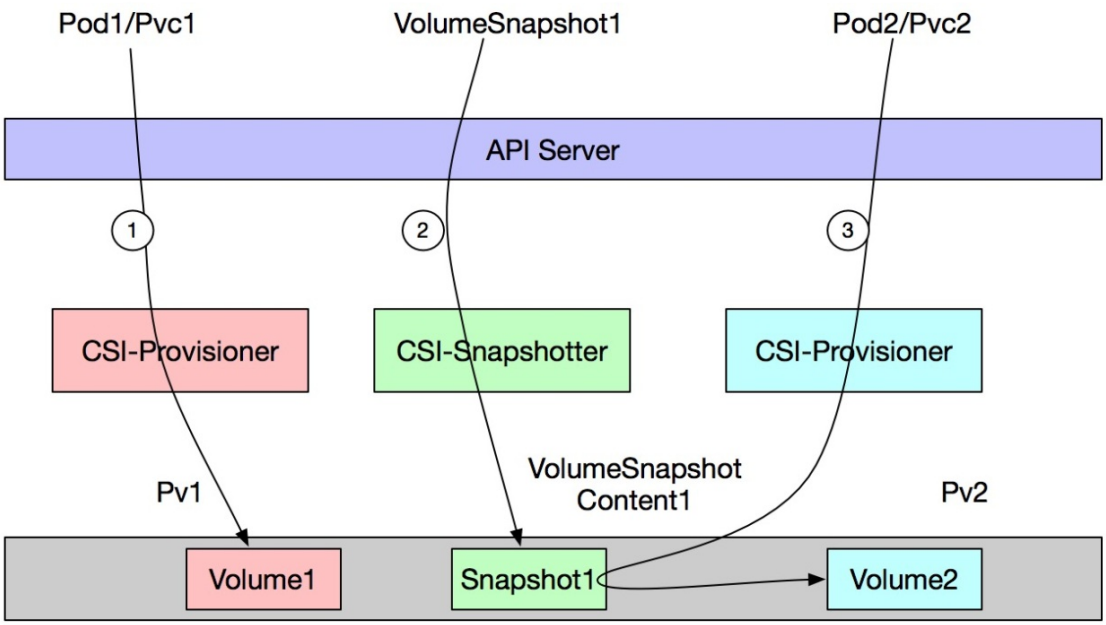
存储快照资源的绑定规则如下：

- 在使用Snapshot资源类型时，和PV、PVC一样，首先您需绑定VolumeSnapshot与VolumeSnapshotContent。
- 集群会自动为正确配置了VolumeSnapshotClassName字段的VolumeSnapshot资源，创建VolumeSnapshotContent资源。 如果您没有配置或者配置错误的话则无法自动创建VolumeSnapshotContent，您需要手动创建VolumeSnapshotContent，并绑定VolumeSnapshot。
- VolumeSnapshotContent与VolumeSnapshot绑定是一对一的关系。

> **注意** 删除VolumeSnapshotContent同时，后端的快照也会被删除。

动态创建快照

下图为在ACK使用的云盘上动态创建快照的使用流程图。



使用流程说明如下。

流程步骤	描述
①	创建原始应用，创建云盘卷保存数据。
②	创建VolumeSnapshot，此时集群会自动创建VolumeSnapshotContent和存储端的快照实例。
③	创建新的应用，并配置PVC引用步骤②中创建的快照对象。

上述的三个步骤实现：

- 备份：图中的**Volume1**的数据被备份到**Snapshot1**。
- 恢复：**Snapshot1**的数据（也就是**Volume1**的数据）被恢复成**Volume2**。

本文示例将通过创建Nginx应用并恢复应用的数据，说明如何使用存储快照功能。操作步骤如下。

1. 创建VolumeSnapshotClass快照类。

i. 使用以下YAML内容创建*volumesnapshotclass.yaml*文件。

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: default-snapclass
driver: diskplugin.csi.alibabacloud.com
parameters:
  snapshotType: "xxx"
  instantAccessRetentionDays: "1"
deletionPolicy: Delete
```

参数	描述
deletionPolicy	■ 当值为*Delete*的时候，说明删除VolumeSnapshot时，VolumeSnapshotContent以及关联的快照也会被一起删除。 ■ 当值为*Retain*的时候，说明删除VolumeSnapshot时，VolumeSnapshotContent以及关联的快照不会被删除。
parameters.snapshotType	指定为*InstantAccess*的时候，动态创建出来的是极速快照，只针对于ESSD云盘。更多信息，请参见开启或关闭快照极速可用功能。 默认值为*normal*，表示创建普通快照。
parameters.instantAccessRetentionDays	与参数parameters.snapshotType共同使用，指定了创建极速快照保留天数，过了该天数后，极速快照将会转换成普通快照。

ii. 执行以下命令创建VolumeSnapshotClass快照类。

```
kubectl apply -f volumesnapshotclass.yaml
```

2. 创建原始应用并写入数据。

i. 使用以下YAML内容创建*nginx.yaml*文件。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
  matchLabels:
   app: nginx
 serviceName: "nginx"
 replicas: 1
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
    volumeMounts:
    - name: disk-ssd
     mountPath: /data
 volumeClaimTemplates:
 - metadata:
   name: disk-ssd
  spec:
   accessModes: [ "ReadWriteOnce" ]
   storageClassName: "alicloud-disk-ssd"
   resources:
    requests:
     storage: 20Gi
```

ii. 执行以下命令创建应用Nginx。

```
kubectl apply -f nginx.yaml
```

iii. 执行以下命令往Pod（web-0）中写入数据。

```
kubectl exec -it web-0 touch /data/test
kubectl exec -it web-0 ls /data
```

输出：

```
lost+found test
```

3. 创建VolumeSnapshot。

i. 使用以下YAML内容创建*snapshot.yaml*文件。

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
 name: new-snapshot-demo
spec:
 volumeSnapshotClassName: default-snapclass
 source:
  persistentVolumeClaimName: disk-ssd-web-0
```

ii. 执行以下命令创建VolumeSnapshot。

```
kubectl apply -f snapshot.yaml
```

iii. 执行以下命令查看是否创建VolumeSnapshot和VolumeSnapshotContent。您也可以登录ECS控制台查看快照实例。

查看VolumeSnapshot命令：

```
kubectl get volumesnapshots.snapshot.storage.k8s.io
```

输出：

```
NAME                    AGE
new-snapshot-demo       36m
```

查看VolumeSnapshotContent命令：

```
kubectl get VolumeSnapshotContent
```

输出：

```
NAME                    AGE
snapshotcontent-222222  36m
```

4. 恢复数据。

i. 使用以下YAML内容创建*nginx-restore*文件。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 ports:
 - port: 80
  name: web
 clusterIP: None
 selector:
  app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web-restore
spec:
 selector:
  matchLabels:
   app: nginx
 serviceName: "nginx"
 replicas: 1
 template:
  metadata:
   labels:
    app: nginx
  spec:
   hostNetwork: true
   containers:
   - name: nginx
    image: nginx
    command: ["sh", "-c"]
    args: ["sleep 10000"]
    volumeMounts:
    - name: disk-ssd
     mountPath: /data
 volumeClaimTemplates:
 - metadata:
   name: disk-ssd
  spec:
   accessModes: [ "ReadWriteOnce" ]
   storageClassName: alicloud-disk-ssd
   resources:
    requests:
     storage: 20Gi
   dataSource:
    name: new-snapshot-demo
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

说明 在volumeClaimTemplates中，需设置dataSource的类型为VolumeSnapshot，且VolumeSnapshot的名字为步骤中创建的new-snapshot-demo。

ii. 执行以下命令恢复数据。

```
kubectl apply -f nginx-restore.yaml
```

5. 执行以下命令查看Pod（web-restore-0）数据。

```
kubectl exec -it web-restore-0 ls /data
```

输出：

```
lost+found test
```

可见在不同的Pod中返回相同数据，实现了数据的恢复。

静态创建快照（使用ECS上已有快照）

将已有ECS快照导入至ACK集群中的操作步骤如下。

1. 使用以下YAML内容创建VolumeSnapshotContent。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
 name: new-snapshot-content-test
spec:
 deletionPolicy: Retain
 driver: diskplugin.csi.alibabacloud.com
 source:
  snapshotHandle: <your-snapshotid>
 volumeSnapshotRef:
  name: new-snapshot-demo
  namespace: default
```

参数	描述
snapshotHandle	填写在ECS页面上的已生成的快照ID。
volumeSnapshotRef	填写要创建的VolumeSnapshot的信息： ○ name：将要创建的VolumeSnapshot的名称。 ○ namespace：将要创建的VolumeSnapshot所在的命名空间。

2. 使用以下YAML内容创建VolumeSnapshot。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
 name: new-snapshot-demo
spec:
 source:
  volumeSnapshotContentName: new-snapshot-content-test
```

参数	描述
metadata.name	VolumeSnapshot的名称，需要和上述VolumeSnapshotContent中指定的相同。
spec.source.volumeSnapshotContentName	是当前VolumeSnapshot绑定的VolumeSnapshotContent的名称，需与上述创建的VolumeSnapshotContent的名称一致。

3. 恢复数据。

   i. 使用以下YAML内容创建*nginx-restore*文件。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 ports:
 - port: 80
  name: web
 clusterIP: None
 selector:
  app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web-restore
spec:
 selector:
  matchLabels:
   app: nginx
 serviceName: "nginx"
 replicas: 1
 template:
  metadata:
   labels:
    app: nginx
  spec:
   hostNetwork: true
   containers:
   - name: nginx
    image: nginx
    command: ["sh", "-c"]
    args: ["sleep 10000"]
    volumeMounts:
    - name: disk-ssd
     mountPath: /data
 volumeClaimTemplates:
 - metadata:
   name: disk-ssd
  spec:
   accessModes: [ "ReadWriteOnce" ]
   storageClassName: alicloud-disk-ssd
   resources:
    requests:
     storage: 20Gi
   dataSource:
    name: new-snapshot-demo
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

> **说明** 在volumeClaimTemplates中，需设置dataSource的类型为VolumeSnapshot，且VolumeSnapshot的名字为步骤2中创建的new-snapshot-demo。

ii. 执行以下命令恢复数据。

```
kubectl apply -f nginx-restore.yaml
```

4. 执行以下命令查看Pod（web-restore-0）数据。

```
kubectl exec -it web-restore-0 ls /data
```

输出：

```
lost+found test
```

可见在不同的Pod中返回相同数据，实现了数据的恢复。

10.4.8. 加密云盘存储卷

数据加密适用于有高安全性或合规性要求的应用场景，您无需自建和维护密钥管理基础设施，通过加密保护存储在阿里云ECS上的数据就能保护数据的隐私性和自主性。本文介绍如何使用阿里云密钥管理服务KMS（Key Management Service）中管理的密钥对集群中的云盘存储卷数据加密。

前提条件

- 已创建好一个Kubernetes集群，并且在该集群中安装CSI插件。具体操作，请参见创建Kubernetes托管版集群。
- 已开通密钥管理服务。具体操作，请参见开通密钥管理服务。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。

功能介绍

ECS加密采用行业标准的AES-256加密算法，利用密钥加密云盘。密钥可以是服务密钥和普通密钥（BYOK）。在加密解密的过程中，实例的性能几乎没有衰减。

- 使用加密的系统盘（或镜像）创建ECS实例后，ECS实例操作系统内数据会被自动加密，并在读取数据时自动解密。具体步骤，请参见加密系统盘。
- 创建加密的数据盘并将其挂载到ECS实例后，以下数据会被自动加密，并在读取数据时自动解密。具体步骤，请参见加密数据盘。
  - 加密云盘中的静态数据。
  - 加密云盘和实例间传输的数据（不包括操作系统内的数据）。
  - 加密云盘从实例传递到后端存储集群的数据。
  - 从加密云盘创建的所有快照，并且该快照的加密密钥与云盘的加密密钥保持相同。
  - 从加密快照创建的所有云盘。

使用限制

- 支持加密的数据盘包括ESSD云盘、SSD云盘、高效云盘和普通云盘。
- 支持加密的系统盘包括ESSD云盘、SSD云盘和高效云盘。
- 不支持加密本地盘。
- 加密系统盘时，仅支持在复制自定义镜像时完成加密。加密后，不支持以下操作：
  - 转换加密镜像为非加密镜像。
  - 跨地域复制加密镜像。
  - 共享加密镜像。
  - 导出加密镜像。

- 非加密云盘不能直接转换成加密云盘。
- 加密云盘不能直接转换为非加密云盘。

配置加密云盘数据卷

在创建云盘时才可以启用云盘加密，而挂载或卸载云盘时并不需要有加密相关的配置和操作。

1. 配置StorageClass参数。
   i. 将以下内容复制到*sc-kms.yaml*文件中。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: csi-disk-encrypted
provisioner: diskplugin.csi.alibabacloud.com
parameters:
  fsType: ext4
  type: cloud_ssd
  encrypted: "true"
  kmsKeyId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
reclaimPolicy: Delete
```

   > 说明
   > - `encrypted`：配置创建的云盘是否为加密云盘，`true`表示生成加密云盘。
   > - `kmsKeyId`：表示创建加密云盘使用的KMS密钥，不配置时会使用默认的CMK；当`encrypted`为`false`时，此配置不生效。

   ii. 执行以下命令创建StorageClass。

```
kubectl create -f sc-kms.yaml
```

   iii. 执行以下命令查看生成的StorageClass。

```
kubectl get sc csi-disk
```

   预期输出：

```
NAME     PROVISIONER               AGE
csi-disk  diskplugin.csi.alibabacloud.com  9m5s
```

2. 创建PVC。
   i. 将以下内容复制到*sc-pvc.yaml*文件中。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: disk-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 20Gi
 storageClassName: csi-disk
```

ii. 执行以下命令创建PVC。

```
kubectl create -f sc-pvc.yaml
```

iii. 执行以下命令查看创建的PVC。

```
kubectl get pvc
```

预期输出：

```
NAME       STATUS  VOLUME              CAPACITY  ACCESS MODES  STORAGECLASS  AGE
disk-pvc    Bound  d-wz92s6d95go6ki9x****    25Gi    RWO       csi-disk    10m
```

iv. 执行以下命令查看PV。

```
kubectl get pv
```

预期输出：

```
NAME           CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM         STORAGECLASS  REA
SON  AGE
d-wz92s6d95go6ki9x****  25Gi    RWO       Retain      Bound   default/disk-pvc   csi-disk        10m
```

从以上预期输出可得，创建的云盘ID为：*d-wz92s6d95go6ki9x****。

3. 查看加密云盘是否生效。

i. 登录ECS管理控制台。

ii. 在左侧导航栏，选择**存储与快照 > 云盘**。

iii. 在**云盘**页面可看到*d-wz92s6d95go6ki9x****云盘为**已加密**状态。

10.4.9. 云盘存储卷FAQ

本文为您介绍云盘存储卷常见问题的处理方法。

- 使用云盘时提示The specified disk is not a portable disk
- 启动挂载了云盘的Pod时提示had volume node affinity conflict
- 启动挂载了云盘的Pod时提示can't find disk
- 动态创建PV失败且提示The specified AZone inventory is insufficient
- 动态创建PV失败且提示disk size is not supported
- 磁盘阻塞导致Pod长时间无法启动
- 启动Pod时出现FailedMount警告
- Pod卸载失败且Kubelet出现不受ACK管理的Pod日志
- Pod删除后无法重新启动，提示挂载失败且无法自动恢复。

使用云盘时提示 `The specified disk is not a portable disk`

问题现象：

卸载云盘时，提示 `The specified disk is not a portable disk` 。

问题原因：

您申请了包年包月的云盘，或者在升级ECS时，将ECS关联的云盘一起升级为包年包月。

解决方法：

将云盘的付费方式改为按量付费。

启动挂载了云盘的Pod时提示 `had volume node affinity conflict`

问题现象：

在启动挂载云盘的Pod的时候，出现Pod无法启动的情况并报错 had volume node affinity conflict 。

问题原因：

您在PV中编写了 nodeaffinity 属性，这个属性与Pod中声明的属性不一致，导致Pod无法被调度到正确的节点上。

解决方法：

修改PV或者Pod的属性，使二者属性保持一致。

启动挂载了云盘的Pod时提示 can't find disk

问题现象：

启动挂载了云盘的Pod时提示 can't find disk 。

问题原因：

- 您在编写PV的时候输入了错误的DiskID。
- 您的账号无权限操作DiskID，可能不是当前账号的DiskID。

解决方法：

更换DiskID。

动态创建PV失败且提示 The specified AZone inventory is insufficient

问题现象：

创建PV失败，PVC Event提示 The specified AZone inventory is insufficient 。

问题原因：

ECS库存不足，导致创建云盘失败。

解决方法：

更换云盘类型或者更换Azone。

动态创建PV失败且提示 disk size is not supported

问题现象：

动态创建PV失败，提示 disk size is not supported 。

问题原因：

在PVC中指定的云盘大小不符合规范，要求最小20 GiB。

解决方法：

调整PVC中云盘的大小，使其符合规范。

磁盘阻塞导致Pod长时间无法启动

问题现象：

磁盘出现阻塞，kubelet日志出现以下大量报错。

```
Operation for "{volumeName:kubernetes.io/csi/diskplugin.csi.alibabacloud.com^d-2zejaz33icbp2vvv**** podName
: nodeName:}" failed.
No retries permitted until 2020-11-05 14:38:12.653566679 +0800 CST m=+9150650.781033052 (durationBeforeRetry
2m2s).
Error: "MountVolume.MountDevice failed for volume \"d-2zejaz33icbp2vvv****\" (UniqueName: \"kubernetes.io/csi
/diskplugin.csi.alibabacloud.com^d-2zejaz33icbp2vvv****\") pod \"pod-e5ee2d454cdb4d1d916d933495e56cbe-358
****\" (UID: \"f8d71e90-d934-4d5a-b54f-62555da5****\") : rpc error: code = Aborted desc = NodeStageVolume: Previ
ous attach action is still in process
```

问题原因：

您使用的是旧版本的CSI，旧版本的CSI使用了BlkID通过UID获取盘符。由于使用快照恢复的云盘有着相同的UID，导致BlkID命令执行阻塞。

解决方法：

重启当前节点，或升级CSI到最新版本。

启动Pod时出现FailedMount警告

问题现象：

启动Pod的时候，出现以下警告。

```
Warning  FailedMount     104s            kubelet, cn-zhangjiakou.172.20.XX.XX  Unable to attach or mount volumes: unmounted volumes=[sysdata-nas], unattached volumes=[kun-log kun-script kun-app sysdata-nas kun-patch default-token-rbx8p kun-etc kun-bin]: timed out waiting for the condition
Warning  FailedMount     98s (x9 over 3m45s)  kubelet, cn-zhangjiakou.172.20.XX.XX  MountVolume.MountDevice failed for volume "nas-9d9ead08-8a1d-4463-a7e0-7bd0e3d3****" : kubernetes.io/csi: attacher.MountDevice failed to create newCsiDriverClient: driver name nasplugin.csi.alibabacloud.com not found in the list of registered CSI drivers
```

问题原因：

一般出现在新增的节点。由于CSI的Pod是和业务Pod一起启动，且CSI注册需要一定时间，所以业务Pod开始挂载的时候，CSI还没有注册完成，导致出现警告。

解决方法：

无需处理，不影响Pod启动。

Pod卸载失败且Kubelet出现不受ACK管理的Pod日志

问题现象：

Pod卸载失败，并且Kubelet出现不受ACK管理的Pod日志。

问题原因：

Pod异常退出，导致数据卷挂载点在卸载过程中没有清理干净，最终导致Pod无法删除。Kubelet的GC流程对数据卷垃圾回收实现并不完善，目前需要手动或脚本自动化实现垃圾挂载点的清理工作。

解决方法：

在问题节点运行以下脚本，对垃圾挂载点进行清理。

```
wget https://raw.githubusercontent.com/AliyunContainerService/kubernetes-issues-solution/master/kubelet/kubelet.sh
sh kubelet.sh
```

Pod删除后无法重新启动，提示挂载失败且无法自动恢复。

问题现象：

Pod在删除之后无法拉起，报出以下异常提示，且无法自动恢复。

```
Warning FailedMount 9m53s (x23 over 40m) kubelet MountVolume.SetUp failed for volume “xxxxx” : rpc error: code = Internal desc = stat /var/lib/kubelet/plugins/kubernetes.io/csi/pv/xxxxx/globalmount: no such file or directory
```

影响范围：

- ACK集群版本为1.20.4-aliyun-1。
- 应用的存储介质为云盘。
- 使用StatefulSet并且设置了 `podManagementPolicy: "Parallel"` 属性。

问题原因：

更多信息，请参见Pod fails to start after restarting rapidly。

解决方法：

- 为集群新增节点后移除旧节点，将旧节点全部替换为新节点，有故障的Pod将自动恢复。具体操作，请参见扩容节点池及移除节点。
- 调整StatefulSet为 `orderedready` 或去除 `podManagementPolicy: "Parallel"` 属性。
- 如果当前集群节点数较少，可以采用以下方案：
  i. 将当前Pod所在的节点添加 `cordon` 标签，设置为不可调度状态。
  ii. 删除Pod，等待Pod状态变为Pending。
  iii. 去除节点上的 `cordon` 标签，等待Pod重启。
- 如果当前集群节点数较多，则将Pod调度到其他节点后可正常启动。

若有其他疑问，请联系我们。

10.5. 容器网络文件系统

10.5.1. 容器网络文件系统CNFS概述

传统的共享文件系统存在缺少容量配额的精确控制、无法恢复误删文件、缺失存储卷容量性能等监控指标、无安全加密及小文件读写延迟等问题。阿里云容器服务ACK推出了容器网络文件系统CNFS提升NAS文件系统的性能，QoS控制。本文介绍容器网络文件系统CNFS的功能、存储规格、适用场景、使用限制及计费说明。

功能介绍

阿里云容器服务使用容器网络文件系统CNFS（Container Network File System），将阿里云的文件存储抽象为一个K8s对象（CRD）进行独立管理，包括创建、删除、描述、挂载，监控及扩容等运维操作。

功能特性	CNFS版本
回收站	支持打开文件回收站功能，避免误删除。
资源配额	存储卷的容器配额（Quota）限制，支持动态扩容提升容量上限。关于如何使用容器配额能力，请参见配置NAS存储卷容量。
存储卷监控	支持PVC级别的容量及IO性能监控。关于如何使用监控，请参见使用csi-plugin组件监控节点侧存储资源。
生命周期	对NAS文件系统，以及OSS Bucket全生命周期管理。
资源保护	避免误删除整个文件系统。
无损升级	热升级存储驱动支持。

创建CNFS后，NAS文件系统的部分高级功能需要在文件存储管理控制台配置（例如，回收站、资源配额、生命周期等功能）。

存储规格

NAS提供了通用容量型、通用性能型以及极速型存储类型。更多信息，请参见规格类型。

使用说明

- 简化存储卷声明的方式，减少NAS控制台与ACK控制台之间多次切换挂载NAS存储卷，导致容易挂载失败且操作冗余。具体操作，请参见使用CNFS托管NAS文件系统。
- 通过定义扩容策略，在NAS存储卷的使用率高于某个阈值时触发自动扩容。具体操作，请参见使用CNFS自动扩容

NAS存储卷。

- 阿里云CSI支持创建多个PV，且多个PV可以共享同一个NAS的子目录。具体操作，请参见使用CNFS动态创建NAS共享存储卷。
- 支持CNFS文件回收站功能，避免文件误删除。具体操作，请参见使用CNFS的回收站功能恢复数据。

计费说明

关于NAS的计费说明，请参见产品定价。

10.5.2. 使用CNFS托管NAS文件系统

为了提升NAS文件系统的性能，容器服务ACK使用CNFS（Container Network File System）托管NAS文件系统，对文件系统层独立管理。本文介绍如何使用CNFS托管NAS文件系统及CNFS在工作负载中的应用。

前提条件

- 已创建Kubernetes集群，且存储插件选择为CSI。具体操作，请参见创建Kubernetes托管版集群。
- csi-plugin和csi-provisioner组件版本不低于v1.20.5-ff6490f-aliyun。关于升级CSI-Plugin和CSI-Provisioner组件的操作，请参见升级CSI-Plugin和CSI-Provisioner。
- storage-operator组件版本不低于v1.18.8.56-2aa33ba-aliyun。关于升级storage-operator组件的操作，请参见管理组件。
- 已通过kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。
- 若您需要对NAS文件系统进行KMS加密，请先开通KMS服务。具体操作，请参见开通密钥管理服务。

功能介绍

可通过以下三种方式使用CNFS托管NAS文件系统：

- 方式一：使用CNFS创建默认NAS文件系统
  创建一个默认的CNFS文件系统，并使用NAS动态存储卷自动挂载默认创建的NAS文件系统，同时将NAS动态存储卷绑定到工作负载Deployment和StatefulSet中。
- 方式二：使用CNFS创建自定义NAS文件系统
  使用CNFS创建自定义的NAS文件系统，并使用NAS静态存储卷或NAS动态存储卷挂载自定义创建的NAS文件系统，然后将创建的NAS静态存储卷或NAS动态存储卷绑定到工作负载Deployment中。
- 方式三：使用已有的NAS文件系统创建CNFS
  使用CNFS配置已有的NAS文件系统，并使用NAS静态存储卷或NAS动态存储卷挂载已有的NAS文件系统，然后将创建的NAS静态存储卷或NAS动态存储卷绑定到工作负载Deployment中。

方式一：使用CNFS创建默认NAS文件系统

执行以下内容，创建一个默认的CNFS文件系统，并使用NAS动态存储卷自动挂载到默认创建的NAS文件系统中，同时将NAS动态存储卷绑定到工作负载Deployment和StatefulSet中。

```
# 创建CNFS、storageclass和deployment、statefulset对象。
cat << EOF | kubectl apply -f -
apiVersion: storage.alibabacloud.com/v1beta1
kind: ContainerNetworkFileSystem
metadata:
 name: cnfs-nas-filesystem
spec:
 description: "cnfs"
 type: nas
 reclaimPolicy: Retain #只支持Retain策略，删除CNFS时并不会删除NAS文件系统。
 parameters:
  encryptType: SSE-KMS #可选参数，不指定表示对文件不使用NAS托管加密，指定SSE-KMS表示开启此功能。
  enableTrashCan: "true" #可选参数，不指定表示不打开回收站功能，指定true表示开启此功能。
---
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
 name: alibabacloud-cnfs-nas
mountOptions:
 - nolock,tcp,noresvport
 - vers=3
parameters:
 volumeAs: subpath
 containerNetworkFileSystem: cnfs-nas-filesystem
 path: "/"
provisioner: nasplugin.csi.alibabacloud.com
reclaimPolicy: Retain
allowVolumeExpansion: true #可选参数，指定为true表示允许对NAS文件系统进行扩容。
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: cnfs-nas-pvc
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alibabacloud-cnfs-nas
 resources:
  requests:
   storage: 70Gi #如果打开目录限额功能，则storage字段会生效，动态创建目录写入数据量最大为70 GiB。
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: cnfs-nas-deployment
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.14.2
    volumeMounts:
    - mountPath: "/data"
     name: cnfs-nas-pvc
   volumes:
   - name: cnfs-nas-pvc
    persistentVolumeClaim:
     claimName: cnfs-nas-pvc
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: cnfs-nas-sts
 labels:
  app: nginx
```

```
spec:
 serviceName: "nginx"
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.14.2
    volumeMounts:
    - mountPath: "/data"
     name: www
 volumeClaimTemplates:
 - metadata:
   name: www
  spec:
   accessModes: [ "ReadWriteOnce" ]
   storageClassName: "alibabacloud-cnfs-nas"
   resources:
    requests:
     storage: 50Gi #如果打开目录限额功能，则storage字段会生效，动态创建目录写入数据量最大为50 GiB。
EOF
```

注意 根据当前集群所在的VPC创建NAS。若当前地域不支持容量型NAS，则创建性能型NAS，默认对文件不进行加密处理。

方式二：使用CNFS创建自定义NAS文件系统

使用CNFS创建自定义的NAS文件系统，并使用NAS静态存储卷或NAS动态存储卷挂载自定义创建的NAS文件系统，然后将创建的NAS静态存储卷或NAS动态存储卷绑定到工作负载Deployment中。

1. 创建自定义的NAS文件系统。
   i. 执行以下命令创建自定义的NAS文件系统。

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.alibabacloud.com/v1beta1
kind: ContainerNetworkFileSystem
metadata:
 name: cnfs-nas-filesystem
spec:
 description: "cnfs"
 type: nas
 reclaimPolicy: Retain
 parameters:
  filesystemType: standard
  storageType: Capacity
  protocolType: NFS
  encryptType: SSE-KMS
  enableTrashCan: true
  trashCanReservedDays: 5
  vSwitchId: vsw-2ze9l3ppwzg6bl02j****
EOF
```

参数	说明
（可选）description	当前文件系统的简单描述。
type	需要创建的存储卷类型。
reclaimPolicy	回收策略，目前仅支持Retain策略，删除CNFS时并不会删除NAS文件系统。
（可选）parameters.filesystemType	文件系统类型。默认为通用型standard。
（可选）parameters.storageType	存储类型，当filesystemType为standard时，可选择性能型Performance或容量型CapacityNAS文件系统。
（可选）parameters.protocolType	文件传输协议类型为NFS，目前仅支持NFS V3协议。
（可选）parameters.encryptType	加密方式，None表示不加密，SSE-KMS表示使用NAS服务端KMS加密。
（可选）parameters.enableTrashCan	是否开启回收站功能，false表示不开启回收站功能，true表示开启回收站功能。
（可选）parameters.trashCanReservedDays	表示回收站文件保留的最长时间，默认为7天。示例中 `trashCanReservedDays: 5` ，表示回收站的文件最长保留5天。
（可选）parameters.vSwitchId	创建出的NAS文件系统需要使用的交换机ID。

ii. 执行以下命令查看创建的NAS文件系统。

```
kubectl get cnfs
```

预期输出：

```
NAME                  AGE
cnfs-nas-filesystem   6d
```

iii. 执行以下命令查看NAS文件系统的详细信息。

```
kubectl get cnfs/cnfs-nas-filesystem -o yaml
```

预期输出：

```
apiVersion: storage.alibabacloud.com/v1beta1
kind: ContainerNetworkFileSystem
metadata:
 annotations:
  kubectl.kubernetes.io/last-applied-configuration: |
   {"apiVersion":"storage.alibabacloud.com/v1beta1","kind":"ContainerNetworkFileSystem","metadata"
:{"annotations":{},"name":"nas-load-mount-target"},"spec":{"description":"filesystem4","parameters":{"
filesystemId":"17f7e4****","server":"17f7e4****-hlm35.cn-beijing.nas.aliyuncs.com"},"reclaimPolicy":"Re
tain","type":"nas"}}
 creationTimestamp: "2021-05-14T08:20:09Z"
 finalizers:
 - protection.alibabacloud.com/cnfs
 generation: 6
 name: cnfs-nas-filesystem
 resourceVersion: "122342382"
 selfLink: /apis/storage.alibabacloud.com/v1beta1/containernetworkfilesystems/nas-load-mount-target
 uid: a9e9650c-68b2-405b-8274-0f5b6063****
spec:
 description: "cnfs"
 type: nas
 reclaimPolicy: Retain
 parameters:
  filesystemType: standard
  storageType: Capacity
  protocolType: NFS
  encryptType: SSE-KMS
  vSwitchId: vsw-XXX
  enableTrashCan: true
status:
 conditions:
 - lastProbeTime: "2021-05-14 16:20:15"
  reason: The nas filesystem and mount target complete initialization.
  status: Ready
 fsAttributes:
  accessGroupName: DEFAULT_VPC_GROUP_NAME
  encryptType: SSE-KMS
  enableTrashCan: true
  filesystemId: 17f7e48ece
  filesystemType: standard
  protocolType: NFS
  regionId: cn-beijing
  server: 17f7e48ece-hlm35.cn-beijing.nas.aliyuncs.com
  storageType: Capacity
  vSwitchId: vsw-2ze9l3ppwzg6bl02j****
  vpcId: vpc-2ze9sgmehjvwv5x74****
  zoneId: cn-beijing-h
 status: Available
```

参数	说明
status	CNFS的当前状态，包括Pending（等待）、Creating（创建中）、Initialization（创建文件系统中）、Available（可用）、Unavailable（暂时不可用，可以恢复为可用状态）、Fatal（不可用，无法恢复）、Terminating（删除中）。
conditions.lastProbeTime	最后探活的时间。

参数	说明
conditions.reason	处于当前状态的原因。
conditions.status	当前状态是否可用，可用为Ready，不可用为NotReady。
fsAttributes.accessGroupName	挂载点所应用的权限组名称，目前仅支持DEFAULT_VPC_GROUP_NAME（专有网络默认权限组）。
fsAttributes.encryptType	加密方式，None表示不加密，SSE-KMS表示使用NAS全托管的服务端KMS加密。
fsAttributes.enableTrashCan	是否开启回收站功能，false表示不开启回收站功能，true表示开启回收站功能。
fsAttributes.filesystemId	文件系统ID。
fsAttributes.filesystemType	文件系统类型。默认为通用型standard。
fsAttributes.protocolType	文件传输协议类型。目前支持NFS协议。
fsAttributes.regionId	CNFS所在的地域。
fsAttributes.server	CNFS的挂载点Domain。
fsAttributes.storageType	存储类型，当filesystemType为standard时，可选择性能型Performance或容量型CapacityNAS文件系统。
fsAttributes.vSwitchId	CNFS使用的vSwitch。
fsAttributes.vpcId	CNFS所在的VPC。
fsAttributes.zoneId	CNFS所在的Zone。

2. 通过PV绑定NAS文件系统。

   通过创建静态PV或动态PV绑定NAS文件系统：

   - 创建静态PV。

a. 执行以下内容创建引用NAS文件系统的PV对象。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolume
metadata:
 name: cnfs-nas-pv
 labels:
  alicloud-pvname: cnfs-nas-pv
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 csi:
  driver: nasplugin.csi.alibabacloud.com
  volumeHandle: cnfs-nas-pv #需要与pv的name保持一致
  volumeAttributes:
   containerNetworkFileSystem: cnfs-nas-filesystem
   path: "/"
   mode: "644"
 mountOptions:
  - nolock,tcp,noresvport
  - vers=3
EOF
```

参数	说明
containerNetworkFileSystem	指定需要使用的CNFS名称。
path	挂载存储卷在CNFS中使用的路径。

b. 执行以下命令查看PV是否创建成功。

```
kubectl get pv
```

预期输出：

```
NAME      CAPACITY  ACCESS MODES   RECLAIM POLICY  STATUS    CLAIM  STORAGECLASS  REASON  A
GE
cnfs-nas-pv  5Gi     RWX        Retain       Available                  4s
```

- 创建动态PV。

a. 执行以下内容创建引用NAS文件系统的PV对象。

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alibabacloud-nas-cnfs
mountOptions:
 - nolock,tcp,noresvport
 - vers=3
parameters:
 volumeAs: subpath
 containerNetworkFileSystem: nas-load-mount-target
 path: "/"
provisioner: nasplugin.csi.alibabacloud.com
reclaimPolicy: Retain
allowVolumeExpansion: true
EOF
```

> **说明** allowVolumeExpansion：取值为true或false。表示是否开启Quota及扩容功能。

b. 执行以下命令查看PV是否创建成功。

```
kubectl get pv
```

预期输出：

```
NAME      CAPACITY  ACCESS MODES   RECLAIM POLICY  STATUS    CLAIM  STORAGECLASS  REASON  A
GE
cnfs-nas-pv  5Gi     RWX        Retain       Available                 4s
```

3. 创建PVC。

i. 执行以下内容创建引用NAS文件系统的PVC。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: cnfs-nas-pvc
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alibabacloud-nas-cnfs
 resources:
  requests:
   storage: 70Gi
EOF
```

4. 创建应用。

i. 执行以下内容创建引用PVC的应用。

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
 name: cnfs-nas-deployment
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
       - containerPort: 80
      volumeMounts:
       - name: cnfs-nas-pvc
         mountPath: "/data"
   volumes:
    - name: cnfs-nas-pvc
      persistentVolumeClaim:
       claimName: cnfs-nas-pvc
EOF
```

ii. 执行以下命令查看所创建应用的工作状态。

```
kubectl get pod
```

预期输出：

```
NAME                    READY  STATUS   RESTARTS  AGE
cnfs-nas-deployment-86959b****  1/1    Running  0     2s
```

从预期输出可得，所创建的Deployment为Running状态，表示CNFS在该Deployment中使用成功。

方式三：使用已有的NAS文件系统创建CNFS

使用CNFS配置已有的NAS文件系统，并使用NAS静态存储卷或NAS动态存储卷挂载已有的NAS文件系统，然后将创建的NAS静态存储卷或NAS动态存储卷绑定到工作负载Deployment中。

1. 使用已有的NAS文件系统创建CNFS。

i. 执行以下内容使用已有的NAS文件系统创建CNFS。

```
# 当NAS文件系统存在时，加载已创建的NAS文件系统
cat <<EOF | kubectl apply -f -
apiVersion: storage.alibabacloud.com/v1beta1
kind: ContainerNetworkFileSystem
metadata:
 name: cnfs-nas-filesystem
spec:
 description: "cnfs"
 type: nas
 reclaimPolicy: Retain
 parameters:
  server: 17f7e4****-hlm35.cn-beijing.nas.aliyuncs.com
EOF
```

参数	说明
description	当前文件系统的简单描述。
type	需要创建的存储类型。
reclaimPolicy	回收策略，目前仅支持Retain策略，删除CNFS时并不会删除NAS文件系统。
parameters.server	NAS的挂载点URL地址。

ii. 执行以下命令查看NAS文件系统。

```
kubectl get cnfs
```

预期输出：

```
NAME              AGE
cnfs-nas-filesystem   6d
```

iii. 执行以下命令查看NAS文件系统的详细信息。

```
kubectl get cnfs/cnfs-nas-filesystem -o yaml
```

预期输出：

```
apiVersion: storage.alibabacloud.com/v1beta1
kind: ContainerNetworkFileSystem
metadata:
 annotations:
  kubectl.kubernetes.io/last-applied-configuration: |
   {"apiVersion":"storage.alibabacloud.com/v1beta1","kind":"ContainerNetworkFileSystem","metadata"
:{"annotations":{},"name":"nas-load-mount-target"},"spec":{"description":"filesystem4","parameters":{"
filesystemId":"17f7e4****","server":"17f7e48ece-h****.cn-beijing.nas.aliyuncs.com"},"reclaimPolicy":"Ret
ain","type":"nas"}}
 creationTimestamp: "2021-05-14T08:20:09Z"
 finalizers:
 - protection.alibabacloud.com/cnfs
 generation: 6
 name: cnfs-nas-filesystem
 resourceVersion: "122342382"
 selfLink: /apis/storage.alibabacloud.com/v1beta1/containernetworkfilesystems/nas-load-mount-target
 uid: a9e9650c-68b2-405b-8274-0f5b6063****
spec:
 description: cnfs
 parameters:
  server: 17f7e48ece-h****.cn-beijing.nas.aliyuncs.com
 reclaimPolicy: Retain
 type: nas
status:
 conditions:
 - lastProbeTime: "2021-05-14 16:20:15"
  reason: The nas filesystem and mount target complete initialization.
  status: Ready
 fsAttributes:
  accessGroupName: DEFAULT_VPC_GROUP_NAME
  encryptType: None
  enableTrashCan: true
  filesystemId: 17f7e4****
  filesystemType: standard
  protocolType: NFS
  regionId: cn-beijing
  server: 17f7e48ece-h****.cn-beijing.nas.aliyuncs.com
  storageType: Capacity
  vSwitchId: vsw-2ze9l3ppwzg6bl02j****
  vpcId: vpc-2ze9sgmehjvwv5x74****
  zoneId: cn-beijing-h
 status: Available
```

2. 在NAS存储卷中应用CNFS。具体操作，请参见方式二：使用CNFS创建自定义NAS文件系统的步骤到步骤。

后续步骤

关于如何监控节点侧NAS存储资源，请参见NAS存储监控使用示例。

10.5.3. 使用CNFS自动扩容NAS存储卷

通过定义CNFS中托管的NAS存储卷的扩容策略，可在NAS存储卷的使用率高于某个阈值时触发自动扩容。本文介绍如何使用CNFS实现NAS存储卷的自动扩容。

前提条件

- 已创建Kubernetes集群，且存储插件选择为CSI。具体操作，请参见创建Kubernetes托管版集群。

- 若需要新建集群，选择CSI存储插件时请同时选中**创建默认NAS文件系统和CNFS容器网络文件系统动态存储类型**。
- 若已创建集群，但未选中**创建默认NAS文件系统和CNFS容器网络文件系统动态存储类型**，请使用CNFS托管NAS文件系统。具体操作，请参见使用CNFS托管NAS文件系统。

- csi-plugin和csi-provisioner组件版本不低于v1.20.5-ff6490f-aliyun。关于升级CSI-Plugin和CSI-Provisioner组件的操作，请参见升级CSI-Plugin和CSI-Provisioner。
- storage-operator组件版本不低于v1.18.8.56-2aa33ba-aliyun。关于升级storage-operator组件的操作，请参见管理组件。
- 已通过kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

步骤一：配置NAS自动扩容策略

1. 查看CNFS对象的状态。

   需要确保CNFS对象的状态为**Available**。

   i. 执行以下命令查看CNFS对象。

   ```
   kubectl get cnfs
   ```

   预期输出：

   ```
   NAME                        AGE
   default-cnfs-nas-837d6ea-20210819155623   14d
   ```

   ii. 执行以下命令查看CNFS对象的状态。

   ```
   kubectl get cnfs default-cnfs-nas-837d6ea-20210819155623 -o yaml | grep Available
   ```

   预期输出：

   ```
   status: Available
   ```

2. 使用以下YAML模板创建NAS扩容策略。

```
cat << EOF | kubectl apply -f -
apiVersion: storage.alibabacloud.com/v1alpha1
kind: StorageAutoScalerPolicy
metadata:
 name: hybrid-expand-policy
spec:
 pvcSelector:
  matchLabels:
   app: nginx    #需要与PVC及Deployment下labels的配置相同，本示例均为app: nginx。
 namespaces:
  - default
  - nginx
 conditions:
  - name: condition1
    key: volume-capacity-used-percentage
    operator: Gt
    values:
     - "80"
 actions:
  - name: action1
    type: volume-expand
    params:
     scale: 100%
     limits: 500Gi
EOF
```

参数	说明
pvcSelector	选择目标PVC，通过Label-Selector方式实现扩容策略与PVC之间的匹配。本示例为nginx。
namespaces	表示目标PVC所在的命名空间，多个命名空间为或逻辑。若不配置，默认为default。本示例为default及nginx。
conditions	表示触发规则的条件，多个condition为与逻辑。每个condition包含以下参数： ○ key：定义一个Metric的类型。 ○ volume-capacity-used-percentage：表示容量使用百分比。 ○ operator：定义规则，包含Gt（大于）、Lt（小于）、Eq（等于）或Ne（不等于），不限制英文字母大小写。 ○ values：规则的具体数值。 本示例表示当PVC容量的使用率高于80%时会触发actions。
actions	表示满足上述conditions时执行的操作，可以是多个操作。包含以下参数： ○ type：表示行为方式，目前只支持扩容。 ○ scale：表示扩容的大小，单位为GiB，或可使用百分比。 ○ limits：表示PVC在此action中的最大限制。 若actions中有多个action时，则从首个action开始匹配，执行首个满足条件的action，其余跳过。 本示例的action1表示，当满足扩容条件时NAS存储卷容量以当前容量的100%为单位扩容，最大扩容到500 GiB。

3. 使用以下YAML模板创建PVC及Deployment。

> **注意** 通过匹配扩容策略与对应PVC及Deployment之间的标签，将扩容策略应用到PVC及Deployment中。例如，本示例扩容策略中pvcSelector.matchLabels的值为 `app: nginx` ，对应PVC及Deployment中labels的值为 `app: nginx` 。

```
cat << EOF | kubectl apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: cnfs-nas-pvc
 labels:
  app: nginx  #需要与扩容策略YAML模板下pvcSelector.matchLabels的配置相同，本示例均为app: nginx。
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alibabacloud-cnfs-nas
 resources:
  requests:
   storage: 50Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: cnfs-nas-deployment
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx  #需要与扩容策略YAML模板下pvcSelector.matchLabels的配置相同，本示例均为app: nginx。
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.14.2
    volumeMounts:
    - mountPath: "/data"
     name: cnfs-nas-pvc
   volumes:
   - name: cnfs-nas-pvc
    persistentVolumeClaim:
     claimName: cnfs-nas-pvc
EOF
```

步骤二：验证NAS存储卷的自动扩容

1. 执行以下命令查看命名空间default下的所有Pod。

```
kubectl get pod
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE
cnfs-nas-deployment-56dbcc7fb7-wh79z  1/1    Running      0     20m
```

2. 执行以下命令进入应用cnfs-nas-deployment-56dbcc7fb7-wh79z。

```
kubectl exec cnfs-nas-deployment-56dbcc7fb7-wh79z -ti sh
```

3. 执行以下命令在NAS存储卷挂载目录写入50 GiB的测试数据。

```
cd /data
dd if=<数据路径> of=<挂载路径>
```

4. 执行以下命令查看触发扩容的事件。

```
kubectl get events
```

预期输出：

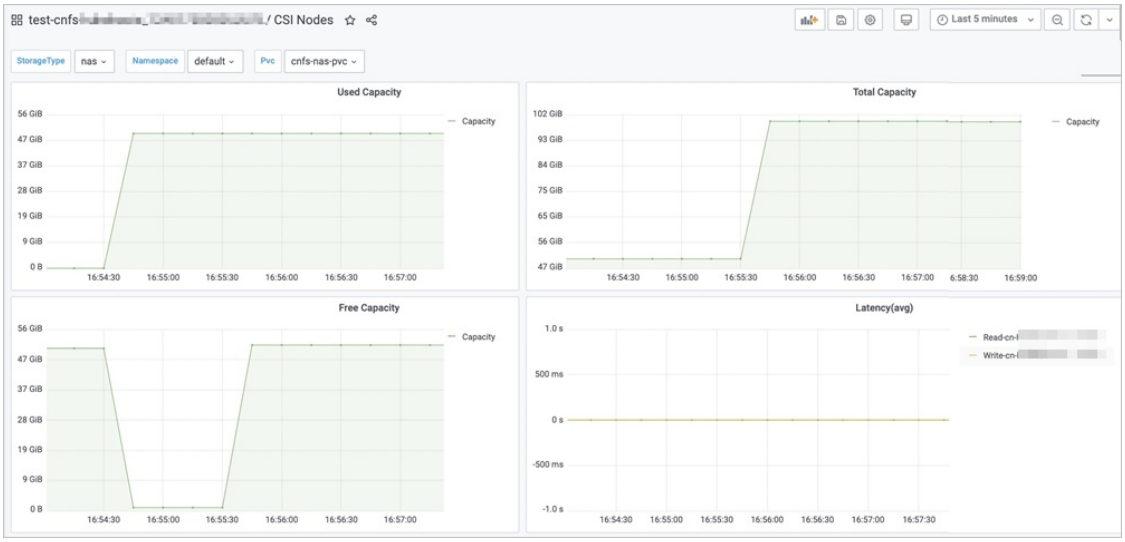
```
default 0s Warning NotEnoughDiskSpace persistentvolumeclaim/cnfs-nas-pvc Pvc cnfs-nas-pvc i
s not enough disk space, namespace: default, totalSize:50Gi, usedSize:49Gi, usedPercentage:98.00%, threshold
:85.00%
default 1s Warning StartExpand persistentvolumeclaim/cnfs-nas-pvc Start to expand of pvc c
nfs-nas-pvc from 50Gi to 100Gi, usedCapacityPercentage:98%, freeSize:1024MB.
default 0s Normal Resizing persistentvolumeclaim/cnfs-nas-pvc External resizer is resizing v
olume nas-1acba306-e626-46f3-8441-110c10a6****
default 0s Warning ExternalExpanding persistentvolumeclaim/cnfs-nas-pvc Ignoring the PVC: did
n't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
default 0s Normal FileSystemResizeRequired persistentvolumeclaim/cnfs-nas-pvc Require file syst
em resize of volume on node
default 0s Normal FileSystemResizeSuccessful pod/cnfs-nas-deployment-56dbcc7fb7-wh79z MountVol
ume.NodeExpandVolume succeeded for volume "nas-1acba306-e626-46f3-8441-110c10a6****"
```

从以上预期输出可得，当前NAS存储卷的容量使用量已超过80%，触发扩容，NAS存储卷容量将从50 GiB扩容到100 GiB。

5. 在CSI Nodes监控大盘查看CNFS监控信息。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**运维管理 > Prometheus监控**。
   v. 在**Prometheus监控**页面，单击**CSI Nodes**页签。

vi. 在**CSI Nodes**页签，选择NAS存储卷所在的**StorageType**、**Namespace**及PVC名称后，在**Total Capacity**区域可查看NAS存储卷当前容量。

本示例**StorageType**选择为**nas**、**Namespace**选择为**default**、PVC为**cnfs-nas-pvc**。



从上图可看到，在16:55:30触发扩容，且扩容后容量为100 GiB。

10.5.4. 使用CNFS动态创建NAS共享存储卷

当您的多个Kubernetes应用或Pod需要挂载相同的NAS存储卷共享数据时，可通过使用CNFS动态创建sharepath类型的共享存储卷。本文介绍如何使用CNFS动态创建NAS共享存储卷。

前提条件

- 已创建Kubernetes集群，且存储插件选择为CSI。具体操作，请参见创建Kubernetes托管版集群。
  - 若需要新建集群，选择CSI存储插件时请同时选中**创建默认NAS文件系统和CNFS容器网络文件系统动态存储类型**。
  - 若已创建集群，但未选中**创建默认NAS文件系统和CNFS容器网络文件系统动态存储类型**，请使用CNFS托管NAS文件系统。具体操作，请参见使用CNFS托管NAS文件系统。
- csi-plugin和csi-provisioner组件版本不低于v1.20.5-ff6490f-aliyun。关于升级CSI-Plugin和CSI-Provisioner组件的操作，请参见升级CSI-Plugin和CSI-Provisioner。
- 已通过kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

操作步骤

您需要手动创建sharepath类型的storageclass并在containerNetworkFileSystem引用集群中CNFS的名称。

1. 执行以下命令查看CNFS的名称。

   ```
   kubectl get cnfs
   ```

   预期输出：

   ```
   NAME                                    AGE
   default-cnfs-nas-837d6ea-20210819155623  17d
   ```

2. 创建StorageClass。

i. 使用以下YAML示例创建*storageclass.yaml*。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alibabacloud-cnfs-nas-sharepath
mountOptions:
 - nolock,tcp,noresvport
 - vers=3
parameters:
 volumeAs: "sharepath"
 containerNetworkFileSystem: "default-cnfs-nas-837d6ea-20210819155623"
 path: "/sharepath"
provisioner: nasplugin.csi.alibabacloud.com
reclaimPolicy: Retain
allowVolumeExpansion: true
```

> **注意** NAS动态共享存储卷中volumeAs必须为*sharepath*，同时reclaimPolicy为*Retain*。如果reclaimPolicy为*Delete*，会导致动态创建的PVC为Pending状态。

ii. 执行以下命令创建StorageClass。

```
kubectl create -f storageclass.yaml
```

3. 创建PVC1及引用该PVC1的Deployment1。

i. 使用以下YAML模板创建*nas-sharepath1.yaml*。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-nas-sharepath1
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alibabacloud-cnfs-nas-sharepath
 resources:
  requests:
   storage: 10Gi
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas-sharepath1
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
       - containerPort: 80
      volumeMounts:
       - name: pvc-nas-sharepath1
         mountPath: "/data"
   volumes:
    - name: pvc-nas-sharepath1
      persistentVolumeClaim:
       claimName: pvc-nas-sharepath1
```

ii. 执行以下命令创建PVC1及引用该PVC1的Deployment1。

```
kubectl create -f nas-sharepath1.yaml
```

4. 在Deployment1的挂载目录下创建测试文件。

i. 执行以下命令查看Deployment1。

```
kubectl get pod
```

预期输出：

```
NAME                                READY  STATUS  RESTARTS  AGE
deployment-nas-sharepath1-586686b789-v4bwt  1/1   Running  0     6m
```

ii. 执行以下命令在Deployment1的*/data*目录下创建测试文件*test.txt*且写入 `hello world` 。

```
kubectl exec deployment-nas-sharepath1-586686b789-v4bwt -ti sh
cd /data
echo "hello world" > test.txt
```

5. 再次创建一个PVC2及引用该PVC2的Deployment2。

i. 使用以下YAML模板创建*nas-sharepath2.yaml*。

> **说明**
>
> - 建议PVC2的storageClassName参数与PVC1的storageClassName参数配置相同。本示例PVC1和PVC2的storageClassName参数都配置为 `alibabacloud-cnfs-nas-sharepath` 。
> - Deployment1与Deployment2的mountPath参数必须配置相同，即使用相同的NAS子目录实现共享。本示例为*/data*。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-nas-sharepath2
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alibabacloud-cnfs-nas-sharepath
 resources:
  requests:
   storage: 10Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas-sharepath2
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
       - containerPort: 80
      volumeMounts:
       - name: pvc-nas-sharepath2
         mountPath: "/data"
   volumes:
    - name: pvc-nas-sharepath2
      persistentVolumeClaim:
       claimName: pvc-nas-sharepath2
```

ii. 执行以下命令创建PVC2及引用该PVC2的Deployment2。

```
kubectl create -f nas-sharepath2.yaml
```

6. 在Deployment2的挂载目录下查看Deployment1的文件是否存在。

i. 执行以下命令查看Deployment2。

```
kubectl get pod
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE
deployment-nas-sharepath2-74b85f4c86-tj9xz  1/1    Running  0      6m
```

ii. 执行以下命令进入Deployment2的*/data*目录下。

```
kubectl exec deployment-nas-sharepath2-74b85f4c86-tj9xz -ti sh
cd /data
```

iii. 执行以下命令查看*/data*下的文件。

```
ls -arlth
```

预期输出：

```
total 8.5K
drwxr-xr-x 2 root root 4.0K Sep  6 03:56 .
-rw-r--r-- 1 root root   14 Sep  6 03:56 test.txt
drwxr-xr-x 1 root root 4.0K Sep  6 03:57 ..
```

从预期输出可得，Deployment2的*/data*目录下存在*test.txt*文件。

iv. 执行以下命令查看*test.txt*文件中的内容。

```
cat test.txt
```

预期输出：

```
hello world
```

从预期输出可得，Deployment2的*/data*目录下的*test.txt*文件内容与之前在Deployment1的*/data*目录下*test.txt*文件的写入内容一致。

7. 删除Deployment2、PVC2及其PV。

```
kubectl delete deployment/deployment-nas-sharepath2 pvc/pvc-nas-sharepath2 pv/nas-daf9f0f0-16cf-412c-9a81-ad8b53490293
```

预期输出：

```
deployment.apps "deployment-nas-sharepath2" deleted
persistentvolumeclaim "pvc-nas-sharepath2" deleted
persistentvolume "nas-daf9f0f0-16cf-412c-9a81-ad8b53490293" deleted
```

8. 在Deployment1的*/data*目录下，查看*test.txt*文件。

i. 执行以下命令进入Deployment1的*/data*目录下。

```
kubectl exec deployment-nas-sharepath1-586686b789-v4bwt -ti sh
cd /data
```

ii. 执行以下命令查看*/data*下的文件。

```
ls
```

预期输出：

```
test.txt
```

从预期输出可得，Deployment1的*/data*目录下存在*test.txt*文件。

iii. 执行以下命令查看*test.txt*文件中的内容。

```
cat test.txt
```

预期输出：

```
hello world
```

从预期输出及上述操作可得，当Deployment2及其PV/PVC删除时，并不影响共享子目录。

10.5.5. 使用CNFS的回收站功能恢复数据

容器网络文件系统CNFS支持文件回收站功能，避免文件误删除。本文以公网可访问的Nginx应用为例，介绍如何通过CNFS的回收站恢复被删除的文件。

背景信息

通过使用CNFS的StorageClass创建示例目录并绑定Nginx，然后在示例目录删除index.html文件，此时Nginx无法访问欢迎页。然后可通过控制台回收站功能恢复*index.html*文件，同时也恢复了Nginx欢迎页。

前提条件

- 已创建Kubernetes集群，且存储插件选择为CSI。具体操作，请参见创建Kubernetes托管版集群。
  - 若需要新建集群，选择CSI存储插件时请同时选中**创建默认NAS文件系统和CNFS容器网络文件系统动态存储类型**。
  - 若已创建集群，但未选中**创建默认NAS文件系统和CNFS容器网络文件系统动态存储类型**，请使用CNFS托管NAS文件系统。具体操作，请参见使用CNFS托管NAS文件系统。
- 集群已开启公网访问。
  - 关于创建集群时如何开启公网访问，请参见创建集群时绑定EIP。
  - 关于已有集群如何开启访问，请参见创建集群后绑定EIP。
- csi-plugin和csi-provisioner组件版本不低于v1.20.5-ff6490f-aliyun。关于如何升级CSI-Plugin和CSI-Provisioner组件，请参见升级CSI-Plugin和CSI-Provisioner。
- storage-operator组件版本不低于v1.18.8.56-2aa33ba-aliyun。关于如何升级storage-operator组件，请参见管理组件。

步骤一：创建PVC并绑定到公网可访问的Nginx

1. 查看CNFS对象的状态。

   需要确保CNFS对象的状态为**Available**。

   i. 执行以下命令查看CNFS对象。

   ```
   kubectl get cnfs
   ```

   预期输出：

   ```
   NAME                    AGE
   default-cnfs-nas-7938cef-20210907193713  21h
   ```

ii. 执行以下命令查看CNFS对象的状态。

```
kubectl get cnfs default-cnfs-nas-7938cef-20210907193713 -o yaml | grep Available
```

预期输出：

```
status: Available
```

2. 使用以下YAML示例创建PVC。

> **注意** PVC对象的storageClassName引用CNFS的StorageClass。本例中PVC名称为**cnfs-nas-pvc**，storageClassName为**alibabacloud-cnfs-nas**。

```
cat << EOF | kubectl apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: cnfs-nas-pvc
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alibabacloud-cnfs-nas
 resources:
  requests:
   storage: 30Gi
EOF
```

3. 使用以下YAML示例创建Deployment。

> **注意** Deployment对象引用上个步骤创建的PVC。本示例Deployment名称为**cnfs-nas-deployment**，引用的PVC名称为**cnfs-nas-pvc**，映射到容器的路径为*/app*，并开启名称为 `http` 的容器端口8080。

```
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
 name: cnfs-nas-deployment
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   securityContext:
    runAsUser: 0
   containers:
   - name: nginx
    image: docker.io/bitnami/nginx:1.16.1-debian-9-r56
    volumeMounts:
    - mountPath: "/app"
     name: cnfs-nas-pvc
    ports:
    - containerPort: 8080
     name: http
   volumes:
   - name: cnfs-nas-pvc
    persistentVolumeClaim:
     claimName: cnfs-nas-pvc
EOF
```

4. 使用以下YAML示例为Deployment对象创建Service。

> **说明** 创建Loadbalancer类型的Service，SLB会暴露公网IP，可通过SLB的公网IP访问Deployment。本示例创建名为**nginx-default**的Service，类型为LoadBalancer，该Service将公网暴露的HTTP请求转发到label为app:nginx的Pod上。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
 name: nginx-default
 labels:
  app: nginx
spec:
 type: LoadBalancer
 externalTrafficPolicy: "Cluster"
 ports:
  - name: http
   port: 80
   targetPort: http
 selector:
  app: nginx
EOF
```

5. 将Nginx的欢迎页加入到启动状态的Pod的*/app*路径下。
   i. 执行以下命令查看Pod。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME                READY  STATUS   RESTARTS  AGE
   cnfs-nas-deployment-597bc9fb45-cmkss  1/1    Running  0      3h23m
   ```

   ii. 执行以下命令进入Pod `cnfs-nas-deployment-597bc9fb45-cmkss` 的*/app*路径。

   ```
   kubectl exec cnfs-nas-deployment-597bc9fb45-cmkss -ti sh
   cd /app
   ```

   iii. 执行以下命令将*index.html*写入到*/app*路径下。

   ```
   cat << EOF >> index.html
   <!DOCTYPE html>
   <html>
   <head>
   <title>Welcome to nginx!</title>
   <style>
   html { color-scheme: light dark; }
   body { width: 35em; margin: 0 auto;
   font-family: Tahoma, Verdana, Arial, sans-serif; }
   </style>
   </head>
   <body>
   <h1>Welcome to nginx!</h1>
   <p>If you see this page, the nginx web server is successfully installed and
   working. Further configuration is required.</p>
   <p>For online documentation and support please refer to
   <a href="http://nginx.org/">nginx.org</a>.<br/>
   Commercial support is available at
   <a href="http://nginx.com/">nginx.com</a>.</p>
   <p><em>Thank you for using nginx.</em></p>
   </body>
   </html>
   EOF
   ```

   iv. 执行以下命令退出Pod。

   ```
   exit
   ```

6. 执行以下命令获取SLB的公网IP。

   ```
   kubectl get svc
   ```

   预期输出：

   ```
   NAME      TYPE      CLUSTER-IP     EXTERNAL-IP    PORT(S)     AGE
   nginx-default  LoadBalancer  192.168.XX.XX   47.115.XX.XX   80:30989/TCP  20h
   ```

7. 在浏览器地址栏输入公网IP，可以看到Nginx的欢迎页。

   本示例中公网IP为 `47.115.XX.XX` 。



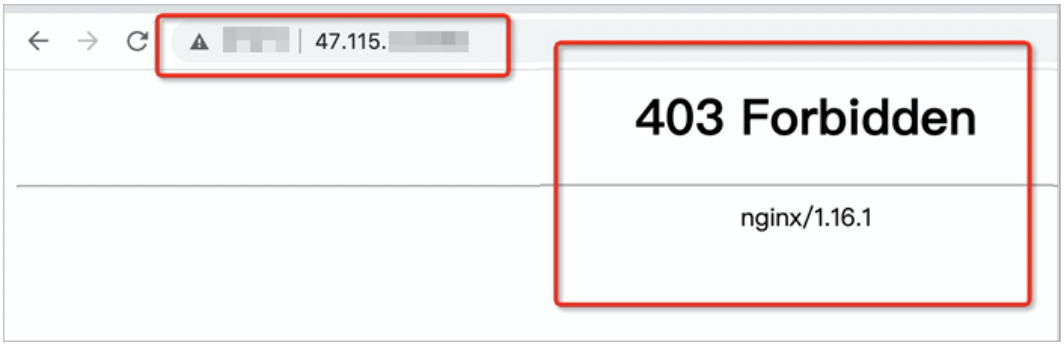
步骤二：验证CNFS回收站的恢复功能

以下示例介绍在删除文件的场景下，如何通过CNFS托管的NAS存储卷的回收站恢复文件。

1. 执行以下命令删除*index.html*文件。

   ```
   kubectl exec cnfs-nas-deployment-597bc9fb45-cmkss -- rm -rf /app/index.html
   ```

   刷新浏览器再次查看Nginx的欢迎页，可以看到页面提示 `403 Forbidden` 。

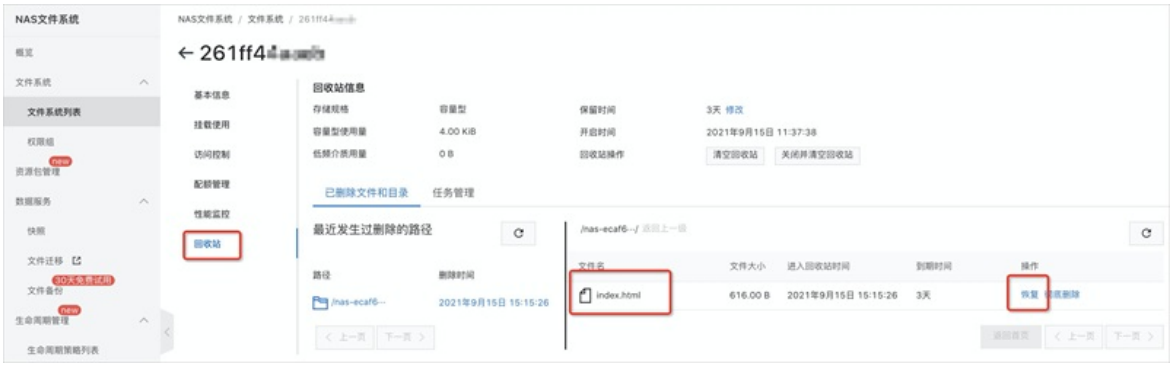


2. 登录容器服务管理控制台。
3. 在控制台左侧导航栏中，单击**集群**。
4. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
5. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。
6. 在**存储卷**页面，在目标存储卷右侧**操作**列单击**回收站**。

   CNFS托管的NAS默认开启回收站，可以在控制台恢复Nginx的*index.html*文件。
   本示例的PV名称为**nas-ecaf6018-5250-4e19-b570-5d9e657d23bc**。



7. 在目标文件系统**回收站**页签，单击**已删除文件和目录**页签，在目标删除路径的右侧**操作**列单击**恢复**。



选择**恢复至原路径**，并单击**确定**。



8. 刷新浏览器可以看到Nginx的欢迎页已恢复正常。

   表示Nginx的*index.html*文件已恢复。



10.6. NAS存储卷

10.6.1. NAS存储卷概述

您可以在容器服务Kubernetes集群中使用阿里云NAS存储卷。本文介绍NAS存储卷的功能介绍、存储规格、适用场景、使用限制及计费说明。

功能介绍

阿里云文件存储NAS（Apsara File Storage）是面向阿里云ECS实例、E-HPC和容器服务等计算节点的文件存储服务。它是一种可共享访问、弹性扩展、高可靠以及高性能的分布式文件系统。

NAS基于POSIX文件接口，天然适配原生操作系统，提供共享访问，同时保证数据一致性和锁互斥。它提供了简单的可扩展文件存储以供与ECS配合使用，多个ECS实例可以同时访问NAS文件系统，并且存储容量会随着您添加和删除文件而自动弹性增长和收缩，为在多个实例或服务器上运行产生的工作负载和应用程序提供通用数据源。

存储规格

NAS提供了通用容量型、通用性能型以及极速型存储类型。更多信息，请参见规格类型。

适用场景

根据业务需求，您可以对NAS存储卷做以下操作：

业务需求	参考链接
存储应用数据	具体操作，请参见以下文档： • 使用NAS静态存储卷 • 使用NAS动态存储卷
在资源分配的基础上通过资源限制提升整体资源的利用率	具体操作，请参见配置NAS存储卷容量。
托管NAS文件系统	简化存储卷声明的方式，减少NAS控制台与ACK控制台之间多次切换挂载NAS存储卷，从而解决容易挂载失败且操作冗余的问题。具体操作，请参见使用CNFS托管NAS文件系统。
NAS存储卷自动扩容	通过定义扩容策略，在NAS存储卷的使用率高于某个阈值时触发自动扩容。具体操作，请参见使用CNFS自动扩容NAS存储卷。
动态创建NAS共享存储卷	阿里云CSI支持创建多个PV，且多个PV可以共享同一个NAS的子目录。具体操作，请参见使用CNFS动态创建NAS共享存储卷。

使用限制

- 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无响应。
- NAS挂载点创建后，等待一定时间，待挂载点**状态**为**可用**的后方可使用。
- 数据卷挂载协议推荐使用NFSv3。
- 使用NAS数据卷前，建议将CSI存储插件升级到最新版本。
- 通用NAS与极速NAS在挂载连通性、文件系统数量及协议类型等方面存在相应约束条件。更多信息，请参见使用限制。

计费说明

关于NAS的计费说明，请参见NAS计费说明。

10.6.2. 使用NAS静态存储卷

NAS存储卷是一种可共享访问、弹性扩展、高可靠以及高性能的分布式文件系统。本文介绍如何使用阿里云NAS静态存储卷，及如何实现持久化存储与共享存储。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建NAS文件系统。具体操作，请参见创建文件系统。
  若需要加密NAS存储卷中的数据，创建NAS文件系统时请配置加密类型。
- 已创建NAS挂载点。具体操作，请参见管理挂载点。
  NAS挂载点需要和集群节点在同一个VPC内。

- 已使用kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

使用场景

- 对磁盘I/O要求较高的应用。
- 读写性能相对于对象存储OSS高。
- 可实现跨主机文件共享，例如可作为文件服务器。

注意事项

- 在使用极速NAS文件系统时，配置数据卷的 `path` 需要以*/share*为父目录。例如，Pod挂载的NAS文件系统子目录可配置为*/share/path1*。
- NAS支持同时被多个Pod挂载，此时多个Pod可能同时修改相同数据，需要应用自行实现数据的同步。

  > **说明** NAS存储的/目录不支持修改权限、属主和属组。

- 若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间延长。

  > **说明** 若已配置securityContext.fsgroup参数，且需要减少挂载时间。具体操作，请参见NAS存储卷挂载时间延长。

通过控制台的方式使用NAS静态存储卷

步骤一：创建PV

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。
5. 在**存储卷**页面单击右上角的**创建**。
6. 在**创建存储卷**对话框中配置参数。

参数	说明
存储卷类型	支持云盘、NAS、OSS三种云存储类型。本文中选择为NAS。
名称	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。
存储驱动	支持Flexvolume和CSI。本文中选择为CSI。
总量	所创建存储卷的容量。注意NAS文件系统本身不限制使用量。此处不是NAS文件系统的使用限额，只是所创建存储卷的容量声明。
访问模式	支持ReadWriteMany和ReadWriteOnce。默认为ReadWriteMany。
挂载点域名	您可以通过**选择挂载点**或者**自定义**的方式定义集群在NAS文件系统中挂载点的挂载地址。

参数	说明
显示高级选项	○ **子目录**：NAS路径下的子目录，以/为根目录，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*为父目录。 ○ **版本**：所创建存储卷的版本。
标签	为该存储卷添加标签。

7. 参数配置完成后，单击**创建**。

步骤二：创建PVC

1. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
2. 在**存储声明**页面，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。

参数	说明
存储声明类型	支持云盘、NAS、OSS三种云存储类型。 本文中选择NAS。
名称	创建的存储声明名称在集群内必须唯一。
分配模式	选择已有存储卷。 **说明** 若未创建存储卷，您可以设置**分配模式**为**创建存储卷**，配置创建存储卷参数。更多信息，请参见创建PV。
已有存储卷	单击**选择已有存储卷**，在目标存储卷右侧操作列单击**选择**，选择存储卷。
总量	所创建存储卷的容量。 **说明** 所创建存储卷声明的容量不能超过待挂载的存储卷容量。

4. 单击**创建**。

   创建成功后可以在列表中看到创建的存储声明，并且已绑定相应的存储卷。

步骤三：创建应用

1. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
2. 在**无状态**页面中，单击**使用镜像创建**。
3. 配置创建应用的参数信息。

   以下主要为您介绍数据卷的配置。关于其他参数的描述，请参见创建无状态工作负载Deployment。
   ACK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，请参见Volumes。

   ○ **云存储**：支持云存储类型。

本例中配置了一个NAS类型的数据卷，将该NAS存储卷挂载到容器中*/tmp*路径下。



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。

通过kubectl命令行方式使用NAS静态存储卷

1. 执行以下命令创建静态PV。

   ```
   kubectl create -f pv-nas.yaml
   ```

   以下为创建静态卷PV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: "2564f4****-ysu87.cn-shenzhen.nas.aliyuncs.com"
      path: "/csi"
    mountOptions:
    - nolock,tcp,noresvport
    - vers=3
   ```

参数	说明
name	PV的名称。
labels	设置PV的标签。
storage	NAS的可使用量。
accessModes	配置访问模式。
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。

参数	说明
volumeHandle	配置PV的唯一标识符。若需要同时使用多个PV，则各个PV中该值必须不一致。
server	NAS挂载点。
path	挂载子目录，极速NAS需要以*/share*为父目录。
vers	挂载NAS数据卷的NFS协议版本号，推荐使用v3，极速类型NAS只支持v3。

2. 执行以下命令创建静态PVC。

   创建NAS存储声明PVC，使用selector筛选PV，精确配置PVC和PV的绑定关系。

   ```
   kubectl create -f pvc-nas.yaml
   ```

   以下为创建静态卷PVC的YAML示例文件。

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
    name: pvc-nas
   spec:
    accessModes:
     - ReadWriteMany
    resources:
     requests:
       storage: 5Gi
    selector:
     matchLabels:
       alicloud-pvname: pv-nas
   ```

参数	说明
name	PVC的名称。
accessModes	配置访问模式。
storage	声明应用使用量，不能大于存储卷的总量。
mathLabels	输入PV的标签，用于关联PV。

3. 执行以下命令创建名为nas-static的应用，并挂载PVC。

   ```
   kubectl create -f nas.yaml
   ```

   以下为创建nas-static应用的*nas.yaml*示例文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nas-static
 labels:
  app: nginx
spec:
 replicas: 2
 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: pvc-nas
```

参数	说明
mountPath	NAS在容器中挂载的位置。
claimName	PVC的名称，用于绑定PVC。

4. 执行以下命令，查看Pod信息。

```
kubectl get pod
```

预期输出：

```
NAME              READY  STATUS  RESTARTS  AGE
nas-static-5b5cdb85f6-n****  1/1   Running  0     32s
nas-static-c5bb4746c-4****   1/1   Running  0     32s
```

验证NAS的持久化存储

1. 查看部署应用和NAS文件。
   i. 执行以下命令，查看部署的应用名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME              READY  STATUS  RESTARTS  AGE
   nas-static-5b5cdb85f6-n****  1/1   Running  0     32s
   nas-static-c5bb4746c-4****   1/1   Running  0     32s
   ```

ii. 执行以下命令，查看任意一个应用的*/data*路径下的文件，本文以名为 `nas-static-5b5cdb85f6-n****` 的Pod为例。

```
kubectl exec nas-static-5b5cdb85f6-n**** ls /data
```

无返回结果，说明*/data*路径下无文件。

2. 执行以下命令，在名为 `nas-static-5b5cdb85f6-n****` Pod的*/data*路径下创建文件*nas*。

```
kubectl exec nas-static-5b5cdb85f6-n**** touch /data/nas
```

3. 执行以下命令，查看名为 `nas-static-5b5cdb85f6-n****` Pod的*/data*路径下的文件。

```
kubectl exec nas-static-5b5cdb85f6-n**** ls /data
```

预期输出：

```
nas
```

4. 执行以下命令，删除Pod。

```
kubectl delete pod nas-static-5b5cdb85f6-n****
```

5. 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
```

6. 验证删除Pod后，NAS里创建的文件是否还存在。

i. 执行以下命令，查看Kubernetes重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME              READY  STATUS   RESTARTS  AGE
nas-static-5b5cdb85f6-n****  1/1   Running  0     32s
nas-static-c5bb4746c-4****   1/1   Running  0     32s
```

ii. 执行以下命令，查看名为 `nas-static-5b5cdb85f6-n****` 的Pod */data*路径下的文件。

```
kubectl exec nas-static-5b5cdb85f6-n**** ls /data
```

预期输出：

```
nas
```

*nas*文件仍然存在，说明NAS的数据可持久化保存。

验证NAS的共享存储

1. 查看部署的应用所在的Pod和NAS文件。

i. 执行以下命令，查看应用所在Pod的名称。

```
kubectl get pod
```

预期输出：

```
NAME           READY  STATUS   RESTARTS  AGE
nas-static-5b5cdb85f6-n****  1/1   Running  0     32s
nas-static-c5bb4746c-4****   1/1   Running  0     32s
```

ii. 执行以下命令，查看2个Pod */data*路径下的文件。

```
kubectl exec nas-static-5b5cdb85f6-n**** ls /data
kubectl exec nas-static-c5bb4746c-4**** ls /data
```

2. 执行以下命令，在任意一个Pod的*/data*路径下创建文件*nas*。

```
kubectl exec nas-static-5b5cdb85f6-n**** touch /data/nas
```

3. 执行以下命令，查看2个Pod */data*路径下的文件。

i. 执行以下命令，查看名为 `nas-static-5b5cdb85f6-n****` 的Pod */data*路径下的文件。

```
kubectl exec nas-static-5b5cdb85f6-n**** ls /data
```

预期输出：

```
nas
```

ii. 执行以下命令，查看名为 `nas-static-c5bb4746c-4****` 的Pod */data*路径下的文件。

```
kubectl exec nas-static-c5bb4746c-4**** ls /data
```

预期输出：

```
nas
```

如果在任意一个Pod的*/data*下创建的文件，两个Pod下的*/data*路径下均存在此文件，则说明两个Pod共享一个NAS。

10.6.3. 使用NAS动态存储卷

阿里云Kubernetes CSI支持两种类型的NAS动态存储卷挂载：subpath方式和filesystem方式。本文介绍如何使用阿里云NAS动态存储卷，及如何验证NAS存储卷的持久化存储与共享存储特性。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建NAS文件系统。请参见创建文件系统。
  若需要加密NAS存储卷中的数据，创建NAS文件系统时请配置加密类型。
- 已创建NAS挂载点。请参见管理挂载点。
  NAS挂载点需要和集群节点在同一个VPC内。

使用场景

- 对磁盘I/O要求较高的应用。
- 读写性能相对于对象存储OSS高。
- 可实现跨主机文件共享，例如可作为文件服务器。

注意事项

- 在使用极速NAS文件系统时，配置动态存储卷StorageClass中的 `path` 需要以*/share*为父目录。例如，`0cd8b4a576-g****.cn-hangzhou.nas.aliyuncs.com:/share/subpath` 表示Pod挂载的NAS文件系统子目录为 `/share/subpath` 。
- NAS支持同时被多个Pod挂载，此时多个Pod可能同时修改相同数据，需要应用自行实现数据的同步。

> **说明** NAS存储的/目录不支持修改权限、属主和属组。

- 若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执

行 `chmod` 或 `chown` 操作，导致挂载时间延长。

> **说明** 若已配置securityContext.fsgroup参数，且需要减少挂载时间。具体操作，请参见NAS存储卷挂载时间延长。

通过控制台的方式使用NAS动态存储卷

通过控制台的方式只能创建subpath类型的NAS动态存储卷，若您需要使用filesystem类型的NAS动态存储卷，请使用kubectl命令行方式。

步骤一：创建StorageClass

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 登录容器服务管理控制台。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储类**。
5. 在**存储类**页面，单击右上角的**创建**。
6. 在**创建**对话框，配置StorageClass的相关参数。

   部分参数的说明如下所示：

参数	说明
名称	StorageClass的名称。 名称必须以小写字母开头，只能包含小写字母、数字、小数点（.）和短划线（-）。
存储卷类型	可选择**云盘**或**NAS**。本示例选择**NAS**。
存储驱动	默认为**CSI**。
回收策略	回收策略，默认为Delete，支持Retain。 ○ Delete模式：删除PVC的时候，PV和NAS文件系统会一起删除。 ○ Retain模式：删除PVC的时候，PV和NAS文件系统不会被删除，需要您手动删除。 如果数据安全性要求高，推荐使用Retain方式以免误删数据。
挂载选项	挂载NAS的可选参数，包括NFS协议版本等参数。
挂载点域名	NAS文件系统的挂载点地址。 若无可选的挂载点地址，请先创建NAS文件系统.。具体操作，请参见使用CNFS托管NAS文件系统。
路径	NAS文件系统中的挂载路径。

7. 参数配置完成后，单击**创建**。
   创建成功后在**存储类**列表中可看到刚创建的StorageClass。

步骤二：创建PVC

1. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
2. 在**存储声明**页面，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。

参数	说明
存储声明类型	支持云盘、NAS、OSS三种云存储类型。 本文中选择NAS。
名称	创建的存储声明名称在集群内必须唯一。
分配模式	本文中选择**使用存储类动态创建**。
已有存储类	单击**选择存储类**，在**选择存储类**对话框目标存储类右侧**操作**列单击**选择**。
总量	所创建存储卷的容量。
访问模式	默认为ReadWriteMany，也可选择ReadWriteOnce。

4. 单击**创建**。

   创建成功后可在列表中看到创建的存储声明，并且已绑定相应的存储卷。

步骤三：创建应用

1. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
2. 在**无状态**页面中，单击**使用镜像创建**。
3. 配置创建应用的参数信息。

   以下主要为您介绍数据卷的配置。关于其他参数的描述，请参见创建无状态工作负载Deployment。
   ACK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，请参见Volumes。
   - **云存储**：支持云存储类型。

   本例中配置了一个NAS类型的数据卷，将该NAS存储卷挂载到容器中*/tmp*路径下。



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。

通过kubectl命令行方式使用subpath类型的NAS动态存储卷

当您的多个Kubernetes应用或者Pod需要挂载相同的NAS存储卷共享数据时，或不同的Pod挂载相同NAS文件系统的不同子目录时， 可以使用subpath类型的NAS动态存储卷方式。

NAS动态存储卷的挂载方式为subpath类型时，您需要手动创建NAS文件系统和挂载点。

1. 创建NAS文件系统和挂载点。
   i. 登录NAS控制台。
   ii. 创建NAS文件系统。请参见创建文件系统。
   iii. 添加挂载点。请参见管理挂载点。

2. 创建StorageClass。

   i. 创建并复制以下内容到*alicloud-nas-subpath.yaml*文件中。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-nas-subpath
mountOptions:
- nolock,tcp,noresvport
- vers=3
parameters:
 volumeAs: subpath
 server: "0cd8b4a576-g****.cn-hangzhou.nas.aliyuncs.com:/k8s/"
provisioner: nasplugin.csi.alibabacloud.com
reclaimPolicy: Retain
```

参数	描述
mountOptions	挂载NAS的options参数在mountOptions中配置，包括NFS协议版本。
volumeAs	可选subpath、filesystem，分别表示创建子目录类型的PV和文件系统类型PV。
server	表示创建子目录类型的PV时，NAS文件系统的挂载点地址。
provisioner	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。
reclaimPolicy	PV的回收策略，默认为Delete，支持Retain。 ■ Delete模式：删除PVC的时候，PV和NAS文件系统会一起删除。 ■ Retain模式：删除PVC的时候，PV和NAS文件系统不会被删除，需要您手动删除。 如果数据安全性要求高，推荐使用Retain方式以免误删数据。
archiveOnDelete	表示在reclaimPolicy为Delete时，是否删除后端存储。因为NAS为共享存储，添加此选项进行双重确认。在参数parameters下配置。默认为true，表示不会真正删除目录或文件，而是将其Rename，格式为： `archived-{pvName}.{timestamp}` ；如果是配置为false，表示会真正删除后端对应的存储资源。 **说明** 在业务流量非常大时，不适合配置为false。更多信息，请参见使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV。

   ii. 执行以下命令创建StorageClass。

```
kubectl create -f alicloud-nas-subpath.yaml
```

3. 执行以下命令创建PVC。

i. 创建并复制以下内容到*pvc.yaml*文件中。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: nas-csi-pvc
spec:
 accessModes:
 - ReadWriteMany
 storageClassName: alicloud-nas-subpath
 resources:
  requests:
   storage: 20Gi
```

参数	说明
name	PVC的名称。
accessModes	配置访问模式。
storageClassName	StorageClass的名称，用于绑定StorageClass。
storage	声明应用存储使用量。

ii. 执行以下命令创建PVC。

```
kubectl create -f pvc.yaml
```

4. 执行以下命令创建应用。

   创建应用**nginx-1**和**nginx-2**共享NAS存储卷的同一个子目录。

i. 创建并复制以下内容到*nginx-1.yaml*文件中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas-1
 labels:
  app: nginx-1
spec:
 selector:
  matchLabels:
   app: nginx-1
 template:
  metadata:
   labels:
    app: nginx-1
  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-csi-pvc
```

- `mountPath`：NAS在容器中挂载的位置。
- `claimName`：PVC的名称，用于绑定PVC。本例中为**nas-csi-pvc**。

ii. 创建并复制以下内容到*nginx-2.yaml*文件中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas-2
 labels:
  app: nginx-2
spec:
 selector:
  matchLabels:
   app: nginx-2
 template:
  metadata:
   labels:
    app: nginx-2
  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-csi-pvc
```

- `mountPath`：NAS在容器中挂载的位置。本例为*/data*。
- `claimName`：输入与**nginx-1**应用相同的PVC名称，本例为**nas-csi-pvc**。

iii. 执行以下命令创建应用**nginx-1**和**nginx-2**。

```
kubectl create -f nginx-1.yaml -f nginx-2.yaml
```

5. 执行以下命令查看Pod信息。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE
deployment-nas-1-5b5cdb85f6-n****  1/1   Running  0     32s
deployment-nas-2-c5bb4746c-4****   1/1   Running  0     32s
```

说明 NAS存储卷的`0cd8b4a576-g****.cn-hangzhou.nas.aliyuncs.com:/share/nas-79438493-f3e0-11e9-bbe5-00163e09****`会同时挂载到`deployment-nas-1-5b5cdb85f6-n****`和`deployment-nas-2-c5bb4746c-4****`的*/data*目录下。其中：

- `/share`：StorageClass中指定的subpath。
- `nas-79438493-f3e0-11e9-bbe5-00163e09****`：PV的名称。

如果您需要为不同的Pod挂载同一个NAS文件系统的不同子目录，则需要分别创建**pvc-1**和**nginx-1**以及**pvc-2**和**nginx-2**。

通过kubectl命令行方式使用filesystem类型的NAS动态存储卷

> **注意** filesystem类型的NAS动态卷在删除时默认保留文件系统和挂载点，若需要在释放PV资源的同时释放NAS文件系统和挂载点，则需要同时设置StorageClass中的reclaimPolicy为Delete且deleteVolume的值为true。

当您的Kubernetes应用需要动态创建和删除NAS文件系统和挂载点时，可以使用filesystem类型。

使用filesystem类型NAS存储卷的Pod只能创建一个文件系统和一个挂载点，多个Pod之间无法共享一个存储卷。操作步骤如下。

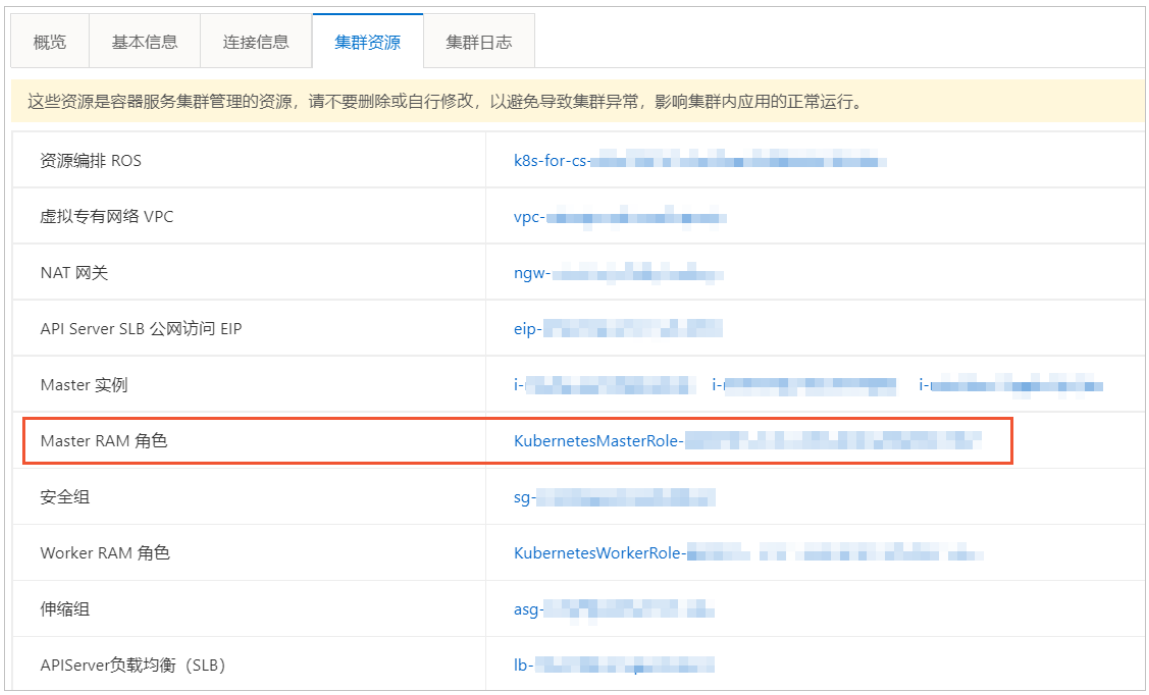
1. RAM Policy设置和授予。

   filesystem类型的NAS存储卷涉及NAS文件系统和挂载点的动态创建与删除，需要授予csi-provisioner相应的权限，RAM Policy的最小集合如下。

```
{
  "Action": [
    "nas:DescribeMountTargets",
    "nas:CreateMountTarget",
    "nas:DeleteFileSystem",
    "nas:DeleteMountTarget",
    "nas:CreateFileSystem"
  ],
  "Resource": [
    "*"
  ],
    "Effect": "Allow"
}
```

   您可以通过以下任意一种方式进行授权：

   - 编辑Kubernetes集群的Master RAM角色中的自定义策略内容，添加以上NAS相关的权限设置。请参见容器服务默认角色。



> **说明** 托管集群是自动添加Master RAM，专有集群则需要加Master RAM。

- 创建RAM用户授权以上RAM Policy并生成AccessKey，配置到csi-provisioner的 `env` 变量中。请参见容器服务默认角色。

```
env:
  - name: CSI_ENDPOINT
    value: unix://socketDir/csi.sock
  - name: ACCESS_KEY_ID
    value: ""
  - name: ACCESS_KEY_SECRET
    value: ""
```

2. 创建StorageClass。

i. 创建并复制以下内容到*alicloud-nas-fs.yaml*文件中。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-nas-fs
mountOptions:
- nolock,tcp,noresvport
- vers=3
parameters:
 volumeAs: filesystem
 storageType: Performance
 zoneId: cn-hangzhou-a
 vpcId: "vpc-2ze9c51qb5kp1nfqu****"
 vSwitchId: "vsw-gw8tk6gecif0eu9ky****"
 accessGroupName: DEFAULT_VPC_GROUP_NAME
 deleteVolume: "false"
provisioner: nasplugin.csi.alibabacloud.com
reclaimPolicy: Retain
```

参数	描述
volumeAs	定义创建数据的类型，可选filesystem和subpath两种类型： ■ filesystem表示Provisioner自动创建NAS文件系统，一个PV对应一个NAS文件系统。 ■ subpath类型表示一个PV对应一个NAS文件系统的子目录，Provisioner自动创建NAS文件系统子目录。
storageType	定义创建NAS文件系统的类型，可选**Performance**及**Capacity**两种类型，分别表示性能型、容量型。默认为性能型。
zoneId	定义创建NAS文件系统所在可用区。
vpcId	定义创建NAS文件系统对应挂载点所在VPC。
vSwitchId	定义创建NAS文件系统对应挂载点所在vSwitch ID。
accessGroupName	定义创建NAS文件系统对应挂载点所用的AccessGroup。默认为DEFAULT_VPC_GROUP_NAME。
deleteVolume	定义数据卷删除时处理NAS文件系统策略，由于NAS为共享文件系统，安全起见需要同时配置。
provisioner	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。
reclaimPolicy	PV的回收策略。当值为Delete且deleteVolume为true才会在删除PVC的时候把NAS文件系统删除。

ii. 执行以下命令创建StorageClass。

```
kubectl create -f alicloud-nas-fs.yaml
```

3. 创建PVC和Pod挂载NAS存储卷。

i. 创建并复制以下内容到*pvc.yaml*文件中。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: nas-csi-pvc-fs
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alicloud-nas-fs
 resources:
  requests:
   storage: 20Gi
```

ii. 创建并复制以下内容到*nginx.yaml*文件中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas-fs
 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-csi-pvc-fs
```

iii. 执行以下命令创建PVC和Pod。

```
kubectl create -f pvc.yaml -f nginx.yaml
```

这种场景下，CSI会在PVC创建时动态新建NAS文件系统和挂载点，PVC删除时动态删除挂载点和文件系统。

验证NAS的持久化存储

NAS提供了持久化存储服务，当某个Pod删除时，重新部署的Pod将自动同步之前Pod的所有数据。

根据以下示例验证NAS的持久化存储特性：

1. 查看部署应用的Pod和NAS文件。

i. 执行以下命令，查看部署的应用所在Pod的名称。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS  RESTARTS  AGE
deployment-nas-1-5b5cdb85f6-n****  1/1   Running  0     32s
deployment-nas-2-c5bb4746c-4****  1/1   Running  0     32s
```

ii. 执行以下命令，查看任意一个Pod*/data*路径下的文件，本文以名为 `deployment-nas-1-5b5cdb85f6-n****` 的Pod为例。

```
kubectl exec deployment-nas-1-5b5cdb85f6-n**** -- ls /data
```

无返回结果，说明*/data*路径下无文件。

2. 执行以下命令，在名为 `deployment-nas-1-5b5cdb85f6-n****` 的Pod*/data*路径下创建文件*nas*。

```
kubectl exec deployment-nas-1-5b5cdb85f6-n**** -- touch /data/nas
```

3. 执行以下命令，查看名为 `deployment-nas-1-5b5cdb85f6-n****` 的Pod*/data*路径下的文件。

```
kubectl exec deployment-nas-1-5b5cdb85f6-n**** -- ls /data
```

预期输出：

```
nas
```

4. 执行以下命令，删除Pod。

```
kubectl delete pod deployment-nas-1-5b5cdb85f6-n****
```

5. 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
```

6. 验证删除Pod后，NAS里创建的文件是否还存在。

i. 执行以下命令，查看Kubernetes重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS  RESTARTS  AGE
deployment-nas-1-5b5cdm2g5-m****  1/1   Running  0     32s
deployment-nas-2-c5bb4746c-4****  1/1   Running  0     32s
```

ii. 执行以下命令，查看名为 `deployment-nas-1-5b5cdm2g5-m****` 的Pod*/data*路径下的文件。

```
kubectl exec deployment-nas-1-5b5cdm2g5-m**** -- ls /data
```

预期输出：

```
nas
```

*nas*文件仍然存在，说明NAS的数据可持久化保存。

验证NAS存储卷的共享存储

NAS存储卷支持同时被多个Pod挂载，当某个Pod修改数据时，其余Pod将自行实现数据的同步。

根据以下示例验证NAS存储卷的共享存储特性：

1. 查看部署的应用所在的Pod和NAS文件。
   i. 执行以下命令，查看应用所在Pod的名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME                READY  STATUS   RESTARTS  AGE
   deployment-nas-1-5b5cdb85f6-n****  1/1   Running  0     32s
   deployment-nas-2-c5bb4746c-4****   1/1   Running  0     32s
   ```

   ii. 执行以下命令，查看2个Pod*/data*路径下的文件。

   ```
   kubectl exec deployment-nas-1-5b5cdb85f6-n**** -- ls /data
   kubectl exec deployment-nas-2-c5bb4746c-4**** -- ls /data
   ```

2. 执行以下命令，在任意一个Pod的*/data*路径下创建文件*nas*。

   ```
   kubectl exec deployment-nas-1-5b5cdb85f6-n**** -- touch /data/nas
   ```

3. 执行以下命令，查看2个Pod*/data*路径下的文件。
   i. 执行以下命令，查看名为 `deployment-nas-1-5b5cdb85f6-n****` 的Pod*/data*路径下的文件。

   ```
   kubectl exec deployment-nas-1-5b5cdb85f6-n**** -- ls /data
   ```

   预期输出：

   ```
   nas
   ```

   ii. 执行以下命令，查看名为 `deployment-nas-2-c5bb4746c-4****` 的Pod*/data*路径下的文件。

   ```
   kubectl exec deployment-nas-2-c5bb4746c-4**** -- ls /data
   ```

   预期输出：

   ```
   nas
   ```

   在任意一个Pod的*/data*下创建的文件，两个Pod下的*/data*路径下均存在此文件，说明两个Pod共享一个NAS。

10.6.4. 配置NAS存储卷容量

配额限制可以确保您在资源分配的基础上通过资源限制提升整体资源的利用率。阿里云Kubernetes CSI支持对NAS卷的子目录进行配额限制。本文介绍如何对NAS卷子目录进行配额限制。

前提条件

- csi-plugin的镜像版本不低于v1.18.8.45。关于csi-plugin的版本说明，请参见csi-plugin。
- NAS卷需要使用子目录进行挂载。

背景信息

关于NAS卷的管理配额，请参见目录配额。

使用限制

- 目前仅容量型NAS支持使用配额限制，其他NAS类型均不支持。关于NAS卷的规格类型，请参见NAS规格类型。
- 仅支持子目录挂载方式设置配额。
- 除呼和浩特和乌兰察布外其他地域均已支持配额限制功能。
- 对于单个文件系统，最多只能对10个目录设置配额。

- 设置限制型配额后，如果文件使用量超过限制会导致增加文件长度，创建文件、目录和特殊文件，移动文件到目录等写入操作失败，应用层会收到IOError。
- 由于限制型配额的高风险性，强烈建议您在业务关键路径上谨慎评估和测试验证后再配置限制型配额。
- NAS配额的设置为异步执行，因此限制型配额的生效和失效都有延迟，正常情况下需要等待5~15分钟。

使用示例

1. 创建带有配额子目录NAS的StorageClass。

   示例模板如下所示：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-nas-sp8
mountOptions:
  - nolock,tcp,noresvport
  - vers=3
parameters:
  volumeAs: subpath
  server: "xxx.cn-hangzhou.nas.aliyuncs.com:/"
  archiveOnDelete: "false"
  path: "/abc"
  volumeCapacity: "true"
provisioner: nasplugin.csi.alibabacloud.com
reclaimPolicy: Delete
allowVolumeExpansion: "true"
```

参数	描述
mountOptions	挂载NAS的options参数在mountOptions中配置，包括NFS协议版本。
volumeAs	可选subpath、filesystem，分别表示创建子目录类型的PV和文件系统类型PV。
server	表示创建子目录类型的PV时，NAS文件系统的挂载点地址。
archiveOnDelete	表示在reclaimPolicy为Delete时，是否删除后端存储。因为NAS为共享存储，添加此选项进行双重确认。默认为true。
path	表示挂载子目录，极速NAS需要以*/share*开头。
volumeCapacity	表示是否使用配额。可选值为：true，false。
provisioner	表示ACK动态卷控制器名称。
reclaimPolicy	表示PV的回收策略。可选值为： ○ Retain：保留后端存储，删除PV及PVC不会删除对应的后端存储，例如云盘。 ○ Delete：当删除PVC时，自动删除PV和后端的存储。
allowVolumeExpansion	表示是否支持NAS存储卷的扩容。

> **说明** 创建带有配额子目录NAS的StorageClass，需要将volumeCapacity设置为true。

2. 创建容量大小为20 GiB的PVC。

   示例模板如下所示：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-nas-dynamic-create-subpath8
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alicloud-nas-sp8
 resources:
  requests:
   storage: 20Gi
```

3. 创建Deployment，引用步骤2创建的PVC。

   示例模板如下所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas-dynamic-create8
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
       - containerPort: 80
      volumeMounts:
       - name: pvc-nas-dynamic-create-subpath8
         mountPath: "/data"
   volumes:
    - name: pvc-nas-dynamic-create-subpath8
      persistentVolumeClaim:
       claimName: pvc-nas-dynamic-create-subpath8
```

结果验证

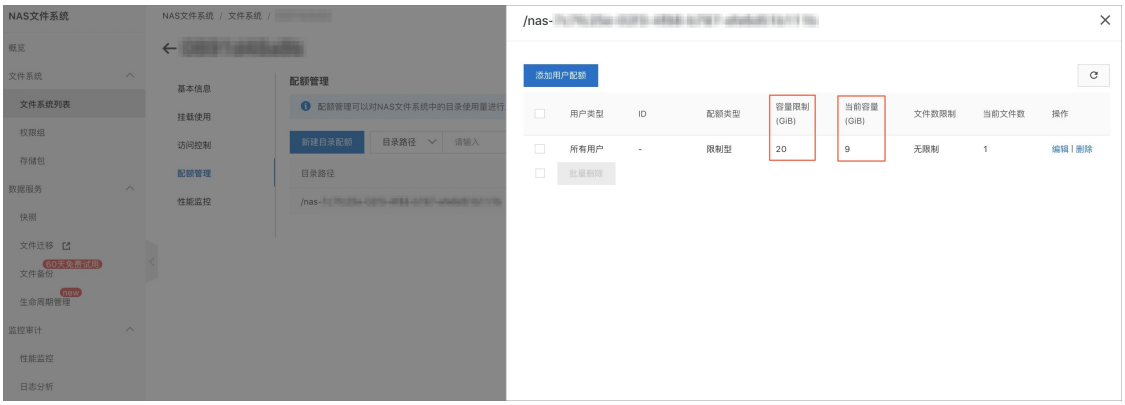
1. 执行以下命令向步骤3Deployment挂载的*/data*目录写入10 G数据。

```
dd if=/dev/zero of=10G.txt bs=1M count=10000
```

2. 等待5~15分钟后，查看子目录的配额详情。
   i. 登录NAS控制台。
   ii. 在控制台左侧导航栏中单击**文件系统 > 文件系统列表**。
   iii. 选择目标文件系统**操作**列下的**更多 > 配额管理**。

iv. 在目标文件系统的**配额管理**页面单击**操作**列的**管理配额**。

可以看到子目录的容量限制为20 GiB，当前容量为9 GiB。



当此子目录写满20 GiB，再写入数据时，会提示超过磁盘配额，即 `Disk quota exceeded` 。

```
sh-4.2# dd if=/dev/zero of=1G.txt bs=1M count=1000
dd: error writing '1G.txt': Disk quota exceeded
311+0 records in
310+0 records out
325058560 bytes (325 MB) copied, 1.01805 s, 319 MB/s
```

10.6.5. NAS存储卷FAQ

本文为您介绍NAS存储卷常见问题的处理方法。

- 使用NAS时，提示chown: Operation not permitted
- 使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV
- NAS存储卷挂载时间延长

使用NAS时，提示 `chown: Operation not permitted`

问题现象：

使用NAS时，提示 `chown: Operation not permitted` 。

问题原因：

您的容器没有权限使用该NAS。

解决方法：

使用Root权限启动容器。

使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV

问题现象：

使用NAS动态存储卷时，若创建子目录写入速度快于Controller删除子目录速度，将可能导致Controller的任务队列阻塞且无法创建新的PV。

问题原因：

当集群使用动态NAS存储卷时，配置的StorageClass回收策略reclaimPolicy为Delete且archiveOnDelete为false。

解决方法：

将archiveOnDelete配置为true，当删除PV时只是修改NAS文件系统中子目录的名称，而不是真正删除文件。

具体的文件删除操作需要您自行处理，例如：在某个节点过载根目录启动定时删除机制，或启动多个Pod并发删除某种格式的子目录。

NAS存储卷挂载时间延长

问题现象：

NAS存储卷挂载时间延长。

问题原因：

同时满足以下配置时，挂载的PV及PVC将执行chmod或chown，导致挂载时间延长。

- 在PV及PVC模板中配置的参数AccessModes值为ReadWriteOnce。
- 在应用模板中配置了securityContext.fsgroup参数。

解决方法：

- 若应用模板中配置了securityContext.fsgroup参数，请删除securityContext下的fsgroup参数。
- 若需要将挂载目录内文件变成期望的UID和mode，可以手动将目标目录挂载到一台ECS。关于ECS实例挂载文件系统，请参见通过控制台实现ECS实例一键挂载文件系统。再通过命令行执行 `chown` 和 `chmod` ，完成后通过CSI使用NAS存储卷。关于如何通过CSI使用NAS存储卷，请参见使用NAS静态存储卷或使用NAS动态存储卷。
- 对于1.20及之后版本的Kubernetes集群，除了上述两种解决方法外，也可通过将fsGroupChangePolicy配置为OnRootMismatch，这时只有在首次启动时才会执行 `chmod` 或 `chown` 操作，导致存在挂载时间延长的问题，后续挂载OSS存储卷时挂载时间将恢复正常。关于fsGroupChangePolicy参数的更多信息，请参见为Pod或容器配置安全性上下文。

10.7. OSS存储卷

10.7.1. OSS存储卷概述

您可以在容器服务Kubernetes集群中使用阿里云OSS存储卷。本文介绍OSS存储卷的功能介绍、存储规格、适用场景、使用限制及计费说明。
目前，ACK仅支持OSS静态存储卷，不支持OSS动态存储卷。

功能介绍

阿里云对象存储OSS（Object Storage Service）是阿里云提供的海量、安全、低成本、高持久的云存储服务。

OSS具有丰富的安全防护能力，支持服务器端加密、客户端加密、防盗链白名单、细粒度权限管控、日志审计、合规保留策略（WORM）等特性。OSS为您的云端数据安全进行全方位的保驾护航，并满足您企业数据的安全与合规要求。

存储规格

对象存储OSS提供标准、低频访问、归档、冷归档四种存储类型，全面覆盖从热到冷的各种数据存储场景。更多信息，请参见存储类型介绍。

适用场景

根据业务需求，您可以对OSS存储卷做以下操作：

业务需求	参考链接
存储应用数据	具体操作，请参见使用OSS静态存储卷。
加密存储在OSS存储卷上的数据	阿里云容器服务只支持服务器端加密。更多信息，请参见服务器端加密。 具体操作，请参见加密OSS存储卷。

使用限制

配置OSS静态存储卷的说明如下：

- OSS为共享存储，可以同时为多个Pod提供共享存储服务。
- 您在使用OSS数据卷时，不要使用subpath的配置。
- 使用数据卷时，推荐每个应用使用独立的PV名字。
- 目前仅支持CentOS、Alibaba Cloud Linux 2及Alibaba Cloud Linux 3操作系统。
- OSS数据卷是使用OSSFS文件进行挂载的FUSE文件系统，适合于读文件场景。例如：读配置文件、视频、图片文件等场景。
- OSSFS不擅长于写文件的应用场景。如果您的业务是将文件写入存储的场景，推荐使用NAS存储卷服务。
- 内核态文件系统相较于用户态模拟的文件系统，在稳定性、性能上更具有优势，生产环境中推荐您使用NAS文件存储，而不推荐使用OSS对象存储。
- OSSFS可以通过调整配置参数的方式，优化其在缓存、权限等方面的表现，请参见常见问题、ossfs/README-CN.md及FAQ。

计费说明

关于OSS的计量计费方式，请参见计量计费概述。

10.7.2. 使用OSS静态存储卷

对象存储OSS（Object Storage Service）是阿里云提供的海量、安全、低成本、高持久的云存储服务。本文为您介绍如何通过命令行及控制台的方式使用OSS静态存储卷。

前提条件

- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建Bucket。具体操作，请参见创建存储空间。

  > **说明** 若Bucket和ECS实例位于相同地域，请选择私网域名。

- 已通过kubectl连接Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。

背景信息

阿里云对象存储服务（OSS）提供海量、安全、低成本、高可靠的云存储服务。OSS支持同时被多个Pod挂载。以下为OSS的使用场景：

- 对磁盘I/O要求低。
- 配置文件、图片、小视频等共享业务。

注意事项

- OSS不支持动态创建PV。
- 容器服务Kubernetes集群升级会重启Kubelet，OSSFS驱动跟随一起重启，导致OSS目录不可用。这时使用OSS的Pod需要重建，可在YAML文件中增加健康检查的配置，在容器内OSS目录不可用时自动重启Pod，重新挂载OSS。

  > **说明** 如果您使用的是v1.18.8.45及以上版本的csi-plugin和csi-provisioner插件，则不会出现上述问题。

- 若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间延长。

  > **说明** 若已配置securityContext.fsgroup参数，且需要减少挂载时间。具体操作，请参见OSS存储卷挂载时间延长。

- 挂载目录path中建议最多存放1000个文件。否则，在文件数量过多时OSSFS会占用大量内存，将导致Pod发生

OOM（Out Of Memory）事件。

通过控制台的方式使用OSS静态存储卷

步骤一：创建PV

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。
5. 在**存储卷**页面单击右上角的**创建**。
6. 在**创建存储卷**对话框中配置参数。

参数	描述
存储卷类型	支持云盘/NAS/OSS三种云存储类型。本文中选择为OSS。
名称	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。
存储驱动	支持Flexvolume和CSI。本文中选择为 CSI。
总量	所创建存储卷的容量。
访问模式	默认为ReadWriteMany。
访问证书	选择访问OSS所需的保密字典。 ○ **选择已有保密字典**：选择命名空间及保密字典。 ○ **新建保密字典**：配置命名空间、名称、AccessKey ID及AccessKey Secret。
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	您可以参考以下情况选择需要的访问域名。 ○ 如果Bucket和ECS实例位于不同地域（Region），请选择**公网域名**。 ○ 如果Bucket和ECS实例位于相同地域，请选择**私网域名**。
标签	为该存储卷添加标签。

7. 参数配置完成后，单击**创建**。

步骤二：创建PVC

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
5. 在**存储声明**页面，单击右上角的**创建**。
6. 在**创建存储声明**页面中，填写界面参数。

参数	描述
存储声明类型	支持云盘、NAS、OSS三种云存储类型。本文中选择OSS。
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** 若未创建存储卷，您可以设置**分配模式**为**创建存储卷**，配置创建存储卷参数。具体操作，请参见步骤一：创建PV。
已有存储卷	单击**选择已有存储类**，在目标存储卷右侧操作列单击**选择**，选择存储卷。
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存储空间容量。

7. 单击**创建**。

   创建成功后可以在列表中看到**csi-oss-pvc**，并且已绑定相应的存储卷。

步骤三：创建应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用镜像创建**。
6. 配置创建应用的参数信息。

   以下主要为您介绍数据卷的配置。关于其他参数的描述，请参见创建无状态工作负载Deployment。
   ACK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**：支持云存储类型。

   本例中配置了一个OSS类型的数据卷，将该OSS存储卷挂载到容器中*/tmp*路径下。



7. 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。

通过kubectl命令行的方式使用OSS静态存储卷

步骤一：创建静态PV及PVC

您有三种方式配置静态PV及PVC。

- 方式一：使用Secret创建静态卷PV及PVC
  通过Secret为CSI插件提供AccessKey信息。
- 方式二：配置PV及PVC的AccessKey
  直接在PV中配置AccessKey信息。
- 方式三：配置PV及PVC的STS权限
  在PV中配置STS进行权限认证。

方式一：使用Secret创建静态卷PV及PVC

1. 创建Secret。

   以下为通过Secret来配置AccessKey信息的YAML示例文件。

```
apiVersion: v1
kind: Secret
metadata:
 name: oss-secret
 namespace: default
stringData:
 akId: <yourAccessKey ID>
 akSecret: <yourAccessKey Secret>
```

   说明 创建Secret选择的Namespace需要和应用所在Namespaces一致。
   本示例中，`akId` 和 `akSecret` 需要替换成用户的AccessKey ID和AccessKey Secret。

2. 执行以下命令创建静态卷PV。

```
kubectl create -f pv-oss.yaml
```

   以下为创建静态卷PV的*pv-oss.yaml*示例文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-oss
 labels:
  alicloud-pvname: pv-oss
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 persistentVolumeReclaimPolicy: Retain
 csi:
  driver: ossplugin.csi.alibabacloud.com
  volumeHandle: pv-oss // 需要和PV名字一致。
  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"
   path: "/"
```

参数	描述
name	PV的名称。
labels	配置PV的标签。
storage	OSS的可使用量。
accessModes	配置访问模式。
persistentVolumeReclaimPolicy	PV回收策略。
driver	定义驱动类型。取值为ossplugin.csi.alibabacloud.com，表示使用OSS CSI插件。
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。目前只支持挂载存储空间Bucket，不支持挂载Bucket下面的子目录或文件。
url	挂载OSS的接入域名。 ○ 挂载节点和Bucket相同地域时，请使用私网地址。 ○ 挂载节点和Bucket不同地域时，请使用公网地址。 ○ 禁止使用VPC网络。
otherOpts	挂载OSS时支持输入定制化参数，格式为：`-o *** -o ***`。
path	表示挂载时相对Bucket根文件的目录结构，默认为/（v1.14.8.32-c77e277b-aliyun及之后版本支持）。

i. 登录容器服务管理控制台。
ii. 在控制台左侧导航栏中，单击**集群**。
iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
iv. 在集群管理页左侧导航栏选择**存储 > 存储卷**。可以在**存储卷**页面查看到创建的PV。

3. 执行以下命令创建静态卷PVC。

```
kubectl create -f pvc-oss.yaml
```

以下为创建静态卷PVC的*pvc-oss.yaml*示例文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-oss
spec:
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 5Gi
 selector:
  matchLabels:
   alicloud-pvname: pv-oss
```

参数	说明
name	PVC的名称。
accessModes	配置访问模式。
storage	声明应用使用量，不能大于存储卷的总量。
alicloud-pvname	通过标签关联PV，与PV标签保持一致。

在集群管理页左侧导航栏选择**存储 > 存储声明**，在**存储声明**页面可以看到创建的PVC。

方式二：配置PV及PVC的AccessKey

方式一：使用Secret创建静态卷PV及PVC是通过Secret为CSI插件提供AccessKey信息，您也可以执行以下命令直接在PV中配置AccessKey信息。

```
kubectl create -f pv-accesskey.yaml
```

以下为配置AccessKey信息的*pv-accesskey.yaml*示例文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-oss
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
 csi:
  driver: ossplugin.csi.alibabacloud.com
  volumeHandle: pv-oss // 需要和PV名字一致。
  volumeAttributes:
   bucket: "oss"
   url: "oss-cn-hangzhou.aliyuncs.com"
   otherOpts: "-o max_stat_cache_size=0 -o allow_other"
   akId: "***"
   akSecret: "***"
```

方式三：配置PV及PVC的STS权限

除了方式一：使用Secret创建静态卷PV及PVC和方式二：配置PV及PVC的AccessKey中使用Secret和使用AccessKey两种方式进行权限认证外。

您还可以在PV中执行以下命令配置STS进行权限认证。

```
kubectl create -f pv-sts.yaml
```

以下为配置STS权限认证的*pv-sts.yaml*示例文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-oss
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
 csi:
  driver: ossplugin.csi.alibabacloud.com
  volumeHandle: pv-oss // 需要和PV名字一致。
  volumeAttributes:
   bucket: "oss"
   url: "oss-cn-hangzhou.aliyuncs.com"
   otherOpts: "-o max_stat_cache_size=0 -o allow_other"
   authType: "sts"
```

步骤二：创建应用

创建一个名为*oss-static*的应用并挂载PVC。

执行以下命令创建*oss-static.yaml*。

```
kubectl create -f oss-static.yaml
```

以下为创建应用*oss-static.yaml*示例文件。

```
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"
     - name: pvc-oss
       mountPath: "/data1"
    livenessProbe:
     exec:
      command:
      - sh
      - -c
      - cd /data
     initialDelaySeconds: 30
     periodSeconds: 30
   volumes:
    - name: pvc-oss
      persistentVolumeClaim:
       claimName: pvc-oss
```

- livenessProbe：配置健康检查，详细介绍，请参见OSS存储卷概述。
- mountPath：OSS在容器中挂载的位置。
- claimName：PVC的名称，用于绑定PVC。

验证OSS的持久化存储

1. 查看部署*oss-static*应用的Pod和OSS文件。
   i. 执行以下命令，查看部署的oss-static应用所在Pod的名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME             READY  STATUS  RESTARTS  AGE
   oss-static-66fbb85b67-d****   1/1   Running  0     1h
   ```

ii. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec oss-static-66fbb85b67-d**** ls /data | grep tmpfile
```

无返回结果，说明*/data*路径下无文件。

2. 执行以下命令，在*/data*路径下创建文件*static*。

```
kubectl exec oss-static-66fbb85b67-d**** touch /data/tmpfile
```

3. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec oss-static-66fbb85b67-d**** ls /data | grep tmpfile
```

预期输出：

```
tmpfile
```

4. 执行以下命令，删除名称为 `oss-static-66fbb85b67-d****` 的Pod。

```
kubectl delete pod oss-static-66fbb85b67-d****
```

预期输出：

```
pod "oss-static-66fbb85b67-d****" deleted
```

5. 同时在另一个窗口中，执行以下命令，查看Pod删除及重建Pod的过程。

```
kubectl get pod -w -l app=nginx
```

预期输出：

```
NAME                READY  STATUS      RESTARTS  AGE
nginx-static-78c7dcb9d7-g****  2/2   Running      0     50s
nginx-static-78c7dcb9d7-g****  2/2   Terminating    0     72s
nginx-static-78c7dcb9d7-h****  0/2   Pending      0     0s
nginx-static-78c7dcb9d7-h****  0/2   Pending      0     0s
nginx-static-78c7dcb9d7-h****  0/2   Init:0/1     0     0s
nginx-static-78c7dcb9d7-g****  0/2   Terminating    0     73s
nginx-static-78c7dcb9d7-h****  0/2   Init:0/1     0     5s
nginx-static-78c7dcb9d7-g****  0/2   Terminating    0     78s
nginx-static-78c7dcb9d7-g****  0/2   Terminating    0     78s
nginx-static-78c7dcb9d7-h****  0/2   PodInitializing  0     6s
nginx-static-78c7dcb9d7-h****  2/2   Running      0     8s
```

6. 验证删除Pod后，存储空间里创建的文件是否还存在。

i. 执行以下命令，查看重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS  RESTARTS  AGE
oss-static-66fbb85b67-z****   1/1   Running  0     40s
```

ii. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec oss-static-66fbb85b67-z**** ls /data | grep tmpfile
```

预期输出：

```
tmpfile
```

*temfile*文件仍然存在，说明OSS的数据可持久化保存。

10.7.3. 加密OSS存储卷

数据加密适用于对高安全性或合规性具有较高要求的应用场景，您无需自建和维护密钥管理基础设施，通过加密保护存储在阿里云ECS上的数据就能保障数据的隐私性和自主性。本文介绍如何使用KMS（Key Management Service）托管的CMK（Customer Master Key）或OSS完全托管密钥功能对阿里云容器服务ACK集群的OSS存储卷数据加密。

前提条件

- 已创建Kubernetes集群，并且在该集群中安装CSI插件。具体操作，请参见创建Kubernetes托管版集群。
- 已创建Bucket。具体操作，请参见创建存储空间。

> **说明** 若Bucket和ECS实例位于相同地域，请选择私网域名。

- 已开通密钥管理服务。具体操作，请参见开通密钥管理服务。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。

加密方式

OSS存储卷加密分为服务端加密SSE（Server-Side Encryption）、客户端加密，阿里云容器服务只支持服务端加密：

- 服务端加密：OSS将数据保存到数据中心的磁盘之前进行加密，并且在下载对象时自动进行解密。
- 客户端加密：可以使用客户端加密SDK，在本地进行数据加密，并将加密后的数据上传到OSS。

OSS针对不同使用场景提供了两种服务器端加密方式，您可以根据实际使用场景选用。

- 使用KMS托管的默认CMK或指定CMK ID加密OSS Object（SSE-KMS）
  - KMS托管的默认CMK加密OSS Object：上传对象文件时，配置 `HTTP X-OSS-server-side-encryption` 为 `KMS` ，不指定CMK ID。
  - KMS托管的指定CMK ID加密OSS Object：上传对象文件时，配置 `HTTP X-OSS-server-side-encryption` 为 `KMS` ，指定 `X-OSS-server-side-encryption-key-id` 为 `CMK ID` 。

> **注意** 使用KMS密钥功能时会产生少量的KMS密钥API调用费用。关于费用详情，请参考计费说明。

  数据无需通过网络发送到KMS服务端进行加解密，是一种低成本的加解密方式。
- 使用OSS完全托管的密钥加密OSS Object（SSE-OSS）
  - 基于OSS完全托管的加密方式，是Object的一种属性。
  - OSS负责生成和管理数据加密密钥，并使用行业标准的强加密算法AES-256。
  - 上传对象文件时，配置 `HTTP X-OSS-server-side-encryption` 为 `AES256` 。

同一对象（Object）在同一时间内仅可以使用一种服务器端加密方式。

为OSS存储卷加密

通过OSSFS工具配置OSS存储卷的加密参数，在挂载PV时配置生效。关于OSSFS的配置及安装，请参见OSSFS的安装及配置。

OSSFS支持三种加密配置方式：

方式一：使用KMS托管的默认CMK加密OSS Object

1. 将以下内容复制到*kms-cmk-default.yaml*文件中。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: oss-csi-pv
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteOnce
 csi:
  driver: ossplugin.csi.alibabacloud.com
  volumeHandle: oss-csi-pv
  volumeAttributes:
   bucket: "python"
   url: "oss-cn-hangzhou.aliyuncs.com"
   otherOpts: "-o max_stat_cache_size=0 -o allow_other"
   akId: "LTAI4G6E3whAAKnzdRPx****"
   akSecret: "uRJeIi0cbahgWOhxncpp54wR5b****"
   path: "/"
   encrypted: "kms"
```

参数	说明
akId	AccessKey ID
akSecret	AccessKey Secret
encrypted	存储卷的加密方式： ○ kms：表示KMS加密方式。 ○ aes256：表示AES-256算法加密方式。

2. 执行以下命令创建加密存储卷。

```
kubectl create -f kms-cmk-default.yaml
```

3. 查看已创建的加密存储卷。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏选择**存储 > 存储卷**。可以在**存储卷**页面查看到创建的加密存储卷。

方式二：使用KMS托管的指定CMK ID加密OSS Object

1. 将以下内容复制到*kms-cmk.yaml*文件中。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: oss-csi-pv
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteOnce
 csi:
  driver: ossplugin.csi.alibabacloud.com
  volumeHandle: oss-csi-pv
  volumeAttributes:
   bucket: "python"
   url: "oss-cn-hangzhou.aliyuncs.com"
   otherOpts: "-o max_stat_cache_size=0 -o allow_other"
   akId: "LTAI4G6E3whAAKnzdRPx****"
   akSecret: "uRJeIi0cbahgWOhxncpp54wR5b****"
   path: "/"
   encrypted: "kms"
   kmsKeyId: "1234abcd-12ab-34cd-56ef-12345678****"
```

2. 执行以下命令创建加密存储卷。

```
kubectl create -f kms-cmk.yaml
```

3. 查看已创建的加密存储卷。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏选择**存储 > 存储卷**。可以在**存储卷**页面查看到创建的加密存储卷。

方式三：使用OSS完全托管的密钥加密OSS Object（SSE-OSS）

1. 将以下内容复制到*sse-oss.yaml*文件中。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: oss-csi-pv
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteOnce
 csi:
  driver: ossplugin.csi.alibabacloud.com
  volumeHandle: oss-csi-pv
  volumeAttributes:
   bucket: "python"
   url: "oss-cn-hangzhou.aliyuncs.com"
   otherOpts: "-o max_stat_cache_size=0 -o allow_other"
   akId: "LTAI4G6E3whAAKnzdRPx****"
   akSecret: "uRJeIi0cbahgWOhxncpp54wR5b****"
   path: "/"
   encrypted: "aes256"
```

2. 执行以下命令创建加密存储卷。

```
kubectl create -f sse-oss.yaml
```

3. 查看已创建的加密存储卷。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏选择**存储 > 存储卷**。可以在**存储卷**页面查看到创建的加密存储卷。

10.7.4. OSS存储卷FAQ

本文为您介绍OSS存储卷常见问题的处理方法。

OSS存储卷挂载时间延长

问题现象：

OSS存储卷挂载时间延长。

问题原因：

同时满足以下配置，kubelet在存储卷挂载过程中将执行chmod或chown操作，导致挂载时间延长。

- 在PV及PVC模板中配置的参数AccessModes值为ReadWriteOnce。
- 在应用模板中配置了securityContext.fsgroup参数。

解决方法：

- 若应用模板中配置了securityContext.fsgroup参数，请删除securityContext下的fsgroup参数。
- 若需要将挂载目录内文件变成期望的UID和mode，可以手动将Bucket挂载到一台ECS。关于如何将OSS挂载到ECS实例，请参见通过云存储网关挂载OSS。再通过命令行执行 `chown` 和 `chmod` ，完成后通过CSI使用OSS存储卷。关于如何通过CSI使用OSS存储卷，请参见使用OSS静态存储卷。
- 对于1.20及之后版本的Kubernetes集群，除了上述两种解决方法外，也可通过将fsGroupChangePolicy配置为OnRootMismatch，这时只有在首次启动时才会执行 `chmod` 或 `chown` 操作，导致存在挂载时间延长的问题，后续挂载OSS存储卷时挂载时间将恢复正常。关于fsGroupChangePolicy参数的更多信息，请参见为Pod或容器配置安全性上下文。

10.8. CPFS存储卷

10.8.1. CPFS存储卷概述

您可以在容器服务Kubernetes集群中使用阿里云CPFS（Cloud Paralleled File System）存储卷。本文介绍CPFS存储卷的功能介绍、存储规格、适用场景、使用限制及计费说明。

功能介绍

文件存储CPFS是阿里云推出的全托管、可扩展并行文件系统，满足高性能计算场景的需求。CPFS提供了统一的命名空间，支持成百上千的机器同时访问，拥有数十GB的吞吐、数百万的IOPS能力的同时还能保证亚毫秒级的延时。

- 高吞吐：IO带宽随容量线性提升，最大支持20 GB/s。如需更大带宽，请提交工单申请。
- 高IOPS：IOPS能力随容量扩展线性提升，最大支持2800000 IOPS。
- 低延时：稳定的亚毫秒级IO时延。
- 海量文件：全对称的元数据服务器架构，100000以上元数据IOPS能力。

存储规格

CPFS包括两种存储规格：100 MB/s/TiB基线和200 MB/s/TiB基线。200 MB/s/TiB基线相比100 MB/s/TiB基线规格，在相同容量下提供更高的IO带宽、IOPS能力和更低的IO时延，性能更优。

指标	100 MB/s/TiB基线	200 MB/s/TiB基线
IO带宽	min{100*存储容量（TiB），20000}MBps	min{200*存储容量（TiB），20000}MBps
IOPS	min{15000*存储容量（TiB），2800000}	min{30000*存储容量（TiB），2800000}
IO时延	800 μs	600 μs
容量规格	• 起步容量：3.6 TiB • 扩容步长：1.2 TiB • 最大容量：1 PiB	

适用场景

推荐您在AI训练、影视渲染、基因测序、自动驾驶、天文气象、油气勘探、EDA仿真设计等需要超高吞吐的应用场景使用CPFS。

根据业务需求，您可以对CPFS存储卷做以下操作：

业务需求	参考链接
存储应用数据	具体操作，请参见以下文档： • CPFS静态卷 • CPFS动态卷

使用限制

关于CPFS的使用限制，请参见使用限制。

计费说明

关于CPFS的计量项、计费方式及相关价格，请参见CPFS计费说明。

10.8.2. CPFS静态卷

CPFS（Cloud Paralleled File System）是一种并行文件系统。CPFS的数据存储在集群中的多个数据节点，并可由多个客户端同时访问，从而能够为大型高性能计算机集群提供高IOPS、高吞吐、低时延的数据存储服务。本文演示如何挂载及使用CPFS存储卷。

背景信息

阿里云容器服务Kubernetes（ACK） CSI支持静态存储卷挂载和动态存储卷挂载两种方式挂载CPFS存储卷。在静态存储卷挂载的方式中，通常需要手动编辑和创建一个PV和PVC进行挂载，当需要的PV和PVC数量很大的时候，手动创建就显得非常繁琐了，这时动态存储卷挂载的功能可以满足您的需求。本文演示如何在ACK集群中使用alibaba-cloud-csi-driver挂载CPFS存储卷，请参见alibaba-cloud-csi-driver。

> **说明** ACK CPFS存储卷服务目前只支持CPFS 1.0版，不支持2.0版本。

操作步骤

1. 部署CSI-CPFS组件。

   如果您需要挂载CPFS存储卷，需要按照以下步骤在ACK集群中先部署cpfs-plugin组件。

```
apiVersion: storage.k8s.io/v1
kind: CSIDriver
metadata:
```

```
metadata:
 name: cpfsplugin.csi.alibabacloud.com
spec:
 attachRequired: false
 podInfoOnMount: true
---
# This YAML defines all API objects to create RBAC roles for csi node plugin.
kind: DaemonSet
apiVersion: apps/v1
metadata:
 name: csi-cpfsplugin
 namespace: kube-system
spec:
 selector:
  matchLabels:
   app: csi-cpfsplugin
 template:
  metadata:
   labels:
    app: csi-cpfsplugin
  spec:
   tolerations:
   - operator: Exists
   priorityClassName: system-node-critical
   serviceAccount: admin
   hostNetwork: true
   hostPID: true
   containers:
   - name: driver-registrar
    image: registry.cn-hangzhou.aliyuncs.com/acs/csi-node-driver-registrar:v1.2.0
    imagePullPolicy: Always
    args:
    - "--v=5"
    - "--csi-address=/var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com/csi.sock"
    - "--kubelet-registration-path=/var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com/csi.sock"
    volumeMounts:
    - name: kubelet-dir
     mountPath: /var/lib/kubelet/
    - name: registration-dir
     mountPath: /registration
   - name: csi-cpfsplugin
    securityContext:
     privileged: true
     capabilities:
      add: ["SYS_ADMIN"]
     allowPrivilegeEscalation: true
    image: registry.cn-hangzhou.aliyuncs.com/acs/csi-cpfsplugin:v1.16.6-d539237c
    imagePullPolicy: "Always"
    args:
    - "--endpoint=$(CSI_ENDPOINT)"
    - "--v=5"
    - "--driver=cpfsplugin.csi.alibabacloud.com"
    env:
    - name: CSI_ENDPOINT
     value: unix://var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com/csi.sock
    volumeMounts:
    - name: kubelet-dir
     mountPath: /var/lib/kubelet/
```

```
      mountPropagation: "Bidirectional"
    - mountPath: /var/log/
      name: host-log
    - name: etc
      mountPath: /host/etc
  volumes:
  - name: kubelet-dir
    hostPath:
      path: /var/lib/kubelet/
      type: Directory
  - name: registration-dir
    hostPath:
      path: /var/lib/kubelet/plugins_registry
      type: DirectoryOrCreate
  - name: host-log
    hostPath:
      path: /var/log/
  - name: etc
    hostPath:
      path: /etc
updateStrategy:
  type: RollingUpdate
```

执行以下命令检查应用运行状态。

```
kubectl get pod -nkube-system |grep cpfs
```

预期输出：

```
csi-cpfsplugin-8t585                 2/2    Running  0      4h43m
csi-cpfsplugin-9z5xj                 2/2    Running  0      4h43m
csi-cpfsplugin-bdm22                 2/2    Running  0      4h43m
csi-cpfsplugin-bjnlx                 2/2    Running  0      4h44m
csi-cpfsplugin-nv7vg                 2/2    Running  0      4h43m
csi-cpfsplugin-zc7z5                 2/2    Running  0      4h43m
```

2. 使用CPFS存储卷。
   i. 创建CPFS文件系统。具体操作，请参见创建文件系统。

ii. 创建PV和PVC。

使用以下模板，创建CPFS类型的PV和PVC。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: cpfs-csi-pv
 labels:
  alicloud-pvname: cpfs-pv
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
 csi:
  driver: cpfsplugin.csi.alibabacloud.com
  volumeHandle: cpfs-csi-pv
  volumeAttributes:
   server: "xxxxxx@tcp:xxxxxx@tcp"
   fileSystem: "xxxxxx"
   subPath: "/k8s"
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: cpfs-pvc
spec:
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
   storage: 5Gi
 selector:
  matchLabels:
   alicloud-pvname: cpfs-pv
```

iii. 创建应用。

使用以下模板引用上述PVC创建应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-cpfs
 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: cpfs-pvc
       mountPath: "/data"
   volumes:
    - name: cpfs-pvc
      persistentVolumeClaim:
       claimName: cpfs-pvc
```

10.8.3. CPFS动态卷

CPFS（Cloud Paralleled File System）是一种并行文件系统。CPFS的数据存储在集群中的多个数据节点，并可由多个客户端同时访问，从而能够为大型高性能计算机集群提供高IOPS、高吞吐、低时延的数据存储服务。本文介绍如何挂载及使用CPFS存储卷。

前提条件

已开通CPFS服务。

若未开通CPFS服务，请提交工单联系CPFS团队添加白名单。

背景信息

阿里云容器服务Kubernetes（ACK） CSI支持静态存储卷挂载和动态存储卷挂载两种方式挂载CPFS存储卷。在静态存储卷挂载的方式中，通常需要手动编辑和创建一个PV和PVC进行挂载，当需要的PV和PVC数量很大的时候，手动创建就显得非常繁琐了，这时动态存储卷挂载的功能可以满足您的需求。本文介绍如何在ACK集群中使用alibaba-cloud-csi-driver挂载CPFS存储卷。具体操作，请参见alibaba-cloud-csi-driver。

> **说明** 容器服务ACK的CPFS存储卷服务目前只支持CPFS 1.0版，不支持2.0版本。

操作步骤

1. 部署CSI-CPFS组件。

   如果您需要挂载CPFS存储卷，需要按照以下步骤在ACK集群中先部署cpfs-provisioner和cpfs-plugin组件。

   - 部署cpfs-provisioner。

```
kind: Deployment
apiVersion: apps/v1
metadata:
 name: csi-cpfs-provisioner
 namespace: kube-system
spec:
 selector:
  matchLabels:
   app: csi-cpfs-provisioner
 replicas: 2
 template:
  metadata:
   labels:
    app: csi-cpfs-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: external-cpfs-provisioner
      image: registry.cn-hangzhou.aliyuncs.com/acs/csi-provisioner:v1.6.0-e360c7e43-aliyun
      args:
       - "--provisioner=cpfsplugin.csi.alibabacloud.com"
       - "--csi-address=$(ADDRESS)"
       - "--volume-name-prefix=cpfs"
       - "--timeout=150s"
       - "--enable-leader-election=true"
       - "--leader-election-type=leases"
       - "--retry-interval-start=500ms"
       - "--v=5"
      env:
       - name: ADDRESS
         value: /socketDir/csi.sock
      imagePullPolicy: "Always"
      volumeMounts:
       - name: socket-dir
         mountPath: /socketDir
   volumes:
    - name: socket-dir
      hostPath:
       path: /var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com
       type: DirectoryOrCreate
```

- 部署csi-plugin。

```
apiVersion: storage.k8s.io/v1
kind: CSIDriver
metadata:
```

```
metadata:
  name: cpfsplugin.csi.alibabacloud.com
spec:
  attachRequired: false
  podInfoOnMount: true
---
# This YAML defines all API objects to create RBAC roles for csi node plugin.
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: csi-cpfsplugin
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: csi-cpfsplugin
  template:
    metadata:
      labels:
        app: csi-cpfsplugin
    spec:
      tolerations:
      - operator: Exists
      priorityClassName: system-node-critical
      serviceAccount: admin
      hostNetwork: true
      hostPID: true
      containers:
      - name: driver-registrar
        image: registry.cn-hangzhou.aliyuncs.com/acs/csi-node-driver-registrar:v1.2.0
        imagePullPolicy: Always
        args:
        - "--v=5"
        - "--csi-address=/var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com/csi.sock"
        - "--kubelet-registration-path=/var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com/csi.sock"
        volumeMounts:
        - name: kubelet-dir
          mountPath: /var/lib/kubelet/
        - name: registration-dir
          mountPath: /registration
      - name: csi-cpfsplugin
        securityContext:
          privileged: true
          capabilities:
            add: ["SYS_ADMIN"]
          allowPrivilegeEscalation: true
        image: registry.cn-hangzhou.aliyuncs.com/acs/csi-cpfsplugin:v1.16.6-d539237c
        imagePullPolicy: "Always"
        args:
        - "--endpoint=$(CSI_ENDPOINT)"
        - "--v=5"
        - "--driver=cpfsplugin.csi.alibabacloud.com"
        env:
        - name: CSI_ENDPOINT
          value: unix://var/lib/kubelet/csi-plugins/cpfsplugin.csi.alibabacloud.com/csi.sock
        volumeMounts:
        - name: kubelet-dir
          mountPath: /var/lib/kubelet/
```

```
      mountPropagation: "Bidirectional"
    - mountPath: /var/log/
      name: host-log
    - name: etc
      mountPath: /host/etc
  volumes:
  - name: kubelet-dir
    hostPath:
      path: /var/lib/kubelet/
      type: Directory
  - name: registration-dir
    hostPath:
      path: /var/lib/kubelet/plugins_registry
      type: DirectoryOrCreate
  - name: host-log
    hostPath:
      path: /var/log/
  - name: etc
    hostPath:
      path: /etc
updateStrategy:
  type: RollingUpdate
```

2. 使用CPFS动态卷。

   i. 使用以下模板创建StorageClass。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-cpfs
parameters:
  volumeAs: subpath
  server: "cpfs-0237cc41-**.cn-shenzhen.cpfs.nas.aliyuncs.com@tcp:cpfs-0237cc41-**.cn-shenzhen.cpfs.nas.aliyuncs.com@tcp:/0237cc41"
  archiveOnDelete: "false"
provisioner: cpfsplugin.csi.alibabacloud.com
reclaimPolicy: Delete
```

参数	说明
volumeAs	定义动态卷的类型。目前只支持subpath类型；标志创建的PV实质是CPFS服务器上的一个子目录。
server	定义CPFS服务器地址。格式为 `挂载点地址@tcp:挂载点地址@tcp:/subpath` 。
archiveOnDelete	表示删除PVC、PV时候，处理CPFS子目录的方式： ■ 如果reclaimPolicy为Delete，且archiveOnDelete为false：会直接删除远端目录和数据，请谨慎使用。 ■ 如果reclaimPolicy为Delete，且archiveOnDelete为true：会将远端的目录更新为其他名字备份。 ■ 如果reclaimPolicy为Retain，远端的目录不作处理。

ii. 使用以下模板创建PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: cpfs-pvc
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: alicloud-cpfs
 resources:
  requests:
   storage: 20Gi
```

> 注意 PVC中要指定使用的storageClassName。

iii. 使用以下模板创建应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-cpfs
 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: cpfs-pvc
       mountPath: "/data"
   volumes:
    - name: cpfs-pvc
      persistentVolumeClaim:
       claimName: cpfs-pvc
```

10.9. 本地存储卷

10.9.1. 本地存储卷概述

您可以在阿里云容器服务ACK集群中使用本地存储卷。目前阿里云CSI插件支持通过PV和PVC方式挂载本地卷。

功能介绍

本地存储是个相对概念：存储介质可以是阿里云提供的本地数据盘（请参见本地盘），可以是阿里云云盘（请参见云盘概述），也可以是内存当做临时本地存储使用。

- 本地盘是ECS实例所在物理机上的本地硬盘设备。本地盘能够为ECS实例提供本地存储访问能力，具有低时延、高随机IOPS、高吞吐量和高性价比的优势。
- HostPath：基于社区标准，支持自动创建主机目录，Pod可迁移（不绑定节点）。
- LocalVolume：基于社区标准，Pod不可迁移或者限制性迁移（绑定节点或者设置Label）。
- LVM：在本地盘上面通过VolumeGroup进行磁盘虚拟化，并通过LVM卷的方式切分给上层应用使用。
  LVM方案支持：卷自动创建、删除、不同的卷隔离、卷的IOPS配置、VG的自动化运维、LVM卷容量规划等能力。
- 内存：通过将内存虚拟化成Tmpfs文件系统，给应用提供高速本地存储的能力。

存储规格

阿里云提供两种本地盘：NVMe SSD本地盘及SATA HDD本地盘。更多信息，请参见NVMe SSD本地盘及SATA HDD本地盘。

适用场景

本地存储适用于对存储I/O性能、海量存储性价比有极高要求的业务场景。

根据业务需求，您可以对本地存储卷做以下操作：

业务需求	参考链接
存储应用数据	• HostPath存储卷可将主机节点文件系统上的文件或目录挂载至Pod中。具体操作，请参见HostPath数据卷。 • LocalVolume存储卷可将本地存储设备（例如磁盘、分区或者目录）挂载至Pod中。具体操作，请参见LocalVolume数据卷。 • 在本地盘上通过VolumeGroup进行磁盘虚拟化，并通过LVM存储卷切分磁盘给Pod使用。具体操作，请参见LVM数据卷。 • 在本地盘上通过文件系统（如Ext4）的ProjectQuota功能，实现目录级别的容量控制，通过CSI插件实现QuotaPath存储卷的切分、限额、挂载等生命周期管理。具体操作，请参见QuotaPath数据卷。 • 内存临时存储卷为容器服务提供高性能的本地临时存储。具体操作，请参见内存数据卷。

使用限制

- 目前支持通过HostPath、LocalVolume、LVM、内存等挂载方式挂载本地存储。
- 本地存储并非高可用存储卷，只适用于一些临时数据的保存及应用自带高可用的场景。

计费说明

本地盘的费用包括在本地盘挂载的实例的费用里。更多信息，请参见包年包月和按量付费。

> **说明** 阿里云提供延停权益，即当按量付费的资源发生欠费后，提供一定额度或时长继续使用云服务的权益。延停期间正常计费。延停的权益额度不是欠费总额的上限。您延停的额度或时长根据您在阿里云的历史消费等因素，每个月自动计算并更新。更多信息，请参见延期免停权益。

10.9.2. HostPath数据卷

HostPath卷能将主机节点文件系统上的文件或目录挂载到您的Pod中。本文介绍如何使用HostPath数据卷。

背景信息

阿里云ACK集群兼容社区HostPath本地挂载方案的详细信息，请参见Hostpath。

挂载模式

HostPath支持以下几种挂载模式。

挂载模式	描述
DirectoryOrCreate	如果在给定路径上什么都不存在，将根据需要创建空目录，权限设置为0755，与Kubelet有相同的组和属主信息。
Directory	在给定路径上必须存在目录。
FileOrCreate	如果在给定路径上什么都不存在，那么将在给定路径根据需要创建空文件，权限设置为0644，具有与Kubelet相同的组和所有权。
File	在给定路径上必须存在文件。

HostPath数据卷使用示例

- Pod中以直接定义 `hostPath` 的方式使用HostPath数据卷：

```
apiVersion: v1
kind: Pod
metadata:
 name: test
spec:
 containers:
 - image: nginx:1.7.9
   name: test
   volumeMounts:
   - mountPath: /test
     name: test-volume
 volumes:
 - name: test-volume
   hostPath:
     path: /data
     type: DirectoryOrCreate
```

- 通过PV和PVC定义 `hostPath` 的方式使用HostPath数据卷：

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: task-pv-volume
 labels:
  type: local
spec:
 capacity:
  storage: 10Gi
 accessModes:
  - ReadWriteOnce
 hostPath:
  path: "/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: hostpath
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 10Gi
```

- Pod运行时挂载HostPath数据卷：
  Kubernetes生态中，Pod运行时，其挂载的数据卷不可以修改，所以原则上运行中的容器不能动态挂载数据卷。下面方式可以在一定程度上实现运行中Pod动态挂载外部存储的需求。

  > **说明**
  > - 本方案依赖Linux中mount Propagation的Bidirectional属性，可以将主机挂载目录映射到容器中。
  > - 目标Pod必须具有Privilege权限（Bidirectional需求）。
  > - 目标Pod需要挂载一个主机目录，并配置Bidirectional，运行中的容器依赖该目录接收外部挂载。

  部署一个Nginx应用使用HostPath数据卷的示例模板如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-nas
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    command: ["sh", "-c"]
    args: ["sleep 10000"]
    securityContext:
     privileged: true
     capabilities:
      add: ["SYS_ADMIN"]
     allowPrivilegeEscalation: true
    volumeMounts:
     - name: dynamic-volume
       mountPropagation: "Bidirectional"
       mountPath: "/dynamic-volume"
   volumes:
    - name: dynamic-volume
      hostPath:
       path: /mnt/dynamic-volume
       type: DirectoryOrCreate
```

> **说明**
> - 主机目录：*/mnt/dynamic-volume*。
> - Pod内目录：*/dynamic-volume*。

上述配置，可以在Pod运行时，在Pod所在节点上的*/mnt/dynamic-volume/*****路径挂载外部存储，从而将外部存储卷映射到Pod内部，实现了运行中Pod动态挂载外部存储的能力。

10.9.3. LocalVolume数据卷

LocalVolume数据卷可使本地存储设备，例如磁盘、分区或者目录挂载至Pod中。本文介绍如何使用LocalVolume数据卷。

背景信息

有关阿里云ACK集群兼容社区LocalVolume本地挂载方案的详情，请参见local。

LocalVolume数据卷与HostPath数据卷的区别：

HostPath	LocalVolume
不支持节点调度。	支持节点调度。

HostPath	LocalVolume
支持挂载目录、文件等格式。	支持挂载目录、裸设备。
支持自动创建目录。	不支持自动创建目录。

使用LocalVolume数据卷

通过PV定义LocalVolume类型卷的示例模板如下。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: example-pv
spec:
 capacity:
  storage: 100Gi
 volumeMode: Filesystem
 accessModes:
 - ReadWriteOnce
 persistentVolumeReclaimPolicy: Delete
 storageClassName: local-storage
 local:
  path: /mnt/disks/ssd1
 nodeAffinity:
  required:
   nodeSelectorTerms:
   - matchExpressions:
    - key: kubernetes.io/hostname
      operator: In
      values:
      - example-node
```

Kubernetes官方LocalVolume不支持动态卷创建能力。社区提供了实现方案，但具体的部署、运维等工作需要自行实现。更多信息，请参见sig-storage-local-static-provisioner。

10.9.4. LVM数据卷

您可以在本地盘上通过VolumeGroup进行磁盘虚拟化，并通过LVM数据卷切分磁盘给应用使用。本文介绍如何使用LVM数据卷。

背景信息

使用HostPath和LocalVolume都可以实现Pod对主机存储空间的访问，但均具有其局限性：

- Kubernetes没有提供上述本地存储卷的生命周期管理能力，需要管理员手动管理、运维存储卷。
- 多个Pod共享一个本地存储时，需要共享存储目录或者分别使用其子目录，无法做到容量隔离。
- 多个Pod共享一个本地存储时，IOPS、吞吐等指标共享了整个存储空间，无法进行限制。
- 新建Pod使用本地存储时，不了解各节点存储空间余量，无法进行合理的存储卷调度。

为此ACK提供了LVM数据卷方案，以解决上述问题。

功能介绍

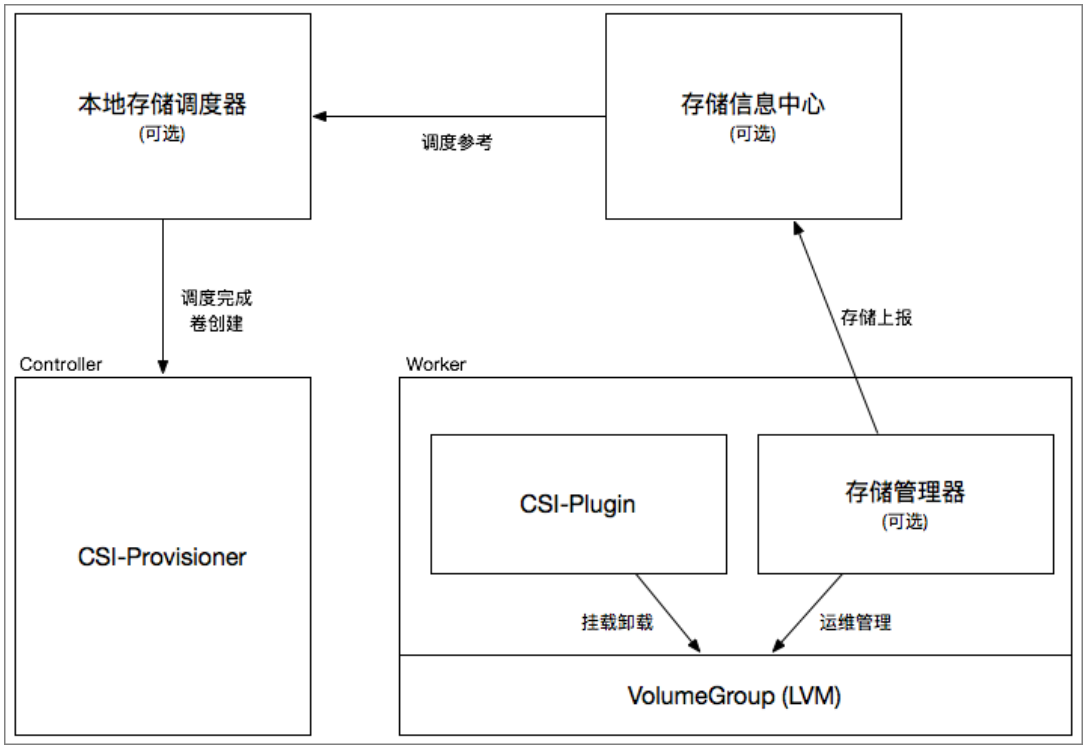
- LVM数据卷生命周期管理：卷自动创建、删除、挂载、卸载。
- LVM数据卷扩容功能。
- LVM卷的监控能力。

- LVM卷的IOPS限制。
- 节点本地存储管理：自动运维VolumeGroup。
- LVM卷的集群容量感知能力。

注意事项

- LVM为本地存储类型，不适用于高可用数据场景。
- VolumeGroup运维、本地存储容量感知为可选项（暂缓提供）。

实现架构



基础的LVM功能（卷的生命周期管理、扩容、挂载、格式化等）由CSI-Provisioner和CSI-Plugin实现。

单元	详情
存储管理器	节点本地存储（VolumeGroup）的运维管理以及容量统计。可以不用该组件，由Worker管理运维VolumeGroup。
存储信息中心	保存节点本地存储信息，包括容量、VolumeGroup等信息。
本地存储调度器	实现新建PVC对集群存储容量感知功能。

部署Plugin和Provisioner组件

在部署Plugin和Provisioner组件前，需要先为 `clusterrole/alicloud-csi-plugin` 增加Secrets的RBAC权限。

1. 执行以下命令，查看集群 `clusterrole/alicloud-csi-plugin` 是否有Secrets的 `create` 权限。

```
echo `JSONPATH='{range .rules[*]}{@.resources}:{@.verbs}\r\n
{end}' \
 && kubectl get clusterrole  alicloud-csi-plugin -o jsonpath="$JSONPATH";` | grep secrets
```

预期输出：

```
["secrets"]:["get","list"]
```

2. 若 clusterrole/alicloud-csi-plugin 没有Secrets的 create 权限，请执行以下命令增加此权限。

```
kubectl patch clusterrole alicloud-csi-plugin --type='json' -p='[{"op": "add", "path": "/rules/0", "value":{ "apiGroups": [""], "resources": ["secrets"], "verbs": ["create"]}}]'
```

预期输出：

```
clusterrole.rbac.authorization.k8s.io/alicloud-csi-plugin patched
```

3. 执行以下命令，查看集群中 clusterrole/alicloud-csi-plugin Secrets的 create 权限是否添加成功。

```
echo `JSONPATH='{range .rules[*]}{@.resources}:{@.verbs} \r\n
{end}' \
 && kubectl get clusterrole  alicloud-csi-plugin -o jsonpath="$JSONPATH";` | grep secrets
```

预期输出：

```
["secrets"]:["create"]
["secrets"]:["get","list"]
```

从预期输出可得，集群中 clusterrole/alicloud-csi-plugin Secrets的 create 权限已添加成功。

4. 安装Plugin和Provisioner组件。

LVM CSI插件分为2个组件：Plugin（负责挂载、卸载LVM卷）和Provisioner（负责创建LVM卷和PV对象）。

部署LVM CSI-Plugin 部署LVM CSI-Provisioner

使用LVM存储卷示例

使用CSI-Provisioner自动创建PV，有以下特点：

- StorageClass中需要指定VolumeGroup名字。
- 如果期望创建的PV在某个节点，需要给PVC添加Annotations：volume.kubernetes.io/selected-node: nodeName。

1. 使用以下模板创建StorageClass。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-local
provisioner: localplugin.csi.alibabacloud.com
parameters:
  volumeType: LVM
  vgName: volumegroup1
  fsType: ext4
  lvmType: "striping"
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

参数	描述
volumeType	表示本地存储类型为LVM（后续将支持其他类型本地存储）。
vgName	必选，VolumeGroup的名字。
lvmType	生成的LVM类型，支持linear（线性）、striping（条带化）。

参数	描述
fsType	格式文件系统类型。

2. 使用以下模板创建PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: lvm-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 2Gi
 storageClassName: csi-local
```

3. 使用以下模板创建应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-lvm
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    volumeMounts:
     - name: lvm-pvc
       mountPath: "/data"
   volumes:
    - name: lvm-pvc
      persistentVolumeClaim:
       claimName: lvm-pvc
```

4. 查看应用状态。

   执行以下命令查看Pod信息。

```
kubectl get pod
```

   预期输出：

```
NAME              READY  STATUS  RESTARTS  AGE
deployment-lvm-9f798687c-m****  1/1   Running  0     9s
```

执行以下命令查看PVC信息。

```
kubectl get pvc
```

预期输出：

```
NAME    STATUS  VOLUME                    CAPACITY  ACCESS MODES  STORAGECLASS  AGE
lvm-pvc  Bound   disk-afacf7a9-3d1a-45da-b443-24f8fb35****  2Gi     RWO       csi-local    16s
```

执行以下命令查看PV信息。

```
kubectl get pv
```

预期输出：

```
NAME                      CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM       STORAGECLASS  RE
ASON  AGE
disk-afacf7a9-3d1a-45da-b443-24f8fb35****  2Gi     RWO       Delete      Bound   default/lvm-pvc  csi-local
12s
```

执行以下命令查看Pod挂载详情。

```
kubectl exec -ti deployment-lvm-9f798687c-m**** sh df /data
```

预期输出：

```
Filesystem                                1K-blocks  Used  Available Use% Mounted on
/dev/mapper/volumegroup1-disk--afacf7a9--3d1a--45da--b443--24f8fb35****  1998672   6144  1976144  1%  /d
ata
```

执行以下命令列出*/data*下的目录。

```
ls /data
```

预期输出：

```
lost+found
```

执行以下命令在*/data*下新增*test*目录并查看。

```
touch /data/test
ls /data
```

预期输出：

```
lost+found  test
```

执行以下命令退出登录。

```
exit
```

执行以下命令删除Pod重建。

```
kubectl delete pod deployment-lvm-9f798687c-m****
```

预期输出：

```
pod "deployment-lvm-9f798687c-m****" deleted
```

执行以下命令查看Pod信息。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE
deployment-lvm-9f798687c-j****  1/1    Running  0      2m19s
```

执行以下命令查看Pod挂载详情。

```
kubectl exec deployment-lvm-9f798687c-j**** ls /data
```

预期输出：

```
lost+found
test
```

5. 扩容LVM卷。

执行以下命令查看PVC信息。

```
kubectl get pvc
```

预期输出：

```
NAME    STATUS  VOLUME                  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
lvm-pvc  Bound   disk-afacf7a9-3d1a-45da-b443-24f8fb35****  2Gi     RWO       csi-local    6m50s
```

执行以下命令将PVC扩容到4 GiB。

```
kubectl patch pvc lvm-pvc -p '{"spec":{"resources":{"requests":{"storage":"4Gi"}}}}'
```

预期输出：

```
persistentvolumeclaim/lvm-pvc patched
```

执行以下命令查看PVC信息。

```
kubectl get pvc
```

预期输出：

```
NAME    STATUS  VOLUME                  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
lvm-pvc  Bound   disk-afacf7a9-3d1a-45da-b443-24f8fb35****  4Gi     RWO       csi-local    7m26s
```

执行以下命令查看LVM卷扩容到4 GiB。

```
kubectl exec deployment-lvm-9f798687c-j**** df /data
```

预期输出：

```
Filesystem                                  1K-blocks  Used  Available Use% Mounted on
/dev/mapper/volumegroup1-disk--afacf7a9--3d1a--45da--b443--24f8fb35****  4062912   8184  4038344  1%  /data
```

6. 执行以下命令监控LVM卷。

```
curl -s localhost:10255/metrics | grep lvm-pvc
```

预期输出：

```
kubelet_volume_stats_available_bytes{namespace="default",persistentvolumeclaim="lvm-pvc"} 1.917165568e+09
kubelet_volume_stats_capacity_bytes{namespace="default",persistentvolumeclaim="lvm-pvc"} 1.939816448e+09
kubelet_volume_stats_inodes{namespace="default",persistentvolumeclaim="lvm-pvc"} 122400
kubelet_volume_stats_inodes_free{namespace="default",persistentvolumeclaim="lvm-pvc"} 122389
kubelet_volume_stats_inodes_used{namespace="default",persistentvolumeclaim="lvm-pvc"} 11
kubelet_volume_stats_used_bytes{namespace="default",persistentvolumeclaim="lvm-pvc"} 5.873664e+06
```

可以将上述监控数据接入Prometheus并在前端展示。具体操作，请参见开源Prometheus监控。

10.9.5. QuotaPath数据卷

您可以在本地盘上通过文件系统（如Ext4）的Project Quota功能，实现目录级别的容量Quota控制，通过CSI插件实现QuotaPath数据卷的切分、限额、挂载等生命周期管理。本文介绍如何使用QuotaPath数据卷。

前提条件

已部署LVM CSI插件。具体操作，请参见部署Plugin和Provisioner组件。

QuotaPath与HostPath、LVM的区别

使用HostPath、LVM、QuotaPath都可以实现Pod对主机存储空间的访问，但其各具特点：

- HostPath将目录进行切分，多个目录间共享相同存储设备的空间、IO等资源。
- LVM将设备进行虚拟化，然后切分成多个卷，每个卷都独立拥有存储限额。
- QuotaPath使用文件系统的Quota功能，将目录进行切分，每个目录拥有独立的存储限额。

功能介绍

- QuotaPath数据卷生命周期管理：自动创建、删除、挂载、卸载。
- QuotaPath数据卷扩容功能。
- 节点本地存储管理：自动运维QuotaPath根目录。
- QuotaPath卷的集群容量感知能力。

注意事项

- QuotaPath为本地存储类型，不适用于高可用数据场景。
- QuotaPath根目录运维、本地存储容量感知为可选项（暂缓提供）。
- 若使用Root特权模式启动容器，则Quota限制不会生效（Ext4特性）。

使用QuotaPath示例

使用CSI-Provisioner自动创建PV，有以下特点：

- StorageClass中需要指定rootPath名字。
- 如果期望创建的PV在某个节点，需要给PVC添加Label：volume.kubernetes.io/selected-node: nodeName。

1. 创建StorageClass。

i. 使用以下内容创建*alicloud-local-quota.yaml*文件。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-local-quota
parameters:
 volumeType: QuotaPath
 rootPath: /mnt/quota
provisioner: localplugin.csi.alibabacloud.com
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

参数	描述
volumeType	本地存储类型，本示例中存储类型为QuotaPath。
rootPath	可选，表示QuotaPath所在目录名称。

ii. 执行以下命令创建StorageClass。

```
kubectl create -f alicloud-local-quota.yaml
```

2. 创建PVC。

i. 使用以下内容创建*csi-quota.yaml*文件。

默认CSI插件会使用*/mnt/quotapath.namespacex.x*作为QuotaPath的根目录，所有的PV都将在这个目录下面分配并创建子目录。
可以在PVC上添加以下Annotation，用来自定义QuotaPath父目录*volume.kubernetes.io/selected-storage: /mnt/xxx*。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: csi-quota
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 2Gi
 storageClassName: alicloud-local-quota
```

ii. 执行以下命令创建PVC。

```
kubectl create -f csi-quota.yaml
```

3. 使用以下模板创建应用。

i. 使用以下内容创建*web-quota.yaml*文件。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 ports:
 - port: 80
  name: web
 selector:
  app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web-quota
spec:
 selector:
  matchLabels:
   app: nginx
 serviceName: "nginx"
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
    volumeMounts:
    - name: disk-ssd
     mountPath: /data
   volumes:
    - name: "disk-ssd"
     persistentVolumeClaim:
      claimName: csi-quota
```

ii. 执行以下命令创建应用。

```
kubectl create -f web-quota.yaml
```

4. 查看应用状态。

执行以下命令查看Pod信息。

```
kubectl get pod |grep quota
```

预期输出：

```
NAME     READY  STATUS   RESTARTS  AGE
web-quota-0  1/1   Running  0     16s
```

执行以下命令查看PVC信息。

```
kubectl get pvc
```

预期输出：

```
NAME     STATUS  VOLUME         CAPACITY  ACCESS MODES  STORAGECLASS      AGE
csi-quota  Bound   local-f4b129a5-****  2Gi     RWO       alicloud-local-quota  48s
```

执行以下命令查看PV信息。

```
kubectl get pv |grep quota
```

预期输出：

```
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM         STORAGECLASS      REASO
N  AGE
local-f4b129a5-****  2Gi     RWO       Delete      Bound   default/csi-quota  alicloud-local-quota      66s
```

执行以下命令查看Pod挂载详情。

```
kubectl exec -ti web-quota-0 sh
df |grep data
```

预期输出：

```
Filesystem  1K-blocks  Used Available Use% Mounted on
/dev/vdd   2097152    4  2097148  1% /data
```

执行以下命令列出*/data*下的目录。

```
ls /data
```

预期输出：

```
lost+found
```

执行以下命令在*/data*下新增test目录并查看。

```
touch /data/test
ls /data
```

预期输出：

```
lost+found test
```

5. 扩容数据卷。

   执行以下命令查看PVC信息。

   ```
   kubectl get pvc
   ```

   预期输出：

   ```
   NAME     STATUS  VOLUME         CAPACITY  ACCESS MODES  STORAGECLASS      AGE
   csi-quota  Bound   local-f4b129a5-****  2Gi     RWO       alicloud-local-quota  42s
   ```

   执行以下命令将PVC扩容到3 GiB。

   ```
   kubectl patch pvc csi-quota -p '{"spec":{"resources":{"requests":{"storage":"3Gi"}}}}'
   ```

   预期输出：

   ```
   persistentvolumeclaim/csi-quota patched
   ```

   执行以下命令查看PVC信息。

   ```
   kubectl get pvc
   ```

预期输出：

```
NAME     STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS       AGE
csi-quota  Bound   local-f4b129a5-****  3Gi     RWO       alicloud-local-quota  4m30s
```

执行以下命令查看数据卷扩容到3 GiB。

```
kubectl exec -ti web-quota-0 sh
df |grep data
```

预期输出：

```
Filesystem  1K-blocks  Used Available Use% Mounted on
/dev/vdd   3145728     4  3145724  1% /data
```

10.9.6. 内存数据卷

ACK提供了内存临时存储卷，可以给容器服务提供高性能的本地临时存储。

背景信息

- 内存存储卷是临时存储，Pod重启后数据将会丢失。
- 本地存储适合于高速缓存应用场景。

插件部署

内存CSI插件由两个组件：Plugin（挂载内存卷）和Provisioner（创建内存卷）。

部署CSI Memory Plugin 部署CSI Memory Provisioner

使用内存数据卷示例

1. 使用以下模板部署StorageClass。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-mem
provisioner: memplugin.csi.alibabacloud.com
reclaimPolicy: Delete
```

2. 使用以下模板创建PVC与应用。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mem-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 2Gi
 storageClassName: csi-mem
 ---
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-mem
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    command: ["sh", "-c"]
    args: ["sleep 10000"]
    volumeMounts:
     - name: mem-pvc
       mountPath: "/data"
   volumes:
    - name: mem-pvc
      persistentVolumeClaim:
       claimName: mem-pvc
```

10.9.7. 自动初始化本地存储资源

Kubernetes集群中的本地存储资源一般通过手动或者使用Ansible进行批量初始化，但是这两种方式操作复杂且不适用于大规模集群，而Node Resource Manager组件根据ConfigMap的定义可自动初始化并更新当前节点上的本地存储资源。本文介绍如何通过Node Resource Manager组件实现本地存储资源的自动初始化。

背景信息

Kubernetes集群中的本地存储资源初始化一直是个令人困扰的问题，迄今为止一般手动或者使用Ansible进行批量初始化，但是这两种方式存在以下问题：

- Ansible无法便捷的使用Kubernetes特有的节点信息进行定制化部署，需要提前安装Kubelet，执行 `shell command` 命令以及需要自助解析返回结果。
- 在大规模集群中，手动登录节点部署本地存储资源是很难实现的。
- 无法持续维护初始化的资源，若某个节点需要更新资源，则需要手动登录节点更新资源，且相关存储资源的使用信息无法上报，在Kubernetes调度Pod时无法被利用起来。

基于以上问题可通过Node Resource Manager组件实现本地存储资源的自动初始化。Node Resource Manager组件是一个简单易用的Kubernetes本地存储管理组件，可上报本地存储资源的使用信息，配合Scheduler自动化调度使用本地存储资源的Pod。关于Node Resource Manager组件的更多信息，请参见node-resource-manager。

步骤一：通过ConfigMap定义资源所在节点

通过以下三个参数共同定义资源所在的节点。

```
key: kubernetes.io/hostname
operator: In
value: xxxxx
```

参数	说明
key	匹配Kubernetes Node Labels中的 key 值。
operator	Kubernetes定义的Labels selector operator，主要包含如下四种操作符： • *In*：只有 value 值与节点上Labels Key对应的 value 值相同时匹配。 • *NotIn*：只有 value 值与节点上Labels Key对应的 value 值不相同时匹配。 • *Exists*：只要节点的Labels上存在Key就会匹配。 • *DoesNotExist*：只要节点的Labels上不存在Key就会匹配。
value	匹配Kubernetes Node Labels Key对应的 value 值。

根据需要选择定义资源拓扑的方式，即LVM或Quotapath，进而初始化定义节点上的本地存储资源。

使用LVM定义资源拓扑

通过LVM定义资源拓扑，支持以下三种方式：

- type: device ：通过Node Resource Manager组件所在宿主机上的块设备 devices 进行LVM的声明，声明的块设备会组成一个 volumegroup ， volumegroup 的名称由 name 指定，供后续应用启动时分配Logical Volume。
- type: alibabacloud-local-disk ：通过Node Resource Manager组件所在宿主机上所有的本地盘（选择带有本地盘的ECS类型）共同创建一个名称为 name 值的 volumegroup 。

  > 注意 在使用本地SSD型 (i2)实例创建的ECS实例中，手动挂载的块设备为云盘，非本地盘。

- type: pmem ：通过Node Resource Manager组件所在宿主机上的PMEM资源创建一个名称为 name 值的 volumegroup ，其中 regions 可以指定当前机器上多个PMEM的 region 资源。

根据以下YAML示例，使用LVM定义资源拓扑：

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: unified-resource-topo
 namespace: kube-system
data:
 volumegroup: |-
  volumegroup:
  - name: volumegroup1
    key: kubernetes.io/hostname
    operator: In
    value: cn-zhangjiakou.192.168.XX.XX
    topology:
     type: device
     devices:
     - /dev/vdb
     - /dev/vdc
  - name: volumegroup2
    key: kubernetes.io/nodetype
    operator: NotIn
    value: localdisk
    topology:
     type: alibabacloud-local-disk
  - name: volumegroup1
    key: kubernetes.io/hostname
    operator: Exists
    value: cn-beijing.192.168.XX.XX
    topology:
     type: pmem
     regions:
     - region0
```

使用Quotapath定义资源拓扑

通过Quotapath定义资源拓扑，支持以下两种方式：

- `type: device`：通过Node Resource Manager组件所在宿主机的块设备初始化Quotapath，初始化路径是`name`字段定义的值。
- `type: pmem`：通过Node Resource Manager组件所在宿主机的PMEM设备初始化Quotapath，初始化路径是`name`字段定义的值。

根据以下YAML示例，使用Quotapath定义资源拓扑：

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: unified-resource-topo
 namespace: kube-system
data:
 quotapath: |-
  quotapath:
  - name: /mnt/path1
    key: kubernetes.io/hostname
    operator: In
    value: cn-beijing.192.168.XX.XX
    topology:
     type: device
     options: prjquota
     fstype: ext4
     devices:
     - /dev/vdb
  - name: /mnt/path2
    key: kubernetes.io/hostname
    operator: In
    value: cn-beijing.192.168.XX.XX
    topology:
     type: pmem
     options: prjquota,shared
     fstype: ext4
     regions:
     - region0
```

相关参数说明如下表所示：

参数	说明
options	在挂载块设备时使用的参数。
fstype	格式化块设备使用的文件系统，默认使用 ext4 。
devices	挂载使用的块设备，可声明多个块设备。在多个块设备的情况下，将依次尝试挂载，直到挂载成功。

步骤二：部署Node Resource Manager组件

通过在集群中创建DaemonSet工作负载部署Node Resource Manager组件，YAML示例如下所示：

```
cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: node-resource-manager
 namespace: kube-system
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: node-resource-manager
rules:
```

```
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "watch", "list", "delete", "update", "create"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: node-resource-manager-binding
subjects:
  - kind: ServiceAccount
    name: node-resource-manager
    namespace: kube-system
roleRef:
  kind: ClusterRole
  name: node-resource-manager
  apiGroup: rbac.authorization.k8s.io
---
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: node-resource-manager
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: node-resource-manager
  template:
    metadata:
      labels:
        app: node-resource-manager
    spec:
      tolerations:
        - operator: "Exists"
      priorityClassName: system-node-critical
      serviceAccountName: node-resource-manager
      hostNetwork: true
      hostPID: true
      containers:
        - name: node-resource-manager
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          image: registry.cn-hangzhou.aliyuncs.com/acs/node-resource-manager:v1.18.8.0-983ce56-aliyun
          imagePullPolicy: "Always"
          args:
            - "--nodeid=$(KUBE_NODE_NAME)"
          env:
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: spec.nodeName
          volumeMounts:
```

```
        volumeMounts:
        - mountPath: /dev
          mountPropagation: "HostToContainer"
          name: host-dev
        - mountPath: /var/log/
          name: host-log
        - name: etc
          mountPath: /host/etc
        - name: config
          mountPath: /etc/unified-config
      volumes:
      - name: host-dev
        hostPath:
          path: /dev
      - name: host-log
        hostPath:
          path: /var/log/
      - name: etc
        hostPath:
          path: /etc
      - name: config
        configMap:
          name: node-resource-topo
EOF
```

部署完成之后，组件会自动将ConfigMap中的配置应用到节点上。如果更新ConfigMap，组件会在一分钟内对资源更新。

> 注意 为了保障数据安全，Node Resource Manager组件不会删除任何资源。

联系方式

若您在使用该功能时有任何疑问或优化建议，请使用钉钉扫描下面的二维码或搜索钉钉群号33936810加入钉钉群进行反馈。



相关文档

- 本地存储卷的初始化、挂载与自动扩容

10.10. AEP非易失性存储卷

10.10.1. AEP非易失性数据卷概述

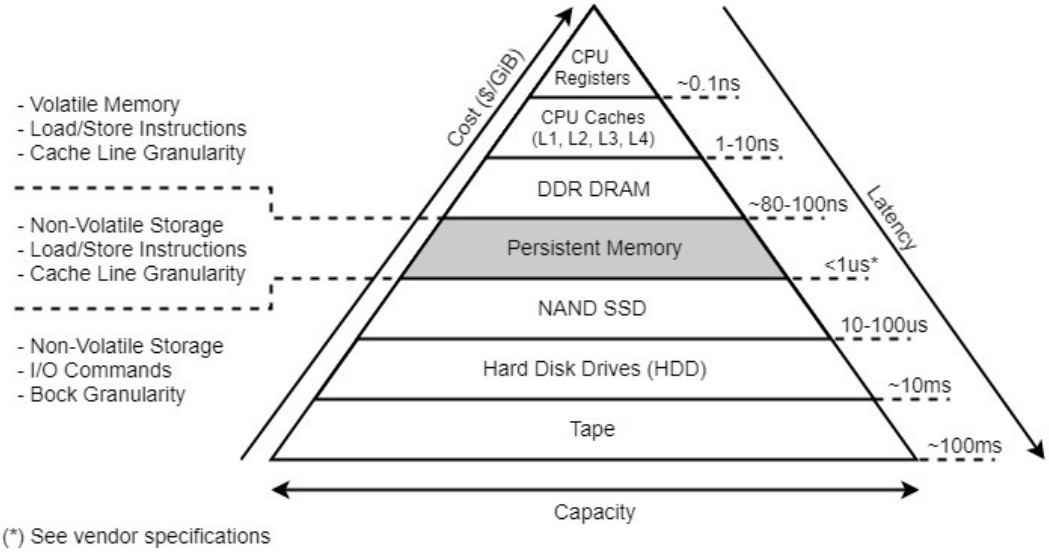
过去内存和存储产品一直受到密度、性能和成本的限制。AEP（Apache Pass），由Intel推出的一种PMEM（持久化内存是一类高性能、数据可持久、内存存储的统称）产品，通过经济地扩展内存容量，增加对持久数据的低延迟访问，从而将一流的内存和存储特性集成于一个产品中。本文介绍ACK中使用AEP非易失性存储的方式和使用示例。

AEP介绍

近年来CPU技术发展迅速，单核CPU计算频率增加、多核CPU技术的出现让计算能力得到长足发展。相比之下，作为数据载体的存储介质，却没有跟上发展的速度：内存（DRAM）可以提供优异的访问性能，但掉电数据丢失，价格昂贵影响了对大容量内存使用场景，而非易失性存储（SSD、磁盘）在时延上无法满足极致性能计算的需求。为此业界一直在探索既能满足CPU计算性能要求、又能保证数据掉电不丢失属性的产品。

既兼顾非易失性存储，又接近内存的使用方式和速度的新型硬件Intel Optane DC Persistent Memory（Apache Pass - AEP ）的发布标志持久化内存Persistent Memory（PMEM）技术已经进入成熟阶段，并开始进入到各种数据加速领域。

下图为AEP的PMEM（Persistent Memory）在存储层次结构中的位置。



ACK中AEP非易失性存储卷的使用方式

目前ACK集群已经支持AEP设备的生命周期管理，通过Alibaba Cloud CSI Driver，您可以以声明式的方式对AEP资源进行分配、挂载、使用。

在ACK中，使用AEP非易失性存储卷的方式有以下两种：

- **PMEP-LVM方式（无侵入的块存储使用方式）**
  您无需修改应用，直接声明即可使用。此方案的原则是把节点上PMEM资源统一成VolumeGroup，然后通过声明PVC类型和容量，使用PMEP-LVM。对无服务器化应用，CICD短生命周期高速临时存储，以及低延迟高吞吐的数据类应用，无需修改应用，即可实现IO性能的2~10倍的提升。
- **PMEM-直接内存（Direct Memory）访问方式**
  为了达到极致的，可以媲美DRAM的访问性能，对应用的内存分配部分函数做有限的修改，可以实现对PMEM设备的直接访问。对内存数据库Redis、HANA等大内存低成本需求，同时满足对TB级别的DRAM和成本的诉求，降低了内存成本30%~50%。

> **说明**
> 逻辑卷管理LVM方式：在ACK环境下，非易失性存储作为块存储或文件系统方式访问，无任何应用侵入和修改，读写性能相比SSD提升2~10倍。
> 直接内存方式：在ACK环境下，非易失性存储作为直接内存访问，需要修改应用适配PMEM SDK做内存分配的代码，达到接近内存访问的吞吐和时延。

PMEM和普通SSD对比

方式	碎片化	在线扩容	持久化	应用修改	时延(4K/RW)	吞吐（4K/RW）	单盘最大容量（ecs.ebmre6p.26xlarge）
PMEM-LVM	无	支持	是	否	10 us	10W	1536 GB
PMEM-Direct	是	否	否	是	1.2 us	56W	768 GB
SSD	无	支持	是	否	100 us	1W	32 TB

插件部署

ACK对非易失性内存支持方案的系统组件包含：

- CSI-Plugin：实现PMEM设备初始化，卷的具体创建、删除操作，实现挂载、卸载操作。
- CSI-Provisioner：感知并发起卷的创建、删除操作。
- CSI-Scheduler：容量调度（ACK调度器内置）。

> **说明**
> 在部署CSI-Plugin时，您需注意：
> - 在节点加上标签 `pmem.csi.alibabacloud.com` 后才会启用AEP设备自运维。
> - `pmem.csi.alibabacloud.com: lvm` 表示此节点使用LVM方式提供PV卷。
> - `pmem.csi.alibabacloud.com: direct` 表示此节点使用内存访问方式提供PV卷。

1. 创建ACK集群。

   ACK集群中添加AEP的PMEM类型的ECS节点实例，例如：ecs.ebmre6p.26xlarge。具体操作，请参见创建Kubernetes托管版集群。

2. 配置AEP的PMEM节点类型。

   您所添加的AEP节点需要配置上相应的标签，方可启用PMEM CSI的挂载能力。
   给节点添加以下Label。

   ```
   pmem.csi.alibabacloud.com/type: direct
   ```

   或：

   ```
   pmem.csi.alibabacloud.com/type: lvm
   ```

3. 部署CSI PMEM插件。

   a. 部署CSI-Plugin。 ❯
   b. 部署CSI-Provisioner。 ❯
   c. 部署StorageClass。 ❯

使用示例

1. **创建块存储类型数据卷**

   i. 创建PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 annotations:
  volume.kubernetes.io/selected-node: cn-zhangjiakou.192.168.XX.XX
 name: pmem-lvm
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 10Gi
 storageClassName: pmem-lvm
```

   为了定向将PVC调度到某个AEP节点，可以配置 `annotations: volume.kubernetes.io/selected-node` 。

   ii. 创建应用负载。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-lvm
 labels:
  app: nginx-lvm
spec:
 selector:
  matchLabels:
   app: nginx-lvm
 template:
  metadata:
   labels:
    app: nginx-lvm
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    command: ["sh", "-c"]
    args: ["sleep 10000"]
    volumeMounts:
     - name: pmem-pvc
       mountPath: "/data"
   volumes:
    - name: pmem-pvc
      persistentVolumeClaim:
       claimName: pmem-lvm
```

iii. 查看结果。

- 执行以下命令查看创建的PVC。

```
kubectl get pvc
```

预期输出：

```
pmem-lvm      Bound   disk-****  10Gi    RWO       pmem-lvm          10m
```

- 执行以下命令查看创建的Pod。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE
deployment-lvm-5bf65c7f76-jb6nl   1/1   Running  0     10m
```

iv. 执行以下命令进入应用查看挂载路径详情。

```
kubectl exec -ti deployment-lvm-5bf65c7f76-jb6nl sh
df /data
```

预期输出：

```
Filesystem              1K-blocks  Used  Available Use% Mounted on
/dev/mapper/pmemvgregion0-disk--**** 10255636  36888 10202364 1%  /data
```

可见已经动态创建了一个块存储数据卷，并挂载给Pod使用。

1. **创建直接内存类型数据卷**

i. 创建PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 annotations:
  volume.kubernetes.io/selected-node: cn-zhangjiakou.192.168.XX.XX
 name: pmem-direct
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 9Gi
 storageClassName: pmem-direct
```

为了定向将PVC调度到某个直接内存类型的AEP节点，可以配置 `annotations: volume.kubernetes.io/selected-node` 。

ii. 创建应用负载。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-direct
 labels:
  app: nginx-direct
spec:
 selector:
  matchLabels:
   app: nginx-direct
 template:
  metadata:
   labels:
    app: nginx-direct
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    command: ["sh", "-c"]
    args: ["sleep 1000"]
    volumeMounts:
     - name: pmem-pvc
       mountPath: "/data"
   volumes:
    - name: pmem-pvc
      persistentVolumeClaim:
       claimName: pmem-direct
```

iii. 查看结果。

- 执行以下命令查看PVC。

  ```
  kubectl get pvc pmem-direct
  ```

  预期输出：

  ```
  NAME     STATUS  VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
  pmem-direct  Bound   disk-****  9Gi     RWO       pmem-direct   17m
  ```

- 执行以下命令查看Pod。

  ```
  kubectl get pod
  ```

  预期输出：

  ```
  NAME                READY  STATUS   RESTARTS  AGE
  deployment-direct-64d448d96-9znfk  1/1    Running  0      17m
  ```

iv. 执行以下命令进入应用查看挂载目录详情。

```
kubectl exec -ti deployment-direct-64d448d96-9znfk sh
df /data
```

预期输出：

```
Filesystem    1K-blocks  Used   Available  Use%  Mounted on
/dev/pmem0    9076344    36888  9023072    1%    /data
```

可见已经动态创建了一个PMEM数据卷，并挂载给Pod使用。

10.10.2. 使用AEP非易失性存储提升读写性能

在ACK环境下，声明化AEP非易失性存储作为块存储或文件系统方式访问，无任何应用侵入和修改，降低AEP使用复杂度，读写性能相比SSD提升2~10倍。本文主要介绍ACK中AEP（Apache Pass）非易失性存储卷在低延迟高速临时读写及数据密集型业务场景下的使用。

应用场景一：低延迟高速临时读写

低延迟高速临时读写包含以下场景：

- CICD持续集成临时存储
- 小文件遍历搜索
- 高吞吐日志写
- 无服务器化临时文件读写

FIO读写性能测试

本文分别使用AEP PMEM-LVM类型和SSD类型卷创建应用，然后对比测试。

1. 部署YAML文件声明PMEM-LVM类型和SSD类型PVC。

PMEM-LVM类型PVC的YAML文件 | SSD类型PVC的YAML文件

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: csi-pmem-lvm
mountOptions:
- dax
parameters:
 lvmType: striping
 nodeAffinity: "true"
 volumeType: PMEM
provisioner: localplugin.csi.alibabacloud.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pmem-lvm-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 100Gi
 storageClassName: csi-pmem-lvm
```

2. 部署YAML文件挂载存储卷和运行 `fio` 测试。

挂载PMEM-LVM存储卷和运行FIO测试 | 挂载SSD-PVC存储卷和运行FIO测试

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-pmem-lvm
 labels:
  app: pmem-lvm
spec:
 selector:
  matchLabels:
   app: pmem-lvm
 template:
  metadata:
   labels:
    app: pmem-lvm
  spec:
   containers:
   - name: fio-test
    image: registry.cn-hangzhou.aliyuncs.com/eric-dev/sysbench:fio
    command: ["sh", "-c"]
    args: ["sleep 10000"]
    volumeMounts:
     - name: pmem
       mountPath: "/data"
   volumes:
    - name: pmem
      persistentVolumeClaim:
       claimName: pmem-lvm-pvc
```

- 通过 `kubectl exec` 进入测试容器内部，运行 `fio` 测试PMEM-LVM卷的写性能。

  ```
  mount | grep csi
  cd /data
  fio -filename=./testfile -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=psync -bs=4k -size=10G -numjo
  bs=50 -runtime=180 -group_reporting -name=rand_100write_4k
  ```

  预期输出：

  ```
  write: IOPS=92.0k, BW=363MiB/s (381MB/s)(8812MiB/24262msec)
  lat (nsec): min=2054, max=95278, avg=10544.00, stdev=1697.17
  ```

- 通过 `kubectl exec` 进入测试容器内部，运行 `fio` 测试SSD卷的写性能。

  ```
  cd /data
  fio -filename=./testfile -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=psync -bs=4k -size=10G -numjo
  bs=50 -runtime=180 -group_reporting -name=rand_100write_4k
  ```

  预期输出：

  ```
  lat (usec): min=20, max=7168, avg=24.76, stdev=13.97
  write: IOPS=37.3k, BW=146MiB/s (153MB/s)(8744MiB/60001msec)
  ```

可见AEP设备的PMEM-LVM性能是SSD本地盘设备的2~3倍。

卷类型	IOPS	Throughput
PMEM-LVM	92000	381 MB/s

卷类型	IOPS	Throughput
SSD	37000	153 MB/s

应用场景二：数据密集型业务

本文通过使用云盘SSD及AEP方式部署MySQL数据库检测写性能。

使用云盘（SSD）部署MySQL数据库

1. 创建云盘SSD数据卷。
   i. 执行以下命令使用阿里云CSI创建云盘SSD数据卷。

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

   *disk-mysql.yaml*模板示例如下所示：

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
    name: disk-mysql
   spec:
    accessModes:
    - ReadWriteOnce
    resources:
     requests:
       storage: 100Gi
    storageClassName: alicloud-disk-topology
   ```

   ii. 执行以下命令查看创建的SSD存储卷。

   ```
   kubectl get pvc disk-mysql
   ```

   预期输出：

   ```
   NAME     STATUS  VOLUME    CAPACITY  ACCESS MODES  STORAGECLASS       AGE
   disk-mysql  Bound  d-***    100Gi    RWO       alicloud-disk-topology  10h
   ```

2. 使用SSD存储卷创建MySQL数据库实例。
   i. 执行以下命令创建MySQL数据库实例。

   ```
   kubectl apply -f mysql-normal-ssd.yaml
   ```

   *mysql-normal-ssd.yaml*模板示例如下所示：

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
 name: mysql-normal
 labels:
  app: wordpress
spec:
 selector:
  matchLabels:
   app: wordpress
   tier: mysql
 strategy:
  type: Recreate
 template:
  metadata:
   labels:
    app: wordpress
    tier: mysql
  spec:
   hostNetwork: true
   containers:
   - image: mysql:5.6
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
     valueFrom:
      secretKeyRef:
       name: mysql-pass
       key: password
    ports:
    - containerPort: 3306
     name: mysql
    volumeMounts:
    - name: mysql
     mountPath: /var/lib/mysql
   volumes:
    - name: mysql
     persistentVolumeClaim:
      claimName: disk-mysql
---
apiVersion: v1
kind: Secret
metadata:
 name: mysql-pass
type: Opaque
data:
 username: YWRt****
 password: YWRt****
```

ii. 执行以下命令查看创建的MySQL数据库实例。

```
kubectl get pod | grep mysql-normal
```

预期输出：

```
mysql-normal-***    1/1   Running  0     10h
```

使用AEP创建MySQL数据库操作步骤如下。

1. 创建PMEM-LVM存储卷。
   i. 执行以下命令使用阿里云CSI创建PMEM-LVM存储卷。

   ```
   kubectl apply -f csi-pmem-lvm.yaml
   ```

   *csi-pmem-lvm.yaml*模板示例如下所示：

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
    name: pmem-mysql
   spec:
    accessModes:
    - ReadWriteOnce
    resources:
     requests:
       storage: 100Gi
    storageClassName: csi-pmem-lvm
   ```

   ii. 执行以下命令查看创建的PMEM-LVM存储卷。

   ```
   kubectl get pvc pmem-mysql
   ```

   预期输出：

   ```
   NAME     STATUS  VOLUME    CAPACITY  ACCESS MODES  STORAGECLASS     AGE
   pmem-mysql  Bound  d-***    100Gi   RWO       csi-pmem-lvm     10h
   ```

2. 在挂载的PMEM存储卷创建MySQL数据库。
   i. 执行以下命令在挂载的PMEM存储卷创建MySQL数据库。

   ```
   kubectl apply -f mysql-normal-pmem.yaml
   ```

   *mysql-normal-pmem.yaml*模板示例如下所示：

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
 name: mysql-normal
 labels:
  app: wordpress
spec:
 selector:
  matchLabels:
   app: wordpress
   tier: mysql
 strategy:
  type: Recreate
 template:
  metadata:
   labels:
    app: wordpress
    tier: mysql
  spec:
   hostNetwork: true
   nodeName: <NODE name of AEP worker node>
   containers:
   - image: mysql:5.6
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
     valueFrom:
      secretKeyRef:
       name: mysql-pass
       key: password
    ports:
    - containerPort: 3306
     name: mysql
    volumeMounts:
    - name: mysql
     mountPath: /var/lib/mysql
   volumes:
    - name: mysql
     persistentVolumeClaim:
      claimName: pmem-mysql
---
apiVersion: v1
kind: Secret
metadata:
 name: mysql-pass
type: Opaque
data:
 username: YWRt****
 password: YWRt****
```

ii. 执行以下命令查看创建的MySQL数据库实例。

```
kubectl get pod | grep mysql-normal
```

预期输出：

```
mysql-normal-***     1/1    Running  0      10h
```

数据库写性能测试

在Pod中执行以下命令进行数据库性能压测。

```
sysbench /root/sysbench/point_select.lua run --db-driver=mysql --report-interval=1
--mysql-table-engine=innodb --oltp-table-size=10000 --oltp-tables-count=32 --oltp-test-mode=complex
--time=100 --mysql-host=127.0.0.1 --mysql-port=3306 --mysql-user=root --mysql-password=admin
--mysql-db=sbtest --oltp-read-only=on --oltp-simple-ranges=0 --oltp-sum-ranges=0
--oltp-order-ranges=0 --oltp-distinct-ranges=0  --oltp-dist-type=uniform --warmup-time=300
--max-time=30 --max-requests=0 --percentile=99 --num-threads=150
```

测试结果：

卷类型	Insert吞吐量（次/秒）
SSD	49812
PMEM-LVM	122156

可见相比于SSD，使用AEP非易失性存储的数据库写入性能约为其2.5倍。

相关文档

- AEP非易失性数据卷概述

10.10.3. 使用AEP的直接内存访问方式部署Redis数据库

在ACK环境下，通过修改应用内存分配部分函数对PMEM（Persistent Memory）设备直接访问，可以实现接近DRAM访问的吞吐和时延。本文主要介绍如何在大容量内存场景下通过DRAM及PMEM方式部署内存数据库Redis。

背景信息

在大容量内存场景下使用AEP直接内存访问（Direct Memory）方式部署内存数据库Redis的优势：

- 声明化AEP内存访问，降低AEP使用复杂度。
- 同样容量内存节省30%~50%的成本。
- 最小化应用修改。

Redis以直接内存访问方式使用AEP设备调用MMAP对PMEM的连续地址空间进行映射。此场景下需要使用特定的Redis版本，可参考本文示例获取Redis镜像。

通过DRAM部署Redis内存数据库

内存（DRAM）可以提供优异的访问性能，但掉电时数据容易丢失且价格昂贵，不益于大容量内存使用。

1. 执行以下命令部署Redis服务。

   ```
   kubectl apply -f redis-normal.yaml
   ```

   *redis-normal.yaml*模板示例如下所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: redis-normal
 labels:
  app: redis-normal
spec:
 selector:
  matchLabels:
   app: redis-normal
 template:
  metadata:
   labels:
    app: redis-normal
  spec:
   containers:
   - name: redis
    image: registry.cn-hangzhou.aliyuncs.com/plugins/redis:v1-normal
    imagePullPolicy: Always
   resources:
    requests:
     memory: 30Gi
    limits:
     memory: 30Gi
```

2. 执行以下命令查看Redis服务。

```
kubectl get pod | grep redis-normal
```

预期输出：

```
redis-normal-***     1/1    Running  0      4d8h
```

3. 执行以下命令测试Redis写性能。

```
LD_PRELOAD=/usr/local/lib/libmemkind.so redis-benchmark -d 102400 -t set -n 1000000 -r 1000000
```

预期输出：

```
1000000 requests completed in 13.03 seconds
76751.86 requests per second
```

从以上预期输出可以得出：通过DRAM部署的Redis内存数据库，每秒大约处理76751个请求。

通过PMEM部署Redis内存数据库

持久化内存Persistent Memory（PMEM）既兼顾非易失性存储，又接近DRAM的使用方式和速度。

1. 执行以下命令部署Redis内存数据库。

```
kubectl apply -f redis-normal.yaml
```

*redis-normal.yaml*的示例模板如下所示：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pmem-pvc-direct
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 60Gi
 storageClassName: csi-pmem-direct
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: redis-pmem-direct
 labels:
  app: redis-pmem-direct
spec:
 selector:
  matchLabels:
   app: redis-pmem-direct
 template:
  metadata:
   labels:
    app: redis-pmem-direct
  spec:
   containers:
   - name: nginx
    image: registry.cn-hangzhou.aliyuncs.com/plugins/redis:v1-pmem
    imagePullPolicy: Always
    volumeMounts:
     - name: pmem-pvc
       mountPath: "/mnt/pmem"
   volumes:
    - name: pmem-pvc
      persistentVolumeClaim:
       claimName: pmem-pvc-direct
```

2. 执行以下命令查看部署的Redis。

```
kubectl get pod | grep redis-pmem-direct
```

预期输出：

```
redis-pmem-direct-5b469546db-5tk2j  1/1    Running  0      4d9h
```

3. 执行以下命令测试Redis写性能。

```
LD_PRELOAD=/usr/local/lib/libmemkind.so redis-benchmark -d 10240 -t set -n 1000000 -r 1000000
```

预期输出：

```
1000000 requests completed in 19.89 seconds
50266.41 requests per second
```

从以上预期输出可以得出：通过PMEM部署的Redis内存数据库，每秒大约处理50266个请求。

相比DRAM部署Redis内存数据库的写性能，PMEM部署的Redis内存数据库的写性能降低仅约34%，读性能不受影响，但成本下降50%。同时使用AEP机型ecs.ebmre6p.26xlarge 1.5TB的PMEM内存空间，相比于普通内存数据库容量提升了4倍。

10.11. 容器存储监控

10.11.1. 使用csi-plugin组件监控节点侧存储资源

容器服务Kubernetes版ACK（Container Service for Kubernetes）的存储监控可以帮助您在使用云盘和NAS时，通过阿里云Prometheus服务监控当前节点侧存储的使用状况。本文介绍如何使用ACK的节点侧存储监控功能，并举例说明如何配置事件告警。

前提条件

目前只有v1.18.8.46-afb19e46-aliyun版本的csi-plugin同时支持云盘监控功能及NAS监控功能，请确保当前的csi-plugin组件的YAML版本和镜像版本为最新版本。您可以通过以下方式升级csi-plugin至最新版。

- 执行GitHub社区中alibaba-cloud-csi-driver的升级脚本。
- 通过控制台升级。具体操作，请参见通过控制台升级csi-plugin组件。

背景信息

在阿里云ACK集群中通常使用云盘数据卷或NAS数据卷存储数据，通过部署csi-plugin的最新版本（ACK集群默认安装csi-plugin组件）可以将磁盘监控与PVC进行绑定。通过阿里云Prometheus监控可以查看当前PVC的状态、配置容量阈值和IOPS阈值，进行事件告警以感知当前PVC使用情况。

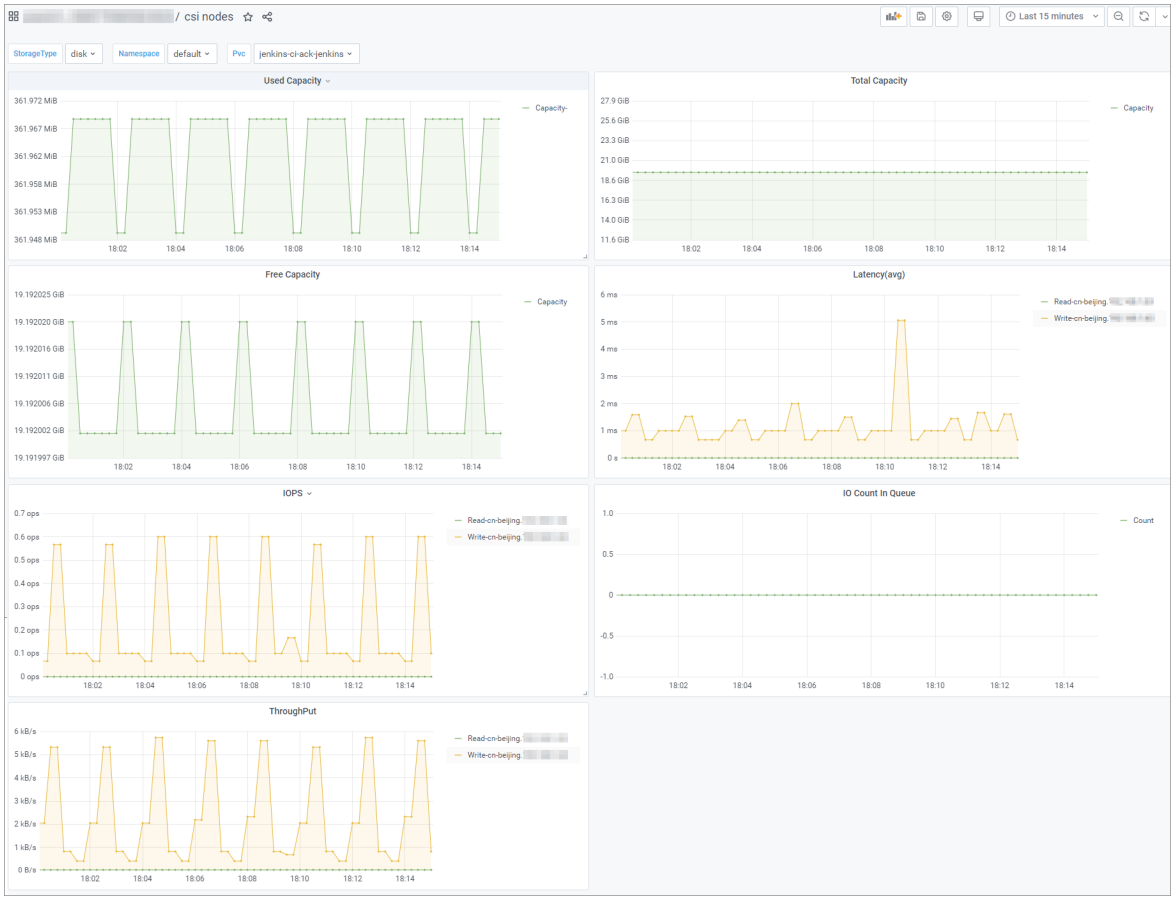
通过控制台升级csi-plugin组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理详情页左侧导航栏，选择**运维管理 > 组件管理**。
5. 单击**存储**页签，在**csi-plugin**组件区域单击**升级**。

存储监控使用说明

目前节点侧监控功能支持云盘及NAS两种存储产品。您可以通过阿里云Prometheus服务监控当前云盘或NAS的使用状况，具体操作如下。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理详情页左侧导航栏，选择**运维管理 > Prometheus监控**。
5. 单击**CSI Nodes**监控大盘名称，进入csi-plugin组件节点侧的监控大盘，查看监控信息。
   监控信息如下图。



CSI Nodes监控大盘过滤项及监控指标说明如下：

- 过滤项：

过滤项	说明
StorageType	存储类型，*disk*表示云盘，*nas*表示NAS文件存储。
Namespace	PVC所在的命名空间
PVC	PVC的名称

- 监控指标：

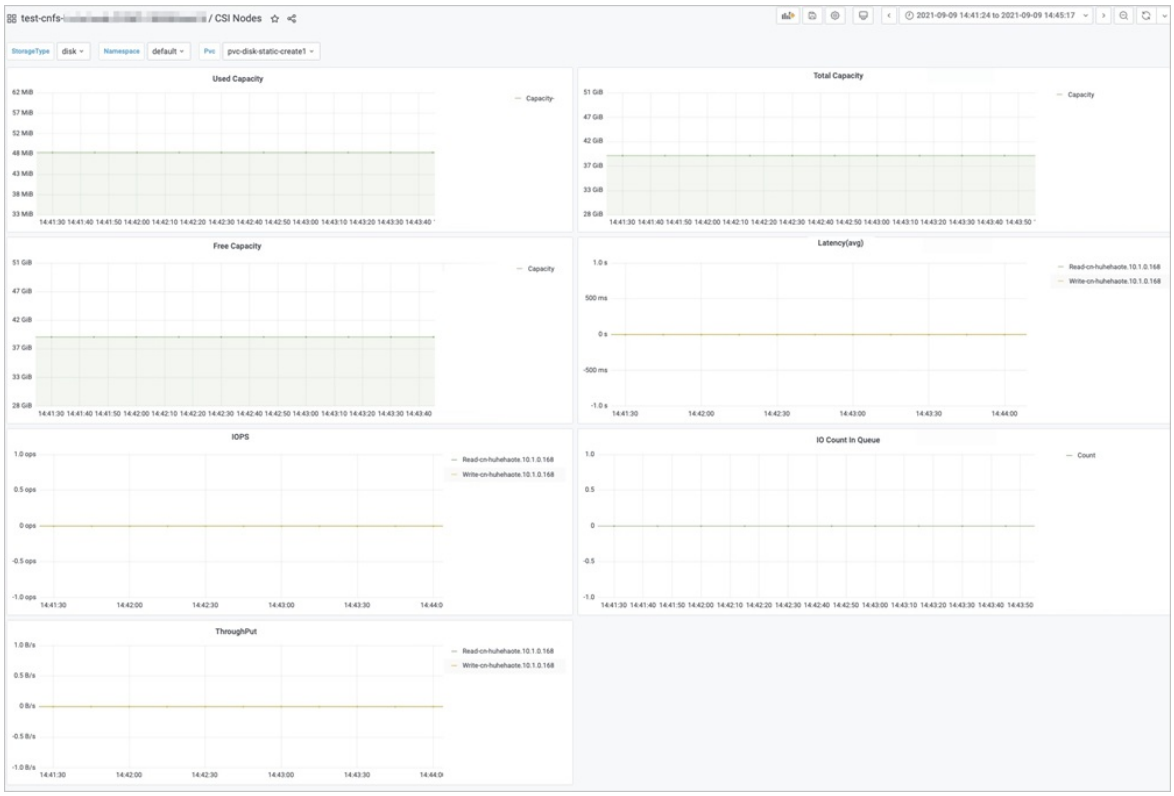
指标名称	说明	显示单位
IOPS	PVC的读写IOPS	ops/sec（每秒读写磁盘次数）
Latency(avg)	PVC的读写请求的平均时延	ms（毫秒）
ThroughPut	PVC的吞吐量	GB
IO Count In Queue	IO请求在磁盘队列的个数	个数
Used Capacity	PVC的磁盘使用空间大小	GB
Total Capacity	PVC的磁盘总空间大小	GB
Free Capacity	PVC的磁盘剩余空间大小	GB

云盘存储监控使用示例

查看云盘存储监控资源需要开启阿里云Prometheus监控。具体操作，请参见开启阿里云Prometheus监控。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理详情页左侧导航栏，选择**运维管理 > Prometheus监控**。
5. 在**Prometheus监控**页面单击**CSI Nodes**页签。
6. 在**CSI Nodes**页签，选择云盘存储卷所在的**StorageType**、**Namespace**及PVC名称后，在**Total Capacity**区域可查看云盘存储卷当前容量。

   本示例**StorageType**选择为**disk**、**Namespace**选择为**default**、PVC为**pvc-disk-static-create1**。



从上图可得，当前云盘总容量为39.25 GiB，使用了48.2 MiB，剩余39.18 GiB。

7. 向云盘中写入测试数据。
   i. 执行以下命令查看挂载云盘的应用名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME                                    READY  STATUS   RESTARTS  AGE
   deployment-disk-static-create1-67f4b5cfdc-p5q7t  1/1    Running  0      20m
   ```

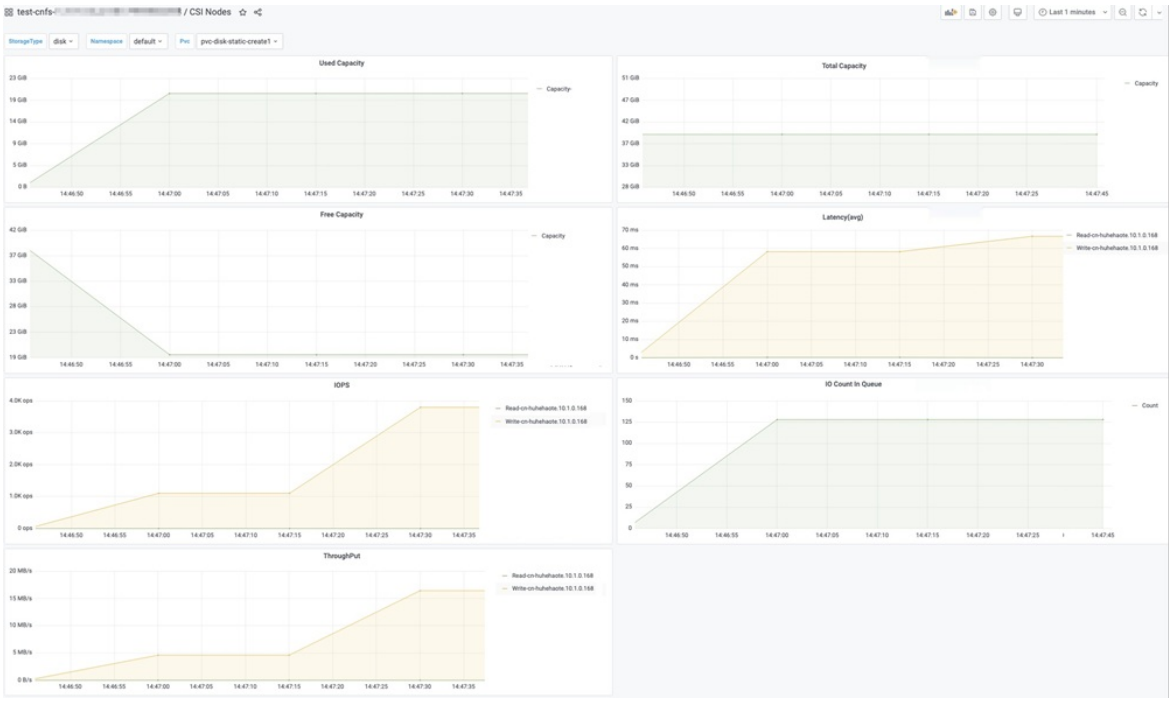
   ii. 执行以下命令进入挂载云盘的应用。

   ```
   kubectl exec deployment-disk-static-create1-67f4b5cfdc-p5q7t -ti sh
   ```

iii. 执行以下命令向云盘挂载目录*/data*写入20 GiB数据。

```
cd /data
/usr/bin/fio -direct=1 -iodepth=256 -rw=randwrite -ioengine=libaio -bs=4k -numjobs=1 -runtime=100 -grou
p_reporting -name=Rand_Write_Testing -filename=./test -size=20G
```

8. 在**Prometheus监控**页面的**CSI Nodes**页签，再次查看云盘存储监控资源。



从上图可得，写入的IOPS为4 k，吞吐率为15 MB/s，写入数据容量为20 GiB，剩余容量为19 GiB，平均延时约为67 ms。

NAS存储监控使用示例

查看NAS存储监控资源需要开启阿里云Prometheus监控。具体操作，请参见开启阿里云Prometheus监控。

1. 向NAS存储卷挂载目录写入测试数据。
   i. 执行以下命令查看挂载NAS存储卷的应用名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME                          READY   STATUS    RESTARTS   AGE
   cnfs-nas-deployment-84f6cdf6cc-wvjjq   1/1     Running   0          20m
   ```

   ii. 执行以下命令进入挂载NAS存储卷的应用。

   ```
   kubectl exec cnfs-nas-deployment-84f6cdf6cc-wvjjq -ti sh
   ```

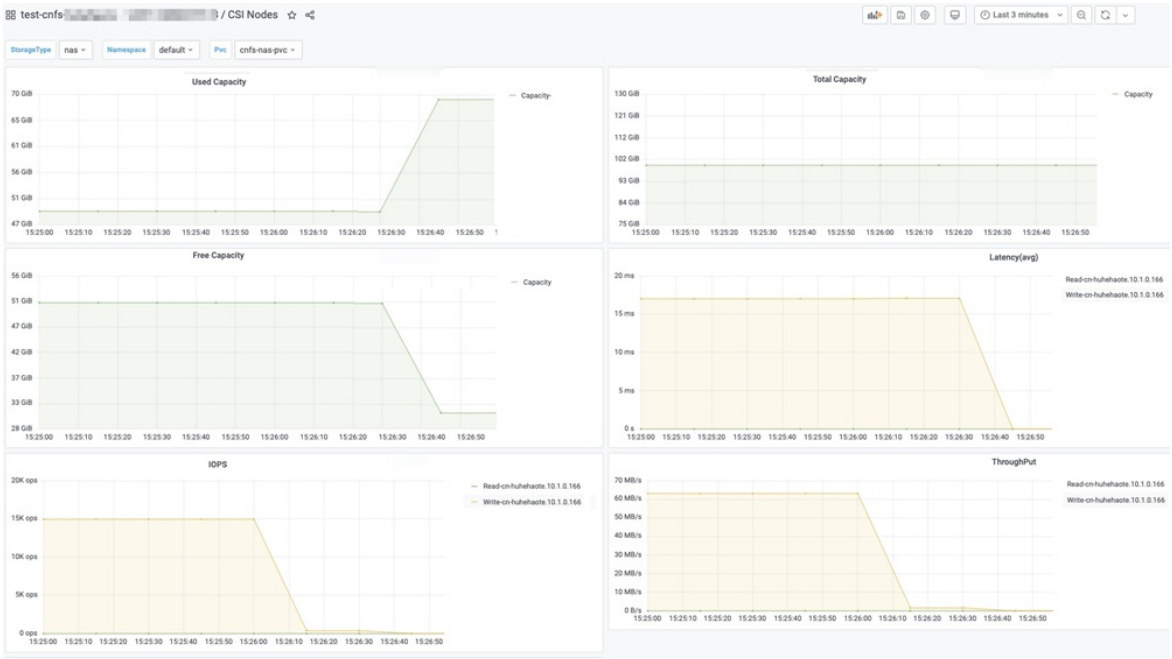
   iii. 执行以下命令向NAS存储卷挂载目录*/data*写入20 GiB数据。

   ```
   cd /data
   /usr/bin/fio -direct=1 -iodepth=256 -rw=randwrite -ioengine=libaio -bs=4k -numjobs=1 -runtime=100 -grou
   p_reporting -name=Rand_Write_Testing -filename=./test -size=20G
   ```

2. 登录容器服务管理控制台。
3. 在控制台左侧导航栏中，单击**集群**。

4. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
5. 在集群管理详情页左侧导航栏，选择**运维管理 > Prometheus监控**。
6. 在**Prometheus监控**页面单击**CSI Nodes**页签。
7. 在**CSI Nodes**页签，选择NAS存储卷所在的**StorageType**、**Namespace**及PVC名称后，在**Total Capacity**区域可查看NAS存储卷当前容量。

   本示例**StorageType**选择为**nas**、**Namespace**选择为**default**、**PVC**为**cnfs-nas-pvc**。



从上图可得，写入的IOPS约为15 k，吞吐率从10 MB/S增长到60 MB/s，平均延时约为15 ms。

10.11.2. 使用storage-operator组件监控集群侧存储资源

在阿里云ACK集群中通过部署storage-operator组件可以将PV及PVC不同状态的分布、变化趋势、异常的PV和PVC的详细信息展示出来。本文介绍如何通过storage-operator组件监控集群侧存储资源。

前提条件

您已创建一个ACK集群。具体操作，请参见创建Kubernetes托管版集群。

步骤一：安装storage-operator组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 单击**存储**页签，在storage-operator组件区域，单击**安装**。

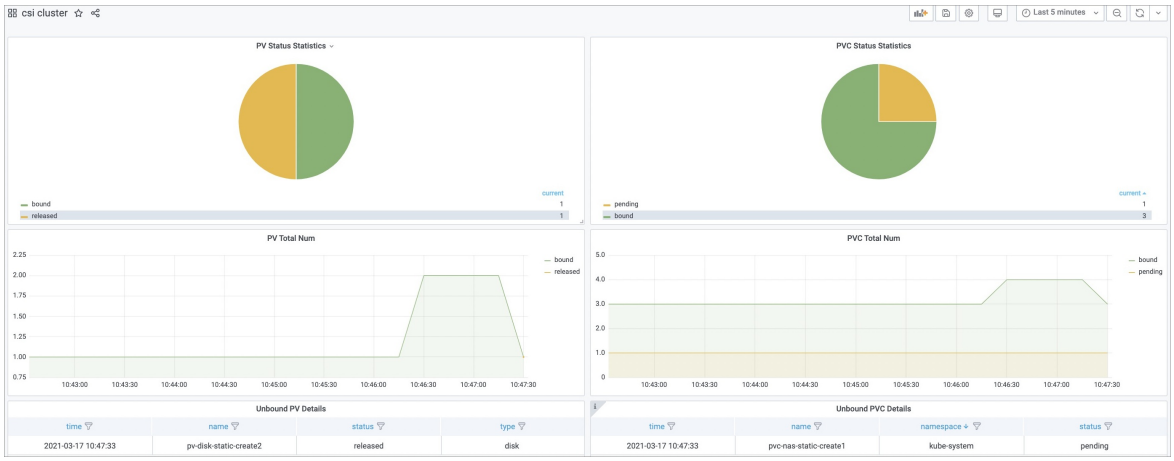
步骤二：查看监控信息

目前该监控功能仅支持PV和PVC监控，您可以通过阿里云Prometheus服务监控当前PV和PVC的使用状况。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > Prometheus监控**。

   若提示组件未安装，请先单击**开始安装**完成组件安装。
5. 单击**CSI Cluster**监控大盘页签，进入csi-plugin组件集群侧的监控大盘，查看监控信息。



CSI Cluster监控大盘指标说明如下所示：

指标名称	说明	显示维度
PV Status Statistics	PV不同状态分布	数量以及占比
PVC Status Statistics	PVC不同状态分布	数量以及占比
PV Total Num	PV不同状态变化趋势	数量
PVC Total Num	PVC不同状态变化趋势	数量
Unbound PV Details	非Bound状态的PV详细信息（支持根据关键字过滤）	详细信息（time-name-status-type）
Unbound PVC Details	非Bound状态的PVC详细信息（支持根据关键字过滤）	详细信息（time-name-namespace-status）

10.12. 容器存储运维

10.12.1. 使用storage-operator进行存储组件的部署与升级

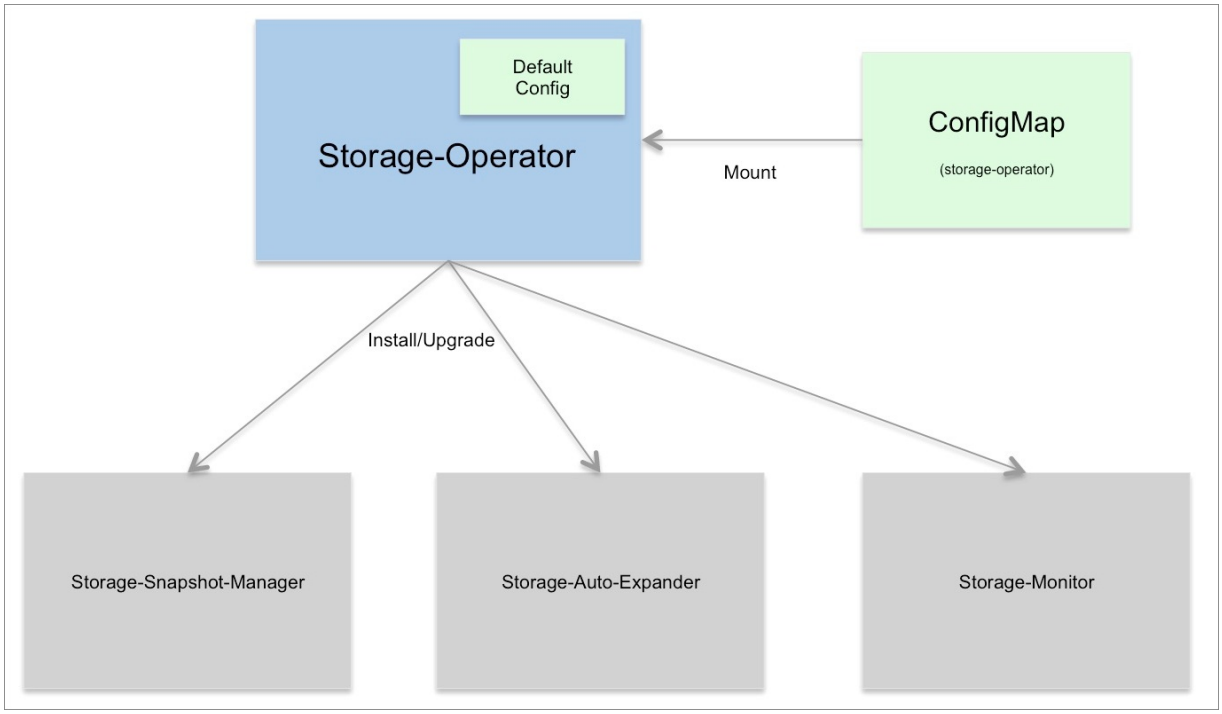
storage-operator组件负责阿里云ACK存储组件的安装与升级。组件的具体配置，可以通过修改ConfigMap来修改。本文档介绍如何配置组件信息及部署storage-operator组件。

组件介绍

storage-operator组件用于管理存储组件的生命周期。storage-operator部署为Deployment，依赖自身镜像中的默认组件配置和通过ConfigMap自定义的组件配置，进行存储组件的部署与升级操作，进而降低容器研发与运维的复杂性。

- 默认组件配置：每个发布的storage-operator组件都有对应的默认配置。
- 通过ConfigMap自定义配置：通过ConfigMap自定义组件配置，可配置版本信息、是否部署组件等信息。

storage-operator会优先使用自定义的组件配置，只有没有指定具体配置时，才使用镜像默认配置。



说明

- storage-operator镜像发布时，都会有默认存储组件配置文件。
- storage-operator运行时，会挂载一个对存储组件配置的ConfigMap文件，且读取该配置。
- storage-operator会整合自定义配置和默认配置，综合判断是否部署与升级组件。

步骤一：配置组件信息

方式一：镜像默认配置

目前默认的存储组件有：storage-snapshot-manager、storage-analyzer、storage-auto-expander及storage-monitor。

每个版本的storage-operator都有默认的组件配置，以v1.18.8.0版本为例，配置示例如下：

```
{
 "storage-snapshot-manager": {
  "install": "true",
  "imageTag": "v1.18.8.0-81508da-aliyun",
  "imageRep": "acs/storage-snapshot-manager",
  "template": "/acs/templates/storage-snapshot-manager/install.yaml"
 },
 "storage-analzyer": {
  "install": "false",
  "imageTag": "v1.18.8.0-06c5560-aliyun",
  "imageRep": "acs/storage-analyzer",
  "template": "/acs/templates/storage-analyzer/install.yaml"
 },
 "storage-auto-expander": {
  "install": "false",
  "imageTag": "v1.18.8.0-4852fd4-aliyun",
  "imageRep": "acs/storage-auto-expander",
  "crdTmpl": "/acs/templates/storage-auto-expander/crd.yaml",
  "template": "/acs/templates/storage-auto-expander/install.yaml"
 },
 "storage-monitor": {
  "install": "true",
  "imageTag": "v1.18.8.0-c4744b6-aliyun",
  "imageRep": "acs/storage-monitor",
  "template": "/acs/templates/storage-monitor/install.yaml",
  "svcTmpl": "/acs/templates/storage-monitor/service.yaml"
 }
}
```

默认配置中各组件定义的参数说明如下：

参数	说明
install	表示该组件是否需要部署，默认为 `false` 。
imageTag	表示如果部署该组件，默认使用的镜像版本。
imageRep	表示如果部署该组件，默认使用的镜像名。
crdTmpl	表示是否需要部署CRD，该值表示CRD模板地址。为空时表示不部署CRD。
svcTmpl	表示是否需要部署Service，该值表示Service模板地址。
template	表示该组件的部署模板地址。

> **说明** 各组件的参数可以在配置（ConfigMap）中修改。

方式二：自定义配置（ConfigMap）

ConfigMap示例模板如下：

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: storage-operator
 namespace: kube-system
data:
 storage-snapshot-manager: |
  # deploy config
  install:  false
  imageTag:  v1.16.aaaa
  imageRep:  acs/storage-snapshot-manager
  template:  /acs/templates/storage-snapshot-manager/install.yaml
  # env config
  SNAPSHOT_INTERVAL: 30
 storage-analyzer: |
  # deploy config
  install:  false
  imageTag:  v1.16.bbbb
  imageRep:  acs/storage-analyzer
  template:  /acs/templates/storage-analyzer/install.yaml
  # env config
  LOOP_INTERVAL: 30
 storage-auto-expander: |
  # deploy config
  install:  false
  imageTag:  v1.16.3adsadfs
  imageRep:  acs/storage-auto-expander
  crdTmpl:  /acs/templates/storage-auto-expander/crd.yaml
  template:  /acs/templates/storage-auto-expander/install.yaml
  # env config
  POLLING_INTERVAL_SECONDS: 60
```

自定义配置需要注意以下几点：

- 每个组件的配置分为两部分：
  - 部署配置参数，即示例中deploy config部分参数，是定义该组件是否部署及部署的版本信息等。storage_operator根据该部分信息安装、升级组件。
  - 环境配置参数，即示例中env config部分参数，是给对应的存储组件使用。例如：`SNAPSHOT_INTERVAL` 配置为storage-snapshot-manager的环境变量。
- 若自定义了部署配置部分的值，将覆盖默认配置中的对应值。
- 部署配置参数是组件部署配置保留字段，不能用作环境变量。
- 可针对于相应组件的特定参数自定义配置。

步骤二：部署storage-operator组件

1. 执行以下命令部署storage-operator组件。

   ```
   kubectl apply -f storage_operator.yaml
   ```

   部署storage-operator组件的示例模板*storage_operator.yaml*如下所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: storage-operator
 namespace: kube-system
 labels:
  app: storage-operator
spec:
 selector:
  matchLabels:
   app: storage-operator
 template:
  metadata:
   labels:
    app: storage-operator
  spec:
   tolerations:
    - operator: "Exists"
   priorityClassName: system-node-critical
   serviceAccount: storage-operator-admin
   containers:
   - name: storage-operator
    image: registry.cn-hangzhou.aliyuncs.com/acs/storage-operator:v1.18.8.28-18cca7b-aliyun
    imagePullPolicy: Always
    volumeMounts:
    - mountPath: /acs/configmap/
      name: storage-operator
   restartPolicy: Always
   volumes:
   - configMap:
      name: storage-operator
     name: storage-operator
```

2. 执行以下命令查看storage-operator组件是否启动。

```
kubectl get pods -nkube-system | grep storage-operator
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE     IP          NODE          NOMINATED NODE  READINESS GAT
ES
storage-operator-***    1/1   Running  0      6d20h    192.168.XX.XX   virtual-kubelet  <none>      <none>
```

10.13. 存储FAQ-CSI

本文为您介绍存储常见问题的分析流程，云盘存储卷与NAS存储卷常见问题的处理方法。

类型	问题
常见问题分析流程	常见问题分析流程

类型	问题
云盘存储卷常见问题	• 使用云盘时提示The specified disk is not a portable disk • 启动挂载了云盘的Pod时提示had volume node affinity conflict • 启动挂载了云盘的Pod时提示can't find disk • 动态创建PV失败且提示The specified AZone inventory is insufficient • 动态创建PV失败且提示disk size is not supported • 磁盘阻塞导致Pod长时间无法启动 • 启动Pod时出现FailedMount警告 • Pod卸载失败且Kubelet出现不受ACK管理的Pod日志 • Pod删除后无法重新启动，提示挂载失败且无法自动恢复。
NAS存储卷常见问题	• 使用NAS时，提示chown: Operation not permitted • 使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV • NAS存储卷挂载时间延长
OSS存储卷常见问题	OSS存储卷挂载时间延长

常见问题分析流程

通过以下操作查看对应存储插件日志文件，明确问题现象。

1. 执行以下命令查看PVC或Pod上是否有明显的Event。

   ```
   kubectl get events
   ```

   预期输出：

   ```
   LAST SEEN  TYPE    REASON          OBJECT                                  MESSAGE
   2m56s      Normal  FailedBinding   persistentvolumeclaim/data-my-release-mariadb-0      no persistent volumes available for this claim and no storage class is set
   41s        Normal  ExternalProvisioning  persistentvolumeclaim/pvc-nas-dynamic-create-subpath8  waiting for a volume to be created, either by external provisioner "nasplugin.csi.alibabacloud.com" or manually created by system administrator
   3m31s      Normal  Provisioning    persistentvolumeclaim/pvc-nas-dynamic-create-subpath8  External provisioner is provisioning volume for claim "default/pvc-nas-dynamic-create-subpath8"
   ```

2. 执行以下命令查看集群是否部署了Flexvolume或CSI存储插件。
   - 执行以下命令查看集群是否部署了Flexvolume存储插件。

     ```
     kubectl get pod -nkube-system |grep flexvolume
     ```

     预期输出：

     ```
     NAME              READY  STATUS       RESTARTS  AGE
     flexvolume-***       4/4   Running      0      23d
     ```

   - 执行以下命令查看集群是否部署了CSI存储插件。

     ```
     kubectl get pod -nkube-system |grep csi
     ```

     预期输出：

```
NAME           READY  STATUS      RESTARTS  AGE
csi-plugin-***       4/4   Running      0      23d
csi-provisioner-***    7/7   Running      0      14d
```

3. 查看使用的数据卷模板是否和集群存储插件（Flexvolume或CSI）的模板匹配。

   若在此集群中初次使用数据卷，请检查您的PV对象及StorageClass对象中定义的Driver名称属于CSI还是Flexvolume。Driver名称必须与当前集群部署的插件类型一致。

4. 查询存储插件是否为最新版本。
   - 执行以下命令查询Flexvolume插件的镜像版本。

     ```
     kubectl get ds flexvolume -nkube-system -oyaml | grep image
     ```

     预期输出：

     ```
     image: registry.cn-hangzhou.aliyuncs.com/acs/Flexvolume:v1.14.8.109-649dc5a-aliyun
     ```

     关于Flexvolume插件信息，请参见Flexvolume。
   - 执行以下命令查询CSI插件的镜像版本。

     ```
     kubectl get ds csi-plugin -nkube-system -oyaml |grep image
     ```

     预期输出：

     ```
     image: registry.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.18.8.45-1c5d2cd1-aliyun
     ```

     关于CSI插件信息，请参见csi-plugin及csi-provisioner。
5. 查看日志。
   - 若云盘PVC处于Pending状态，无法成功创建PV，则需要查看Provisioner日志。
     - 若集群部署了Flexvolume插件，则执行以下命令查询alicloud-disk-controller日志。

       ```
       podid=`kubectl get pod -nkube-system | grep alicloud-disk-controller | awk '{print $1}'`
       kubectl logs <PodID> -nkube-system
       ```

     - 若集群部署了CSI插件，则执行以下命令查询csi-provisioner日志。

       ```
       podid=`kubectl get pod -nkube-system | grep csi-provisioner | awk '{print $1}'`
       kubectl logs <PodID> -nkube-system -c csi-provisioner
       ```

       说明 csi-provisioner的容器组数量为2，上述 kubectl get pod -nkube-system | grep csi-provisioner | awk '{print $1}' 命令操作会输出2个 podid，请分别执行 kubectl logs <PodID> -nkube-system -c csi-provisioner 命令。
   - 若Pod启动时挂载报错，则需要查看Flexvolume或csi-plugin日志。
     - 若集群部署了Flexvolume插件，则执行以下命令查询Flexvolume日志。

       ```
       kubectl get pod <pod-name> -owide
       ```

       登录Pod所在节点，在 /var/log/alicloud/flexvolume_**.log 查看Flexvolume日志。
     - 若集群部署了CSI插件，则执行以下命令查询csi-plugin日志。

       ```
       nodeID=`kubectl get pod <pod-name> -owide | awk 'NR>1 {print $7}'`
       podID=`kubectl get pods -nkube-system -owide -lapp=csi-plugin | grep $nodeID|awk '{print $1}'`
       kubectl logs <PodID> -nkube-system
       ```

   - 查看Kubelet日志。

执行以下命令获取Pod所在节点。

```
kubectl get pod <pod-name> -owide | awk 'NR>1 {print $7}'
```

登录到该节点上查看*/var/log/message*日志文件。

快速恢复

众多Pod挂载不上数据卷场景的问题，可以通过将Pod调度到其他节点快速恢复。具体操作，请参见调度应用Pod至指定节点。

11.存储-Flexvolume

11.1. 存储Flexvolume概述

容器服务ACK支持自动绑定阿里云云盘、阿里云文件存储NAS（Network Attached Storage）、阿里云对象存储OSS（Object Storage Service）存储服务。本文介绍支持的存储服务和数据卷的情况。
容器服务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PV/PVC使用方式	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插件通过PV/PVC使用 • 通过Kubernetes的NFS驱动使用	支持
阿里云OSS	OSS静态存储卷支持PV/PVC使用方式	不支持

11.2. 安装与升级Flexvolume组件

创建阿里云Kubernetes 1.16之前版本的集群时，若存储插件选择为Flexvolume，则控制台默认安装Flexvolume与Disk-Controller组件，但不会默认安装alicloud-nas-controller组件。本文介绍如何对Flexvolume组件升级管理及如何安装alicloud-nas-controller组件。

前提条件

- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 阿里云Kubernetes集群存储插件为Flexvolume。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。

注意事项

若集群中部署了alicloud-nas-controller组件，在将集群升级至Kubernetes 1.20版本之前需要将alicloud-nas-controller的镜像升级到v1.14.8.17-7b898e5-aliyun或更高版本。

> **说明** 如果您没有使用阿里云提供的alicloud-nas-controller组件，而是使用社区的相关方案（例如：nfs-provisioner），您需要到社区找到相应的升级方案来避免selfLink问题。

使用限制

目前支持CentOS 7、Alibaba Cloud Linux 2操作系统。

安装组件

安装Flexvolume组件

- 对于阿里云Kubernetes 1.16及之后版本的集群，不再支持Flexvolume组件的安装，请使用CSI-Plugin组件。更多信息，请参见CSI和Flexvolume存储插件的区别。
- 对于阿里云Kubernetes 1.16之前版本的集群，在创建集群时，若存储插件选择为Flexvolume，则控制台会默认安装Flexvolume组件。具体操作，请参见组件配置。

安装Disk-Controller组件

- 对于阿里云Kubernetes 1.16及之后版本的集群，不再支持Disk-Controller组件的安装，请使用CSI-Provisioner组件。更多信息，请参见CSI和Flexvolume存储插件的区别。

- 对于阿里云Kubernetes 1.16之前版本的集群，在创建集群时，若存储插件选择为Flexvolume，则控制台会默认安装Disk-Controller组件。具体操作，请参见组件配置。

安装alicloud-nas-controller组件

若您的集群存储插件选择为Flexvolume，可以通过手动部署alicloud-nas-controller方式实现动态NAS数据卷。

通过以下YAML模板手动安装。

```
kind: Deployment
apiVersion: apps/v1
metadata:
 name: alicloud-nas-controller
 namespace: kube-system
spec:
 selector:
  matchLabels:
   app: alicloud-nas-controller
 strategy:
  type: Recreate
 template:
  metadata:
   labels:
    app: alicloud-nas-controller
  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: nfs-provisioner
      image: registry.cn-hangzhou.aliyuncs.com/acs/alicloud-nas-controller:v1.14.8.17-7b898e5-aliyun
      env:
      - name: PROVISIONER_NAME
        value: alicloud/nas
      securityContext:
       privileged: true
      volumeMounts:
      - mountPath: /var/log
        name: log
   volumes:
   - hostPath:
     path: /var/log
    name: log
```

验证安装

验证Flexvolume、Disk-Controller与alicloud-nas-controller组件是否成功部署。

- 执行以下命令，查看Flexvolume组件是否成功部署。

```
kubectl get pod -nkube-system | grep flexvolume
```

- 执行以下命令，查看Disk-Controller组件是否成功部署。

```
kubectl get pod -nkube-system | grep alicloud-disk-controller
```

- 执行以下命令，查看alicloud-nas-controller组件是否成功部署。

```
kubectl get pod -nkube-system | grep alicloud-nas-controller
```

升级组件

Flexvolume与Disk Controller组件支持在控制台升级，alicloud-nas-controller组件不支持在控制台升级。

若阿里云Kubernetes 1.16之前版本的集群升级到1.16及之后版本，升级后的集群仍支持Flexvolume存储插件，且可通过控制台升级。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理详情页左侧导航栏，选择**运维管理 > 组件管理**。
5. 单击**存储**页签，在**flexvolume**或**alicloud-disk-controller**组件区域单击**升级**。
6. 在**提示**对话框中确认版本信息后单击**确定**。
   升级成功后，对应组件区域会提示升级成功，且可查看组件当前版本。

- Flexvolume组件升级时，以下场景请联系我们申请手动升级保障。
  - 在线升级失败。
  - Flexvolume组件版本不大于v1.12，且使用了云盘或OSS数据卷。
  - 集群业务敏感，且使用大量数据卷，希望提供升级保障。
- Disk Controller组件升级失败时，请联系我们申请手动升级保障。

联系我们

若您需要申请手动升级保障，请您使用钉钉搜索钉钉群号35532895加入钉钉群咨询。

11.3. 云盘存储卷

11.3.1. 云盘存储卷概述

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，本文主要介绍云盘存储卷的功能介绍、存储规格、适用场景、注意事项、计费说明以及存储类（StorageClass）。

功能介绍

云盘是阿里云为云服务器ECS提供的数据块级别的块存储产品，具有低时延、高性能、持久性、高可靠等特点。云盘采用分布式三副本机制，为ECS实例提供数据可靠性保证。云盘支持在可用区内自动复制您的数据，防止意外硬件故障导致的数据不可用，保护您的业务免于组件故障的威胁。

根据性能分类，云盘包含以下几类产品：

- ESSD云盘：基于新一代分布式块存储架构的超高性能云盘产品，结合25GE网络和RDMA技术，单盘可提供高达100万的随机读写能力和更低的单路时延能力。更多详情，请参见ESSD云盘。
  建议在大型OLTP数据库、NoSQL数据库和ELK分布式日志等场景中使用。
- SSD云盘：具备稳定的高随机读写性能、高可靠性的高性能云盘产品。
  建议在I/O密集型应用、中小型关系数据库和NoSQL数据库等场景中使用。
- 高效云盘：具备高性价比、中等随机读写性能、高可靠性的云盘产品。

建议在开发与测试业务和系统盘等场景中使用。

- 普通云盘：属于上一代云盘产品，已经逐步停止售卖。

存储规格

各类型云盘的性能比较如下表所示。

性能类别	ESSD云盘				SSD云盘	高效云盘	普通云盘
	PL3	PL2	PL1	PL0			
单盘容量范围（GiB）	1261~32768	461~32768	20~32768	40~32768	20~32768	20~32768	5~2000
最大IOPS	1000000	100000	50000	10000	25000	5000	数百
最大吞吐量（MB/s）	4000	750	350	180	300	140	30~40
单盘IOPS性能计算公式	min{1800+50*容量,1000000}	min{1800+50*容量,100000}	min{1800+50*容量,50000}	min{1800+12*容量,10000 }	min{1800+30*容量,25000}	min{1800+8*容量,5000}	无
单盘吞吐量性能计算公式（MB/s）	min{120+0.5*容量,4000}	min{120+0.5*容量,750}	min{120+0.5*容量,350}	min{100+0.25*容量,180}	min{120+0.5*容量,300}	min{100+0.15*容量,140}	无
单路随机写平均时延（ms），Block Size=4K	0.2			0.3~0.5	0.5~2	1~3	5~10
API参数取值	cloud_essd				cloud_ssd	cloud_efficiency	cloud

云盘更多性能介绍请参见块存储性能。

适用场景

根据业务需求，您可以对云盘做以下操作：

业务需求	参考链接
存储应用数据	具体操作，请参见： • 使用云盘静态存储卷 • 通过命令行使用动态云盘卷 • 通过控制台使用动态云盘卷
系统盘或数据盘容量不足	关于扩容的更多信息，请参见扩容概述。 具体操作，请参见： • 在ACK中实现Flexvolume云盘在线扩容 • 手动扩容云盘数据卷

业务需求	参考链接
持久化存储	具体操作，请参见： • 使用云盘静态存储卷实现持久化存储-Flexvolume • 使用云盘动态存储卷实现持久化存储-Flexvolume

注意事项

-
- 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，且不能保证挂载、卸载的优先顺序。使用Deployment时由于升级策略，可能出现重启Pod时新的Pod一直无法挂载，故不推荐使用Deployment。
-
-
- 云盘类型和ECS类型需要匹配才可以挂载，否则会挂载失败。关于云盘类型和ECS类型的匹配关系，请参考实例规格族。
- 每个节点最多可挂载16块云盘，单块云盘容量最大32 TiB。

计费说明

- 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。
- 云盘的具体价格信息，请参见详细定价。

更多信息，请参见计费。

存储类（StorageClass）

如果PVC请求多，就需要创建很多PV，对于运维人员来说维护成本很高，Kubernetes提供一种自动创建PV的机制，叫StorageClass。

StorageClass

容器服务Kubernetes版（ACK）集群默认提供了以下几种StorageClass：

- 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SSD云盘；如果SSD云盘售尽，则创建高效云盘。

  > **注意**
  > - 对于alicloud-csi-provisioner v1.14.8.39-0d749258-aliyun之前的版本，优先创建ESSD云盘。
  > - 如果ESSD云盘售尽，则创建SSD云盘。
  > - 如果SSD云盘售尽，则创建高效云盘。

- alicloud-disk-topology：使用延迟绑定的方式创建云盘。

以上5种StorageClass中，前4种可以在单可用区使用，最后一种更适合在多可用区使用。

通过StorageClass创建云盘，可用区选择规则如下：

- StorageClass配置volumeBindingMode: WaitForFirstConsumer，则取PVC的Pod所在可用区为创建云盘的可用区。
- StorageClass配置volumeBindingMode: Immediate，且配置zoneId参数（一个可用区），则选择此可用区为创建云盘的可用区。

- StorageClass配置volumeBindingMode: Immediate，且配置zoneId参数（多个可用区），则每次选择轮询配置的多个可用区中的一个作为创建云盘的可用区。
- StorageClass配置volumeBindingMode: Immediate，且没有配置zoneId参数，则选择csi-provisioner所在的可用区作为创建云盘的可用区。

综上，如果您是多可用区集群，推荐您使用WaitForFirstConsumer模式的StorageClass。您可以根据需要的云盘类型自行创建StorageClass。

Default StorageClass

Kubernetes提供Default StorageClass机制，您在PVC不指定StorageClass的情况下，可以通过Default StorageClass创建数据卷，请参见Default StorageClass。

> 说明
> - 由于Default StorageClass会对所有PVC起作用，对于具备不同类型存储卷能力的集群，需要小心使用。例如，您想生成一个NAS类型PVC、PV，并绑定PVC和PV，但可能因为有Default StorageClass而自动创建了云盘PV。基于上述原因，ACK集群没有提供Default StorageClass，如果您期望使用Default StorageClass，可以参考以下配置。
> - 一个集群中最多配置一个Default StorageClass，否则默认能力将不起作用。

1. 配置Default StorageClass。

   执行以下命令将StorageClass （alicloud-disk-ssd）配置为一个Default StorageClass。

   ```
   kubectl patch storageclass alicloud-disk-ssd -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
   ```

   这时查询集群中的StorageClass可以看到alicloud-disk-ssd名字后面加了(default)字样。

   ```
   kubectl get sc
   ```

   返回结果如下：

   ```
   NAME                 PROVISIONER            AGE
   alicloud-disk-ssd (default)  diskplugin.csi.alibabacloud.com  96m
   ```

2. 使用Default StorageClass。

   i. 使用以下模板创建一个没有配置StorageClass的PVC。

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
    name: disk-pvc
   spec:
    accessModes:
    - ReadWriteOnce
    resources:
     requests:
       storage: 20Gi
   ```

   集群会自动创建一个云盘卷（PV），且配置了Default StorageClass（alicloud-disk-ssd）。

   ```
   kubectl get pvc
   ```

   返回结果如下：

   ```
   NAME    STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS     AGE
   disk-pvc  Bound  d-bp18pbai447qverm3ttq  20Gi    RWO       alicloud-disk-ssd  49s
   ```

您可以通过以下命令取消默认存储类型配置。

```
kubectl patch storageclass alicloud-disk-ssd -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

11.3.2. 使用云盘静态存储卷

本文介绍如何通过PV和PVC方式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。具体操作，请参见创建云盘。

> 说明

通过PV和PVC方式使用云盘存储卷

1. 创建云盘类型的PV。

   您可以使用YAML文件或者控制台创建云盘类型的PV。

   - 通过YAML文件创建PV。
     a. 使用以下内容创建*disk-pv.yaml*文件。

     ```
     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。

     b. 执行以下命令创建PV。

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

   - 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. 在控制台左侧导航栏中，单击**集群**。
     c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
     d. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。
     e. 在**存储卷**页面，单击右上角的**创建**。

f. 在**创建存储卷**对话框中，配置数据卷的相关参数。

参数名	描述
存储卷类型	本示例中为云盘。
存储驱动	本示例中为**Flexvolume**。
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	您可以选择以哪种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。

g. 完成配置后，单击**创建**。

2. 创建PVC。

i. 使用以下内容创建*disk-pvc.yaml*文件。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-disk
spec:
 accessModes:
  - ReadWriteOnce
 storageClassName: disk
 resources:
  requests:
   storage: 20Gi
```

ii. 执行以下命令创建PVC。

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

3. 创建Pod。

i. 使用以下内容创建*disk-pod.yaml*文件。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 ports:
 - port: 80
  name: web
 clusterIP: None
 selector:
  app: nginx
---
apiVersion: apps/v1
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
```

ii. 执行以下命令创建Pod。

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

11.3.3. 通过命令行使用动态云盘卷

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。

创建指定zoneId的StorageClass

1. 创建并复制以下内容到*storage-class.yaml*中。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: alicloud-disk-ssd-hangzhou-b
provisioner: alicloud/disk
parameters:
 type: cloud_ssd
 regionId: cn-hangzhou
 zoneId: cn-hangzhou-b
reclaimPolicy: Retain
```

参数说明如下。

参数	描述
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_efficiency 、 cloud_ssd 、 cloud_essd 、 available 四种参数，其中 available 会对ESSD、SSD、高效云盘依次尝试创建，直到创建成功。
regionId	期望创建云盘的地域。
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。

2. 执行以下命令，创建StorageClass。

```
kubectl apply -f storage-class.yaml
```

创建延迟绑定的StroageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-disk-topology-ssd
provisioner: alicloud/disk
parameters:
   type: cloud_ssd
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

> 说明
> - 如果配置的StorageClass没有 WaitForFirstConsumer ，且没有配置 zoneid ，这时创建的PV和Disk-Controller组件所在节点的Zone是一样的。
> - 如果配置的StorageClass没有 WaitForFirstConsumer ，但是配置了 zoneid ，这时创建的PV会根据 zoneid 的配置，轮询使用其中的 zoneid 。
> - 如果使用WaitForFirstConsumer，会根据消费此PVC的Pod所调度的节点创建云盘，即在Pod调度的可用区创建云盘。

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。

集群默认提供了以下几种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SSD云盘；如果SSD云盘售尽，则创建高效云盘。

> 注意 对于alicloud-disk-controller v1.14.8.44-c23b62c5-aliyun之前的版本，优先创建ESSD云盘；如果ESSD云盘售尽，则创建SSD云盘；如果SSD云盘售尽，则创建高效云盘。

- alicloud-disk-topology：使用延迟绑定的方式创建云盘。

使用云盘创建多实例StatefulSet

使用volumeClaimTemplates的方式来创建，这样会动态创建多个PVC和PV并绑定。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 ports:
 - port: 80
  name: web
 clusterIP: None
 selector:
  app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
  matchLabels:
   app: nginx
 serviceName: "nginx"
 replicas: 2
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
    ports:
    - containerPort: 80
     name: web
    volumeMounts:
    - name: disk-ssd
     mountPath: /data
 volumeClaimTemplates:
 - metadata:
   name: disk-ssd
  spec:
   accessModes: [ "ReadWriteOnce" ]
   storageClassName: "alicloud-disk-ssd-hangzhou-b"
   resources:
    requests:
     storage: 20Gi
```

您也可以通过控制台的方式使用动态云盘卷，请参见通过控制台使用动态云盘卷。

11.3.4. 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。

前提条件

您已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。

步骤一：创建StorageClass

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储类**。
5. 在**存储类**页面单击**创建**，在**创建**对话框中设置参数。
   - **名称**：存储类的名称。
   - **存储卷类型**：本示例中选中**云盘**。
   - **存储驱动**：本示例中选中**Flexvolume**。
   - **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_efficiency、cloud_ssd、cloud_essd、available四种参数，其中available会对ESSD、SSD、高效云盘依次尝试创建，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。
6. 参数配置完成后，单击**创建**。

步骤二：创建PVC

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
5. 在**存储声明**页面，单击右上角的**创建**。在弹出的**创建存储声明**对话框中设置参数。
   - **存储声明类型**：包含**云盘**、**NAS**、**OSS**三种类型。本示例选择**云盘**。
   - **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**：包含**使用存储类动态创建**、**已有存储卷**、**创建存储卷**三种模式。本示例选择**使用存储类动态创建**。
   - **已有存储类**：单击**选择存储类**，在**选择存储类**对话框中，目标存储类右侧操作列单击**选择**。
   - **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云盘容量。

   - **访问模式**：默认为ReadWriteOnce。
6. 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。



步骤三：创建应用并增加云存储声明

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 有状态**。
5. 在**有状态**页面中，单击**使用镜像创建**。
6. 配置创建有状态应用信息。

   本例主要为您介绍存储卷的配置。关于其他参数的配置，请参见创建有状态工作负载StatefulSet。
   支持增加本地存储、增加云存储声明（PersistentVolumeClaim）或增加存储声明模板。
   本例中配置了一个云盘类型的存储卷，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。



7. 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，请参见通过命令行使用动态云盘卷。

11.3.5. 使用云盘静态存储卷实现持久化存储-Flexvolume

当容器发生宕机故障时，有状态服务容器存储的业务数据存在着丢失和不可靠等风险。使用持久化存储可以解决该问题。本文介绍如何使用云盘静态存储卷实现持久化存储。

前提条件

请确保您已完成以下操作：

- 创建Kubernetes托管版集群
- 创建云盘
- 通过kubectl工具连接集群

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。请参见使用云盘静态存储卷。

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
```

验证结果

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv. 在集群管理页左侧导航栏中，单击**存储 > 存储卷**，可以看到刚刚创建的PV。



创建PVC

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
```

验证结果

i. 登录容器服务管理控制台。
ii. 在控制台左侧导航栏中，单击**集群**。
iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
iv. 在集群管理页左侧导航栏中，单击**存储 > 存储声明**。
v. 在**存储声明**页面，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，创建静态应用。

```
kubectl create -f static.yaml
```

验证结果

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

v. 在**无状态**页面，可以看到刚刚创建的静态应用。

静态云盘的持久化存储

1. 执行以下命令，查看部署的静态应用所在Pod的名称。

```
kubectl get pod | grep static
```

预期输出：

```
nginx-static-78c7dcb9d7-g****  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-g**** df | grep data
```

预期输出：

```
/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-g**** ls /data
```

预期输出：

```
lost+found
```

4. 执行以下命令，在*/data*路径下创建文件*static*。

```
kubectl exec nginx-static-78c7dcb9d7-g**** touch /data/static
```

5. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec nginx-static-78c7dcb9d7-g**** ls /data
```

预期输出：

```
static
lost+found
```

6. 执行以下命令，删除名称为 `nginx-static-78c7dcb9d7-g****` 的Pod。

```
kubectl delete pod nginx-static-78c7dcb9d7-g****
```

预期输出：

```
pod "nginx-static-78c7dcb9d7-g****" 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
```

预期输出：

```
NAME                     READY   STATUS            RESTARTS   AGE
nginx-static-78c7dcb9d7-g****   2/2     Running           0          50s
nginx-static-78c7dcb9d7-g****   2/2     Terminating       0          72s
nginx-static-78c7dcb9d7-h****   0/2     Pending           0          0s
nginx-static-78c7dcb9d7-h****   0/2     Pending           0          0s
nginx-static-78c7dcb9d7-h****   0/2     Init:0/1          0          0s
nginx-static-78c7dcb9d7-g****   0/2     Terminating       0          73s
nginx-static-78c7dcb9d7-h****   0/2     Init:0/1          0          5s
nginx-static-78c7dcb9d7-g****   0/2     Terminating       0          78s
nginx-static-78c7dcb9d7-g****   0/2     Terminating       0          78s
nginx-static-78c7dcb9d7-h****   0/2     PodInitializing   0          6s
nginx-static-78c7dcb9d7-h****   2/2     Running           0          8s
```

8. 执行以下命令，查看Kubernetes重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME                     READY   STATUS    RESTARTS   AGE
nginx-static-78c7dcb9d7-h****   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
```

预期输出：

```
static
lost+found
```

11.3.6. 使用云盘动态存储卷实现持久化存储-Flexvolume

当容器发生宕机故障时，有状态服务容器存储的业务数据存在着丢失和不可靠等风险。使用持久化存储可以解决该问题。本文介绍如何使用云盘动态存储卷实现持久化存储。

背景信息

动态云盘的使用场景：

没有购买云盘，在应用部署时自动购买云盘的情况。

动态云盘的使用方式：

1. 手动创建PVC，在PVC中声明StorageClass。
2. 部署应用时通过StorageClass自动创建PV。

前提条件

- 已创建Kubernetes集群，请参见创建Kubernetes托管版集群。
- 已通过kubectl连接到Kubernetes集群，请参见步骤二：选择集群凭证类型。
- 集群中已安装Provisioner插件，该插件可以根据StorageClass自动创建云盘。

Provisioner插件

容器服务Kubernetes版在创建集群时，默认安装Provisioner插件。

创建StorageClass

阿里云容器服务Kubernetes在系统初始化的时候会默认创建4个StorageClass，且使用参数的默认情况。同时这4个StorageClass仅适用于单可用区集群，若是多可用区集群，则需要您创建StorageClass。这4个StorageClass分别为：

- *alicloud-disk-common*：自动创建普通云盘。
- *alicloud-disk-efficiency*：自动创建高效云盘。
- *alicloud-disk-ssd*：自动创建SSD云盘。
- *alicloud-disk-available*：提供高可用选项，先尝试自动创建SSD云盘；如果相应可用区的SSD云盘资源售尽，再尝试自动创建高效云盘。

1. 创建*storageclass.yaml*文件。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: alicloud-disk-ssd-hangzhou-b
provisioner: alicloud/disk
reclaimPolicy: Retain
parameters:
  type: cloud_ssd
  regionId: cn-hangzhou
  zoneId: cn-hangzhou-b
  fstype: "ext4"
  readonly: "false"
```

参数	描述
provisioner	动态云盘配置为alicloud/disk，标识使用provisioner插件自动创建阿里云云盘。

参数	描述
reclaimPolicy	云盘的回收策略。支持*Delete*和*Retain*，默认情况为*Delete*。 **说明** 如果配置为*Delete*，删除PVC时，云盘会一起删除，云盘上的数据不可恢复。
type	自动创建云盘的类型，支持*cloud*、*cloud_efficiency*、*cloud_ssd*、*available*。
regionId	（可选）自动创建云盘所在的地域，与集群的地域相同。
zoneId	（可选）自动创建云盘所在的区域。 ○ 单可用区集群，与集群所在区域相同。 ○ 多可用区集群，zoneid可同时配置多个，例如： `zoneid: cn-hangzhou-a,cn-hangzhou-b,cn-hangzhou-c`
fstype	（可选）自动创建云盘所使用的文件系统，默认情况为*ext4*。
readonly	（可选）挂载自动创建云盘的权限是否为可读，支持*true*：云盘具有只读权限，*false*：云盘具有可读可写权限，默认情况为*false*。
encrypted	（可选）自动创建的云盘是否加密，支持*true*：加密，*false*：不加密，默认情况为*false*。

2. 执行以下命令，创建StorageClass。

```
kubectl create -f storageclass.yaml
```

创建PVC

1. 创建*pvc-ssd.yaml*文件。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: disk-ssd
spec:
 accessModes:
  - ReadWriteOnce
 storageClassName: alicloud-disk-ssd-hangzhou-b
 resources:
  requests:
   storage: 20Gi
```

2. 执行以下命令，创建PVC。

```
kubectl create -f pvc-ssd.yaml
```

预期结果

在目标**集群**的页面下单击左侧导航栏的**存储 > 存储声明**，可以看到该PVC绑定的存储类型为 StorageClass 中声明的*alicloud-disk-ssd-hangzhou-b*，并且关联存储卷。

创建应用

1. 创建*pvc-dynamic.yaml*文件。

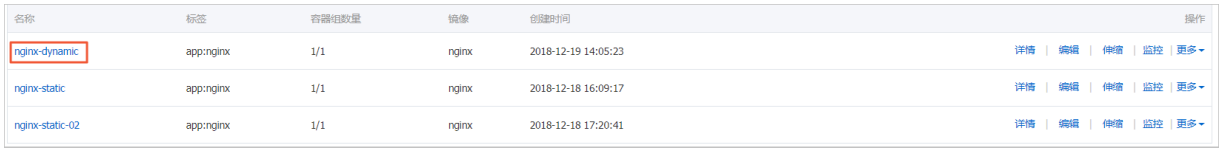
```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-dynamic
 labels:
  app: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
    volumeMounts:
     - name: disk-pvc
       mountPath: "/data"
   volumes:
    - name: disk-pvc
      persistentVolumeClaim:
       claimName: disk-ssd
```

2. 执行以下命令，创建Deployment。

```
kubectl create -f nginx-dynamic.yaml
```

预期结果

在目标**集群**页面下单击左侧导航栏的**工作负载 > 无状态**，可以看到刚刚创建的Deployment。



动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 dynamic
```

预期输出：

```
nginx-dynamic-5c74594ccb-zl9pf   2/2   Running   0      3m
```

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-dynamic-5c74594ccb-zl9pf df | grep data
```

预期输出：

```
/dev/vdh     20511312   45080 20449848  1% /data
```

3. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /data
```

预期输出：

```
lost+found
```

4. 执行以下命令，在*/data*路径下创建文件*dynamic*。

```
kubectl exec nginx-dynamic-5c74594ccb-zl9pf touch /data/dynamic
```

5. 执行以下命令，查看*/data*路径下的文件。

```
kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /data
```

预期输出：

```
dynamic
lost+found
```

6. 执行以下命令，删除名称为 `nginx-dynamic-5c74594ccb-zl9pf` 的Pod。

```
kubectl delete pod nginx-dynamic-5c74594ccb-zl9pf
```

预期输出：

```
pod "nginx-dynamic-5c74594ccb-zl9pf" 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
```

预期输出：

```
NAME                     READY  STATUS   RESTARTS  AGE
nginx-dynamic-5c74594ccb-zl9pf   2/2   Running  0     6m48s
nginx-dynamic-5c74594ccb-zl9pf  2/2  Terminating  0   7m32s
nginx-dynamic-5c74594ccb-45sd4  0/2  Pending  0   0s
nginx-dynamic-5c74594ccb-45sd4  0/2  Pending  0   0s
nginx-dynamic-5c74594ccb-45sd4  0/2  Init:0/1  0   0s
nginx-dynamic-5c74594ccb-zl9pf  0/2  Terminating  0   7m32s
nginx-dynamic-5c74594ccb-zl9pf  0/2  Terminating  0   7m33s
nginx-dynamic-5c74594ccb-zl9pf  0/2  Terminating  0   7m33s
nginx-dynamic-5c74594ccb-45sd4  0/2  PodInitializing  0   5s
nginx-dynamic-5c74594ccb-45sd4  2/2  Running  0   22s
```

8. 执行以下命令，查看Kubernetes重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME                     READY  STATUS   RESTARTS  AGE
nginx-dynamic-5c74594ccb-45sd4   2/2   Running   0     2m
```

9. 执行以下命令，查看*/data*路径下的文件，刚刚创建的文件*dynamic*并没有被删除，说明动态云盘的数据可持久保存。

```
kubectl exec nginx-dynamic-5c74594ccb-45sd4 ls /data
```

预期输出：

```
dynamic
lost+found
```

11.3.7. 在ACK中实现Flexvolume云盘在线扩容

在Kubernetes 1.16版中，云盘数据卷在线扩容已经进入Beta阶段，ACK支持在Kubernetes 1.16及以上版本中实现Flexvolume云盘在线扩容。本文介绍在Flexvolume环境中如何进行云盘的动态扩容，从而增加云盘空间。

背景信息

云盘数据卷扩容包括**云盘扩容**和**文件系统扩容**两部分，云盘和文件系统都支持应用在线扩容（设备挂载到目录状态下扩容），但是从文件系统扩容最佳实践角度，最稳定的扩容方案是先停止应用层服务、解除挂载目录，再进行数据卷扩容。

相关概念

- 在线扩容
  指在业务Pod不暂停的情况下，实现云盘及其文件系统的扩容。
- 离线扩容
  指在业务Pod暂停的情况下，完成云盘数据卷与文件系统扩容后，再启动业务Pod。

从Kubernetes 1.16开始默认实现了Pod不重启即可以实现数据卷的扩容。

使用说明

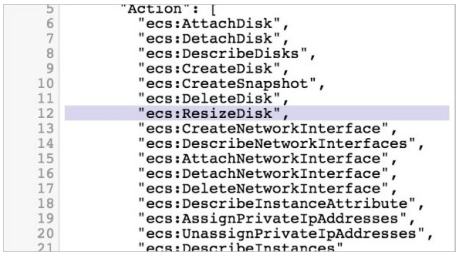
- **使用限制**
  仅支持在线扩容2000 GiB以下容量的云盘。
- **数据备份**
  扩容数据卷前，先备份云盘快照，以防扩容过程异常导致数据出现问题。
- **适用范围**
  - 只有动态存储卷才可以进行数据卷动态扩容，即配置了StorageClassName的PVC。
  - 不支持扩容InlineVolume类型（非PV、PVC方式）的云盘数据卷。
  - 不支持动态扩容普通云盘类型。
  - StorageClass需要配置为**AllowVolumeExpansion: True**。ACK集群默认提供的StorageClass为True，自建的StorageClass需要自行配置。
- **插件版本**
  升级您使用Flexvolume或者CSI插件到最新版本。

为RAM用户添加ResizeDisk权限

实现云盘在线扩容前，您需要为集群的**RAM角色**添加ResizeDisk权限。针对不同集群和插件类型，添加ResizeDisk权限的步骤如下：

专有集群（CSI插件）

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选中目标集群，并在目标集群右侧**操作**列下，单击**详情**。
4. 在集群管理页左侧导航栏中，单击**集群信息**。
5. 单击**集群资源**页签，单击**Master RAM角色**链接。
6. 在RAM控制台中，添加ResizeDisk权限。添加权限具体步骤，请参见修改自定义策略内容。



专有集群（Flexvolume插件）、托管集群（所有插件类型）

重复上述前4个步骤，然后在集群管理页面单击**Worker RAM 角色**链接。

不重启容器实现在线扩容

1. 通过kubectl工具连接集群。

   本文假设应用的当前状态如下。
   执行以下命令获取Pod信息。

   ```
   kubectl get pod
   ```

   返回结果如下：

   ```
   web-0    1/1   Running  0     42s
   ```

   执行以下命令查看Pod的挂载详情。

   ```
   kubectl exec web-0 df /data
   ```

   返回结果如下：

   ```
   Filesystem    1K-blocks  Used Available Use% Mounted on
   /dev/vdb      20511312 45080  20449848   1% /data
   ```

   执行以下命令获取PVC信息。

   ```
   kubectl get pvc
   ```

   返回结果如下：

   ```
   NAME        STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS         AGE
   disk-ssd-web-0   Bound    d-wz9hpoifm43yn9zie6gl   20Gi       RWO            alicloud-disk-available   57s
   ```

   执行以下命令获取PV信息。

   ```
   kubectl get pv
   ```

   返回结果如下：

   ```
   NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS
   REASON   AGE
   d-wz9hpoifm43yn9zie6gl   20Gi       RWO            Delete           Bound    default/disk-ssd-web-0   alicloud-disk-avail
   able            65s
   ```

2. 在符合使用说明的各个条件下，执行以下命令进行数据卷扩容。

   ```
   kubectl patch pvc disk-ssd-web-0 -p '{"spec":{"resources":{"requests":{"storage":"30Gi"}}}}'
   ```

   等待一定时间（一分钟以内）后扩容完成，检查状态如下。
   执行以下命令获取PV信息。

   ```
   kubectl get pv d-wz9hpoifm43yn9zie6gl
   ```

返回结果如下：

```
NAME            CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM            STORAGECLASS        REASON  AGE
d-wz9hpoifm43yn9zie6gl  30Gi    RWO       Delete       Bound   default/disk-ssd-web-0  alicloud-disk-available  5m23s
```

执行以下命令获取PVC信息。

```
kubectl get pvc
```

返回结果如下：

```
NAME        STATUS  VOLUME           CAPACITY  ACCESS MODES  STORAGECLASS        AGE
disk-ssd-web-0  Bound   d-wz9hpoifm43yn9zie6gl  30Gi    RWO       alicloud-disk-available  5m10s
```

执行以下命令查看Pod的挂载详情。

```
kubectl exec web-0 df /data
```

返回结果如下：

```
Filesystem    1K-blocks  Used Available Use% Mounted on
/dev/vdb      30832548 45036  30771128   1% /data
```

您只需要执行一条命令即可完成所有的扩容操作。

重启容器实现在线扩容

1. 使用命令行工具连接Kubernetes集群，请参见步骤二：选择集群凭证类型。

   本文假设应用的当前状态如下。
   执行以下命令获取Pod信息。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   web-0     1/1   Running  0      42s
   ```

   执行以下命令查看Pod的挂载详情。

   ```
   kubectl exec web-0 df /data
   ```

   预期输出：

   ```
   Filesystem    1K-blocks  Used  Available Use% Mounted on
   /dev/vdb      20511312  45080 20449848  1%  /data
   ```

   执行以下命令获取PVC信息。

   ```
   kubectl get pvc
   ```

   预期输出：

   ```
   NAME        STATUS  VOLUME           CAPACITY  ACCESS MODES  STORAGECLASS        AGE
   disk-ssd-web-0  Bound   d-wz9hpoifm43yn9zi****  20Gi    RWO       alicloud-disk-available  57s
   ```

   执行以下命令获取PV信息。

   ```
   kubectl get pv
   ```

   预期输出：

```
NAME           CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS   CLAIM           STORAGECLASS
REASON  AGE
d-wz9hpoifm43yn9zi****  20Gi    RWO       Delete      Bound    default/disk-ssd-web-0  alicloud-disk-avail
able       65s
```

2. 执行以下命令查看PV的调度信息。

```
kubectl get pv d-wz9g2j5qbo37r2lamkg4  -oyaml | grep failure-domain.beta.kubernetes.io/zone
  failure-domain.beta.kubernetes.io/zone: cn-shenzhen-e
```

3. 修改调度信息label，在zone后面添加字段，促使这个PV关联的Pod不可调度。

   例如，把以下示例中原来的cn-shenzhen-e改为cn-shenzhen-e-nozone。

```
kubectl label pv d-wz9g2j5qbo37r2lamkg4 failure-domain.beta.kubernetes.io/zone=cn-shenzhen-e-nozone --o
verwrite
persistentvolume/d-wz9g2j5qbo37r2lamkg4 labeled
```

4. 重启Pod。

   由于修改了调度信息，Pod会暂时Pending。
   执行以下命令删除Pod。

```
kubectl delete pod web-0
```

   执行以下命令获取Pod信息。

```
kubectl get pod
```

   返回结果如下：

```
web-0  0/1   Pending  0      27s
```

5. 执行以下命令扩容数据卷。

```
kubectl patch pvc disk-ssd-web-0 -p '{"spec":{"resources":{"requests":{"storage":"30Gi"}}}}'
```

6. 将PV的调度信息label改回去，促使Pod启动完成（把cn-shenzhen-e-nozone改为起初的cn-shenzhen-e）。

```
kubectl label pv d-wz9g2j5qbo37r2lamkg4 failure-domain.beta.kubernetes.io/zone=cn-shenzhen-e --overwrite
persistentvolume/d-wz9g2j5qbo37r2lamkg4 labeled
```

   等待一定时间（一分钟内）后检查扩容是否符合预期。
   执行以下命令获取Pod信息。

```
kubectl get pod
```

   返回结果如下：

```
web-0  1/1   Running  0      3m23s
```

   执行以下命令获取PVC信息。

```
kubectl get pvc
```

   返回结果如下：

```
disk-ssd-web-0  Bound   d-wz9g2j5qbo37r2lamkg4  30Gi    RWO       alicloud-disk-available  17m
```

   执行以下命令获取PV信息。

```
kubectl get pv d-wz9g2j5qbo37r2lamkg4
```

返回结果如下：

```
d-wz9g2j5qbo37r2lamkg4  30Gi    RWO      Delete     Bound   default/disk-ssd-web-0  alicloud-disk-available      17m
```

执行以下命令查看Pod的挂载详情。

```
kubectl exec web-0 df /data
```

返回结果如下：

```
/dev/vdb     30832548 45036  30771128   1% /data
```

从返回结果中可以看出云盘由20 GiB成功被扩容至30 GiB。

11.3.8. 手动扩容云盘数据卷

在容器服务Kubernetes 1.16之前的旧版本中，目前无法实现云盘自动扩容，需要手动扩容。本文介绍如何手动扩容云盘，从而增加云盘空间。

背景信息

相关概念

- 自动扩容
  指只需要修改PVC的Size，而不需手动进行具体扩容操作，即可完成云盘数据卷及文件系统的扩容。
- 手动扩容
  指需要手动扩容云盘数据卷，且手动执行 `resize2fs` 实现文件系统的扩容。
- 在线扩容
  指在业务Pod不暂停的情况下，实现云盘及其文件系统的扩容。
- 离线扩容
  指在业务Pod暂停的情况下，完成云盘卷、文件系统扩容后，再启动业务Pod。

云盘数据卷扩容包括以下三部分：

- 扩容云盘存储空间：需要在云盘控制台操作。
- 扩容文件系统：需要所挂载的云盘在ECS节点手动操作。
- 更新PV及PVC的Size（暂不支持）。

> **说明**
> 由于以下原因，建议在集群升级到高版本后再通过自动扩容方式修改PV及PVC的大小，低版本不修改。
>
> - 不同版本K8s集群更新PV、PVC对象Size值的操作不同。
> - PV、PVC的Size值不会影响底层存储空间的使用，即：虽然PV、PVC存储空间大小为20 GiB，对应云盘与文件系统存储空间为30 GiB，但是Pod依然可以使用30 GiB的存储空间。

基于稳定性需求，提供以下两种扩容的方式：

- 不暂停应用手动在线扩容：文件系统在线扩容时，如果IO流量高，文件系统存在出现IO报错的风险。但应用可以不重启。
- 暂停应用手动扩容：应用停止后，磁盘IO会暂停，文件系统扩容数据更安全。但应用会暂停一段时间。

使用说明

- **使用限制**
  仅支持扩容2000 GiB以下容量的云盘。
- **数据备份**
  扩容数据卷前，先执行云盘快照备份数据，以防扩容过程异常导致数据丢失。

- **适用范围**
  不支持自动扩容的场景：
  - K8s集群版本小于1.16。
  - PV类型为静态云盘数据卷。

示例应用

本文以Web有状态应用为例，介绍两种扩容方式。执行以下操作，查看云盘使用信息。

- 执行以下命令查看Web应用的Pod。

  ```
  kubectl get pod | grep web
  ```

  预期输出：

  ```
  NAME             READY  STATUS   RESTARTS  AGE
  web-0            1/1    Running  0     11h
  web-1            1/1    Running  0     11h
  ```

- 执行以下命令查看Web应用的PVC。

  ```
  kubectl get pvc | grep web
  ```

  预期输出：

  ```
  NAME       STATUS  VOLUME     CAPACITY  ACCESS MODES  STORAGECLASS     AGE
  disk-ssd-web-0  Bound   d-0jlhaq***  20Gi    RWO       alicloud-disk-essd  11h
  disk-ssd-web-1  Bound   d-0jl0j5***  20Gi    RWO       alicloud-disk-essd  11h
  ```

- 执行以下命令查看Web应用的PV。

  ```
  kubectl get pv | grep web
  ```

  预期输出：

  ```
  NAME      CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM          STORAGECLASS    REASON  AG
  E
  d-0jl0j5***  20Gi    RWO      Delete      Bound   default/disk-ssd-web-1  alicloud-disk-essd      11h
  d-0jlhaq***  20Gi    RWO      Delete      Bound   default/disk-ssd-web-0  alicloud-disk-essd      11h
  ```

从以上预期输出可得：示例应用Web使用了20 GiB的 `d-0jl0j5***` 和 `d-0jlhaq***` 云盘，分别挂载在2个Pod上。

关于如何部署Web有状态应用，请参见创建有状态工作负载StatefulSet。

方式一：不暂停应用手动在线扩容

根据PV信息找到对应的云盘，手动对云盘扩容，然后在云盘对应挂载节点手动执行文件系统扩容。以下示例把2个云盘扩容到30 GiB。

步骤一：云盘数据卷扩容

1. 登录ECS管理控制台。
2. 在左侧导航栏，选择**存储与快照 > 云盘**。
3. 找到Web应用对应的云盘*d-0jl0j5****及*d-0jlhaq****，查看云盘状态，并分别在**操作**列单击**更多 > 云盘扩容**。
4. 在**磁盘扩容**页面选中**在线扩容**，并输入扩容后容量，例如30 GiB。

   > **说明** 设置的扩容后容量不允许小于当前容量。

5. 确认费用，阅读并选中《云服务器ECS服务条款》后，单击**确认扩容**。

   具体操作，请参见在线扩容云盘（Linux系统）。

步骤二：文件系统扩容

扩容云盘后需要完成对文件系统的扩容，否则容器内存储空间依然是扩容前的大小。

> **注意** 该步骤是针对于未分区云盘，是K8s中云盘的标准使用方法，不建议在K8s系统中使用分区云盘。
> - 若PV使用未分区云盘，则不能手动执行分区操作，否则将会对当前文件系统产生破坏，造成数据丢失。
> - 若PV使用分区云盘，则需要扩容分区设备后扩容文件系统。具体操作，请参见步骤三：查看云盘分区情况及步骤四：扩容分区。

1. 查看当前云盘挂载节点。
   i. 登录ECS管理控制台。
   ii. 在左侧导航栏，选择**存储与快照 > 云盘**。
   iii. 找到Web应用对应的云盘*d-0jl0j5****及*d-0jlhaq****，分别单击云盘名称。
   iv. 在**云盘详情**页面的**云盘挂载信息**列表下单击**所挂载的实例**。
   v. 在**实例详情**页签查看实例**网络信息**。

   > **说明** 也可通过在K8s中查看该云盘对应Pod所在的节点。具体操作，请参见查看容器组（Pod）。

2. 连接到挂载云盘的ECS实例节点上，获取云盘设备符。

   关于如何连接到ECS实例节点，请参见连接方式概述。
   可通过以下方法获取云盘设备符。
   - 获取云盘设备符。
   - 执行以下命令获取云盘*d-0jlhaq****的设备符。

     ```
     # 查询 {pv-name}
     mount |grep d-0jlhaq***
     ```

     预期输出：

     ```
     /dev/vdc on /var/lib/kubelet/plugins/kubernetes.io/csi/pv/d-0jlhaq***/globalmount type ext4 (rw,relatime)
     /dev/vdc on /var/lib/kubelet/pods/a26d174f-***/volumes/kubernetes.io~csi/d-0jlhaq***/mount type ext4 (rw,relatime)
     ```

     从以上预期输出可得：所挂载的云盘 `d-0jlhaq***` 在当前节点的设备符为*/dev/vdc*。

3. 执行以下命令扩容文件系统。

   ```
   resize2fs /dev/vdc
   ```

   > **说明** */dev/vdc*为步骤2获取的云盘设备符。

   预期输出：

   ```
   resize2fs 1.43.5 (04-Aug-2017)
   Filesystem at /dev/vdc is mounted on /var/lib/kubelet/plugins/kubernetes.io/csi/pv/d-0jlhaq***/globalmount; on-line resizing required
   old_desc_blocks = 3, new_desc_blocks = 4
   The filesystem on /dev/vdc is now 7864320 (4k) blocks long.
   ```

4. 执行以下命令验证文件系统扩容结果。

   ```
   lsblk /dev/vdc
   ```

预期输出：

```
NAME MAJ:MIN  RM SIZE RO TYPE MOUNTPOINT
vdc  254:32   0  30G  0 disk /var/lib/kubelet/pods/a26d174f-***/volumes/kubernetes.io~csi/d-0jlhaq***/mount
```

从以上预期输出可得，*vdc*文件系统存储空间为30 GiB。

方式二：暂停应用手动扩容

通过删除StatefulSet或者将Replica配置为0的方式暂停应用，手动对每个依赖数据盘扩容，然后再启动应用。通过以下示例把2个云盘扩容到30 GiB。

步骤一：删除应用Pod

1. 执行以下命令将Pod缩容为0个副本。

   ```
   kubectl scale sts web --replicas=0
   ```

   预期输出：

   ```
   statefulset.apps/web scaled
   ```

2. 执行以下命令查看Pod是否删除。

   ```
   kubectl get pod | grep web
   ```

   无输出，代表已删除Web应用。

步骤二：云盘数据卷扩容

1. 登录ECS管理控制台。
2. 在左侧导航栏，选择**存储与快照 > 云盘**。
3. 找到Web应用对应的云盘*d-0jl0j5****及*d-0jlhaq****，查看云盘状态，并分别在**操作**列选择**更多 > 云盘扩容**。
4. 在**磁盘扩容**页面配置扩容方式及扩容后容量。
   - 若云盘处于**待挂载**状态，在**磁盘扩容**页面无需选中**在线扩容**，输入扩容后容量，例如30 GiB。
   - 若云盘处于**使用中**状态，在**磁盘扩容**页面选中**在线扩容**，并输入扩容后容量。

   > **说明** 设置的扩容后容量不允许小于当前容量。

5. 确认费用，阅读并选中《云服务器ECS服务条款》后，单击**确认扩容**。

   具体操作，请参见在线扩容云盘（Linux系统）。

步骤三：文件系统扩容

扩容云盘后需要完成对文件系统的扩容，否则容器内存储空间依然是扩容前的大小。

> **注意** 该步骤是针对于未分区云盘，是K8s中云盘的标准使用方法，不建议在K8s系统中使用分区云盘。
> - 若PV使用未分区云盘，则不能手动执行分区操作，否则将会对当前文件系统产生破坏，造成数据丢失。
> - 若PV使用分区云盘，则需要扩容分区设备后扩容文件系统。具体操作，请参见步骤三：查看云盘分区情况及步骤四：扩容分区。

1. （可选）将待挂载云盘挂载到某个ECS节点。

   > **说明** 需要将云盘挂载到ECS节点上才可实现文件系统扩容。

   i. 登录ECS管理控制台。

   ii. 在左侧导航栏，选择**存储与快照 > 云盘**。

iii. 对于**待挂载**状态的云盘，在**操作**列单击**更多 > 挂载**。

iv. 在**挂载云盘**对话框，选择目标实例并设置云盘相关释放行为。

参数	说明
目标实例	选择需要挂载的ECS实例。
云盘随实例释放	选中此选项，释放实例时会自动释放此云盘。 如果未选中，当实例被释放时该云盘会被保留下来。 **说明** 如果您挂载的是从其他实例卸载的系统盘，**云盘随实例释放**中的实例指系统盘被卸载前的源ECS实例，并非当前操作的实例。
自动快照随云盘释放	选中此选项，当云盘释放时该云盘创建的自动快照都会一起释放。建议您不要选择该选项，以便保留备份数据。

v. 单击**执行挂载**。

如果该云盘的状态变为**使用中**，表示挂载成功。

2. 连接到挂载云盘的ECS实例节点上，获取云盘设备符。

   关于如何连接到ECS实例节点，请参见连接方式概述。

   - 执行以下命令获取云盘设备符。

     ```
     for device in `ls /sys/block | grep vd`; do
      cat /sys/block/$device/serial | grep 0jlhaq*** && echo $device;
     done
     ```

     说明 `0jlhaq***` 为扩容云盘ID `d-0jlhaq***` 中 `d-` 后的字符串。

   - （可选）若上述操作无法获取到云盘设备符，请按以下操作获取。

     a. 将目标云盘卸载后执行 `ls /dev/vd*` 命令查看云盘列表。

     b. 将目标云盘挂载后执行 `ls /dev/vd*` 命令查看云盘列表。

     c. 比较上述获取的云盘列表，多出来的云盘即为目标云盘。

3. 执行以下命令扩容文件系统。

   ```
   resize2fs /dev/vdb
   ```

   说明 */dev/vdb*为步骤2获取的云盘设备符。

   预期输出：

   ```
   resize2fs 1.43.5 (04-Aug-2017)
   Resizing the filesystem on /dev/vdb to 7864320 (4k) blocks.
   The filesystem on /dev/vdb is now 7864320 (4k) blocks long.
   ```

4. 验证文件系统扩容结果。

   i. 执行以下命令创建临时文件夹*/mnt/disk/*，并将云盘挂载到临时文件夹上。

   ```
   mkdir /mnt/disk
   mount /dev/vdb /mnt/disk/
   ```

ii. 执行以下命令查看文件系统大小。

```
df /mnt/disk/
```

预期输出：

```
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdb       30832548 45036  30771128   1% /mnt/disk
```

从以上预期输出可得：临时文件夹*/dev/vdb*的内存为30 GiB，说明已经扩容成功。

iii. 执行以下命令将云盘从临时文件夹卸载。

```
umount /mnt/disk
```

步骤四：重启应用

1. 执行以下命令将Pod扩容为2个副本。

   ```
   kubectl scale sts web --replicas=2
   ```

   预期输出：

   ```
   statefulset.apps/web scaled
   ```

2. 执行以下命令查看Pod是否删除。

   ```
   kubectl get pod | grep web
   ```

   预期输出：

   ```
   NAME    READY  STATUS   RESTARTS  AGE
   web-0   1/1    Running  0     74s
   web-1   1/1    Running  0     42s
   ```

3. 执行以下命令查看文件系统大小。

   ```
   kubectl exec web-0 df /data
   ```

   预期输出：

   ```
   Filesystem     1K-blocks  Used Available Use% Mounted on
   /dev/vdb       30832548 45036  30771128   1% /data
   ```

   从以上预期输出可得，*/dev/vdb*文件系统存储空间为30 GiB。

常见问题

问题现象：

在执行 `resize2fs` 命令之后出现如下错误提示。

```
resize of device /dev/xxx failed: exit status 1 resize2fs output: resize2fs xxx(version)
Please run `e2fsck -f /dev/xxx` first
```

问题原因：

因为不满足文件系统的一致性导致IO崩溃。

解决方案：

请按照提示执行 `e2fsck -f /dev/xxx` 后再尝试扩容。

相关文档

- 在线扩容云盘数据卷

11.3.9. 云盘存储卷FAQ

本文为您介绍云盘存储卷常见问题的处理方法。

- 云盘挂载失败，出现timeout错误
- 升级系统后，云盘报错input/output error
- 卸载云盘时提示The specified disk is not a portable disk
- 挂载云盘的Pod无法启动且提示had volume node affinity conflict
- 启动挂载了云盘的Pod时，提示can't find disk
- 动态创建PV失败，提示disk size is not supported

云盘挂载失败，出现 timeout 错误

问题现象：

云盘挂载失败，出现 timeout 错误。

问题原因：

如果节点为手动添加，可能是由于STS权限的问题导致。

解决方法：

需要手动配置RAM权限。具体操作，请参见授予实例RAM角色。

云盘挂载失败时出现Zone错误

问题现象：

云盘挂载时，出现Zone错误。

问题原因：

ECS挂载的云盘时不在同一个Region下面的相同Zone内。

解决方法：

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，云盘报错 input/output error

问题现象：

升级系统后，云盘报错 input/output error 。

问题原因：

Flexvolume组件版本过低。

解决方法：

1. 升级Flexvolume到v1.9.7-42e8198或以后版本。

   ```
   kubectl set image daemonset/flexvolume acs-flexvolume=registry.cn-hangzhou.aliyuncs.com/acs/flexvolume:v1.9.7-42e8198 -n kube-system
   ```

2. 对于已经出问题的Pod，需要重建。

关于Flexvolume组件版本信息，请参见Flexvolume。

卸载云盘时提示 The specified disk is not a portable disk

问题现象：

卸载云盘时，提示 The specified disk is not a portable disk 。

问题原因：

您申请了包年包月的云盘，或者在升级ECS时，将ECS关联的云盘一起升级为包年包月。

解决方法：

将云盘的付费方式改为按量付费。

挂载云盘的Pod无法启动且提示 had volume node affinity conflict

问题现象：

在启动挂载云盘的Pod的时候，出现Pod无法启动的情况并报错 had volume node affinity conflict 。

问题原因：

您在PV中编写了 nodeaffinity 属性，这个属性与Pod中声明的属性不一致，导致Pod无法被调度到正确的节点上。

解决方法：

修改PV或者Pod的属性，使二者属性保持一致。

启动挂载了云盘的Pod时，提示 can't find disk

问题现象：

卸载云盘时，提示 The specified disk is not a portable disk 。

问题原因：

- 您在编写PV的时候输入了错误的DiskID。
- 您的账号无权限操作DiskID，可能不是当前账号的DiskID。

解决方法：

更换DiskID。

动态创建PV失败，提示 disk size is not supported

问题现象：

动态创建PV失败，提示 disk size is not supported 。

问题原因：

在PVC中指定的云盘大小不符合规范，要求最小20 GiB。

解决方法：

调整PVC中云盘的大小，使其符合规范。

11.4. NAS存储卷

11.4.1. NAS存储卷概述

您可以在容器服务Kubernetes集群中使用阿里云NAS存储卷。本文介绍NAS存储卷的使用场景、注意事项等。

前提条件

使用NAS数据卷之前，您需要在文件存储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。

例如，您的挂载点为 055f84ad83-ixxxx.cn-hangzhou.nas.aliyuncs.com 格式。

背景信息

目前阿里云NAS支持两种Kubernetes挂载驱动：

- 静态存储卷挂载

- 动态存储卷挂载

使用场景

- 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StatefulSet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。

推荐的挂载方式

建议您按照以下推荐的方式来使用。

- 推荐的存储驱动
  建议您通过阿里云Flexvolume驱动来使用文件存储。
  阿里云托管的Kubernetes集群会在创建集群时默认安装Flexvolume驱动，您只需要确认驱动版本是最新即可，如果驱动不是最新版本，请参见升级组件。
- 推荐的挂载方式
  - 静态存储卷挂载，请参见使用NAS静态存储卷。
  - 动态存储卷挂载，请参见使用NAS动态存储卷。
- 不推荐的挂载方式
  容器服务仅支持通过阿里云Flexvolume存储驱动挂载，不支持Kubernetes原生NFS驱动。

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，请务必不要删除NAS挂载点，否则会造成操作系统无响应。
- 创建NAS挂载点后，请等待一定时间，待挂载点**状态**为**可用**后方可使用。
- 数据卷挂载协议推荐使用NFSv3。
- 使用NAS数据卷前，推荐升级Flexvolume到最新版本。
- 极速型NAS只支持NFSv3，挂载参数需要添加 `nolock` 。

11.4.2. 使用NAS静态存储卷

您可以通过阿里云提供的Flexvolume插件使用阿里云NAS文件存储服务。本文介绍如何使用NAS静态存储卷。

前提条件

- 使用存储卷时请将Flexvolume插件更新到最新版本。具体操作，请参见安装与升级Flexvolume组件。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

使用Flexvolume插件，您可以通过PV和PVC方式使用阿里云NAS存储卷。

注意事项

若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间过长。

> **说明** 若已配置securityContext.fsgroup参数，且需要减少挂载时间。具体操作，请参见NAS存储卷挂载时间延长。

操作步骤

Pod可以通过关联创建的PV和PVC的方式使用NAS存储卷。

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存储卷。

   - 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-u****.cn-hangzhou.nas.aliyuncs.com"
        path: "/k8s"
        vers: "3"
        options: "nolock,tcp,noresvport"
     ```

   - 通过控制台界面创建NAS存储卷。

     a. 登录容器服务管理控制台。

     b. 在控制台左侧导航栏中，单击**集群**。

     c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

     d. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。

     e. 在**存储卷**页面的右上角单击**创建**。

     f. 在创建存储卷对话框中，配置存储卷的相关参数。

参数	描述
存储卷类型	本示例中为NAS。
名称	创建的存储卷名称。存储卷名在集群内必须唯一。本例为**pv-nas**。
存储驱动	本示例选择Flexvolume。有关存储插件详细信息，请参见CSI和Flexvolume存储插件的区别。
总量	所创建存储卷的容量。注意不能超过NAS文件系统的存储容量。
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。关于NAS文件系统挂载点的管理，请参见管理挂载点。
子目录	NAS路径下的子目录，以 / 开头，设定后存储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。

参数	描述
权限	设置挂载目录的访问权限，例如755、644、777等。 **说明** ■ 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。 ■ 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。 如果挂载到NAS子目录时，您可以选择设置权限，或者不填此项。 ■ 不填此项，默认权限为NAS文件原来的权限。 ■ 选择设置权限时： ■ 如果是Flexvolume v1.14.6.15-8d3b7e7-aliyun以前版本，则使用递归方式进行权限操作，挂载目录下面所有文件、目录都会被修改权限。 ■ 如果是Flexvolume v1.14.6.15-8d3b7e7-aliyun以及以后版本，配置了此项，按照**权限模式**的配置执行权限操作。
权限模式	定义权限变更方式，支持非递归或递归方式。 ■ 非递归：执行权限变更时，只对挂载目录起作用，其子目录、包含的文件不进行变更权限。 ■ 递归：执行权限变更时，会对其子目录、包含的文件进行递归操作，全部变更权限。 **说明** 当挂载目录下面文件数量较多时，使用递归方式，会出现执行chmod耗时长，从而导致挂载、卸载操作失败的可能，请谨慎使用。
版本	挂载的NAS卷使用的NFS协议版本号，推荐使用V3，且极速类型NAS只支持V3。
标签	为该存储卷添加标签。

g. 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: /data
   volumes:
    - name: pvc-nas
      persistentVolumeClaim:
       claimName: pvc-nas
```

11.4.3. 使用NAS动态存储卷

在NAS文件系统中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。本文介绍如何使用NAS动态存储卷。

前提条件

- 已创建Kubernetes集群，并且在该集群中安装Flexvolume插件。具体操作，请参见创建Kubernetes托管版集群。
- 已部署alicloud-nas-controller组件。具体操作，请参见安装与升级Flexvolume组件。

注意事项

若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间过长。

> **说明** 若已配置securityContext.fsgroup参数，且需要减少挂载时间。具体操作，请参见NAS存储卷挂载时间延长。

创建动态NAS卷

1. 配置StorageClass。

   配置示例如下所示：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-nas
mountOptions:
- nolock,tcp,noresvport
- vers=3
parameters:
 server: "23a9649583-i****.cn-shenzhen.nas.aliyuncs.com:/nasroot1/"
 driver: flexvolume
provisioner: alicloud/nas
reclaimPolicy: Delete
```

参数	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. 在StatefulSet中使用NAS动态存储卷。

   使用以下示例创建Service及StatefulSet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/v1
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
  matchLabels:
   app: nginx
 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
```

11.4.4. 使用NAS实现共享存储和持久化存储-Flexvolume

NAS提供了共享存储和持久化存储的服务，满足您需要多个Pod共享数据的应用场景。本文为您介绍如何使用NAS实现共享存储和持久化存储。

前提条件

- 创建Kubernetes托管版集群。
- 通过kubectl工具连接集群。
- 您已在文件存储控制台创建一个文件系统，请参见Linux系统挂载NFS文件系统。创建的文件系统需要与您的

Kubernetes集群在同一可用区。

- 您已在创建好的文件系统中添加容器服务Kubernetes集群的挂载点，请参见管理挂载点管理挂载点。文件系统挂载时的VPC网络要与您Kubernetes集群所在的VPC网络保持一致。

背景信息

NAS支持同时被多个Pod挂载，此时多个Pod可能同时修改相同数据，需要应用自行实现数据的同步。

有状态服务-NAS的使用场景：

- 对磁盘I/O要求较高的应用。
- 读写性能相对于对象存储OSS高。
- 可实现跨主机文件共享，例如可作为文件服务器。

有状态服务-NAS的使用方式：

1. 手动创建文件系统，并添加挂载点。
2. 手动创建PV及PVC。

本文为您介绍利用阿里云提供的*flexvolume*插件，通过PV或PVC的方式使用阿里云NAS。

创建PV

1. 创建*pv-nas.yaml*文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-nas
 labels:
  alicloud-pvname: pv-nas
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 flexVolume:
  driver: "alicloud/nas"
  options:
   server: "***-**.cn-hangzhou.nas.aliyuncs.com"  ////请替换为您实际的挂载点
   path: "/k8s1"
   vers: "4.0"
```

参数	描述
alicloud-pvname	PV的名称。
server	NAS的挂载点。可在文件存储控制台，单击左侧导航栏的**文件系统列表**，选择目标文件系统，单击**操作**列**管理**，在**挂载点**区域，**挂载地址**即为NAS数据盘的挂载点。
path	NAS的挂载目录，支持挂载NAS的子目录，当子目录不存在时，会在NAS上自动创建子目录并挂载。
vers	（可选）NFS挂载协议的版本号，支持3和4.0，默认情况是3。

参数	描述
mode	（可选）挂载目录的访问权限，默认情况是不配置。 **说明** ○ 挂载NAS的根目录不能配置访问权限。 ○ 当NAS中数据量很大时，配置 mode 会导致挂载非常慢，甚至挂载失败，不建议配置。

2. 执行以下命令，创建PV。

```
kubectl create -f pv-nas.yaml
```

预期结果：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**，可以看到刚刚创建的PV。



创建PVC

创建NAS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-nas.yaml*文件。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-nas
spec:
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 5Gi
 selector:
  matchLabels:
   alicloud-pvname: pv-nas
```

2. 执行以下命令，创建PVC。

```
kubectl create -f pvc-nas.yaml
```

预期结果：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**，可以看到刚刚创建的PVC。



名称	总量	访问模式	状态	存储类型	关联的存储卷	创建时间	操作
disk-ssd	20Gi	ReadWriteOnce	Bound	alicloud-disk-ssd-hangzhou-g	[redacted]	2018-12-19 14:48:40	查看Yaml | 删除
disk-ssd-web-0	20Gi	ReadWriteOnce	Bound	alicloud-disk-ssd	[redacted]	2018-12-20 10:23:57	查看Yaml | 删除
disk-ssd-web-1	20Gi	ReadWriteOnce	Bound	alicloud-disk-ssd	[redacted]	2018-12-20 10:24:13	查看Yaml | 删除
pvc-nas	5Gi	ReadWriteMany	Bound		pv-nas	2018-12-20 19:23:02	查看Yaml | 删除

创建应用

1. 创建*nas.yaml*文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nas-static
 labels:
  app: nginx
spec:
 replicas: 2
 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: pvc-nas
```

2. 执行以下命令，创建Deployment。

```
kubectl create -f nas.yaml
```

预期结果：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
4. 在目标**集群**页面下，选择左侧导航栏的**工作负载 > 无状态**，可以看到刚刚创建的Deployment。



名称	标签	容器组数量	镜像	创建时间	操作
nas-static	app:nginx	2/2	nginx	2018-12-20 19:31:40	详情 | 编辑 | 伸缩 | 监控 | 更多
nginx-dynamic	app:nginx	1/1	nginx	2018-12-19 17:40:53	详情 | 编辑 | 伸缩 | 监控 | 更多

NAS的共享存储

1. 执行以下命令，查看部署的deployment所在Pod的名称。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME                READY  STATUS  RESTARTS  AGE
   nas-static-f96b6b5d7-r****   1/1   Running  0     9m
   nas-static-f96b6b5d7-w****   1/1   Running  0     9m
   ```

2. 执行以下命令，查看*/data*路径下的文件。

   ```
   kubectl exec nas-static-f96b6b5d7-r**** ls /data
   ```

   预期输出：

   ```
   kubectl exec nas-static-f96b6b5d7-w**** ls /data
   ```

   > **说明** */data*路径下为空，没有文件。

3. 执行以下命令，在任意一个Pod的*/data*路径下创建文件*nas*。

   ```
   kubectl exec nas-static-f96b6b5d7-r**** touch /data/nas
   ```

4. 查看Pod下的文件。

   执行以下命令，查看Pod*/data*路径下的文件。

   ```
   kubectl exec nas-static-f96b6b5d7-r**** ls /data
   ```

   预期输出：

   ```
   nas
   ```

   执行以下命令，查看另一个Pod*/data*路径下的文件。

   ```
   kubectl exec nas-static-f96b6b5d7-w**** ls /data
   ```

   预期输出：

   ```
   nas
   ```

   > **说明** 在任意一个Pod的*/data*下创建的文件，两个Pod下的*/data*路径下均存在此文件，说明两个Pod共享一个NAS。

NAS的持久化存储

1. 执行以下命令，删除该应用的所有Pod。

   ```
   kubectl delete pod nas-static-f96b6b5d7-r**** nas-static-f96b6b5d7-wthmb
   ```

   预期输出：

   ```
   pod "nas-static-f96b6b5d7-r****" deleted
   pod "nas-static-f96b6b5d7-w****" deleted
   ```

2. 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
```

预期输出：

```
NAME                READY  STATUS       RESTARTS   AGE
nas-static-f96b6b5d7-r****   1/1   Running       0       27m
nas-static-f96b6b5d7-w****    1/1   Running       0       27m
nas-static-f96b6b5d7-r****   1/1   Terminating     0       28m
nas-static-f96b6b5d7-w****    0/1   Pending       0       0s
nas-static-f96b6b5d7-w****    0/1   Pending       0       0s
nas-static-f96b6b5d7-w****    0/1   ContainerCreating  0       0s
nas-static-f96b6b5d7-w****    1/1   Terminating     0       28m
nas-static-f96b6b5d7-n****    0/1   Pending       0       0s
nas-static-f96b6b5d7-n****    0/1   Pending       0       0s
nas-static-f96b6b5d7-n****    0/1   ContainerCreating  0       0s
nas-static-f96b6b5d7-r****   0/1   Terminating     0       28m
nas-static-f96b6b5d7-w****    0/1   Terminating     0       28m
nas-static-f96b6b5d7-r****   0/1   Terminating     0       28m
nas-static-f96b6b5d7-r****   0/1   Terminating     0       28m
nas-static-f96b6b5d7-w****    1/1   Running       0       10s
nas-static-f96b6b5d7-w****    0/1   Terminating     0       28m
nas-static-f96b6b5d7-w****    0/1   Terminating     0       28m
nas-static-f96b6b5d7-n****    1/1   Running       0       17s
```

3. 执行以下命令，查看Kubernetes重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS  RESTARTS  AGE
nas-static-f96b6b5d7-n****    1/1   Running  0     21s
nas-static-f96b6b5d7-w****    1/1   Running  0     21s
```

4. 查看Pod下的文件。

   执行以下命令，查看Pod*/data*路径下的文件。

```
kubectl exec nas-static-f96b6b5d7-n**** ls /data
```

预期输出：

```
nas
```

执行以下命令，查看另一个Pod*/data*路径下的文件。

```
kubectl exec nas-static-f96b6b5d7-w**** ls /data
```

预期输出：

```
nas
```

> **说明** 刚刚创建的文件*nas*并没有被删除，说明NAS的数据可持久保存。

11.4.5. NAS存储卷FAQ

本文为您介绍NAS存储卷常见问题的处理方法。

- NAS存储卷挂载时间延长

- NAS存储卷挂载失败时出现timeout错误
- 使用NAS存储卷时，提示chown: option not permitted
- 挂载NAS存储卷失败
- 使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV

NAS存储卷挂载时间延长

问题现象：

NAS存储卷挂载时间延长。

问题原因：

若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间延长。

解决方法：

- 若应用模板中配置了securityContext.fsgroup参数，请删除securityContext下的fsgroup参数。
- 若需要将挂载目录内文件变成期望的UID和mode，可以手动将目标目录挂载到一台ECS。关于ECS实例挂载文件系统，请参见通过控制台实现ECS实例一键挂载文件系统。再通过命令行执行 `chown` 和 `chmod` ，完成后通过Flexvolume使用NAS存储卷。关于如何通过Flexvolume使用NAS存储卷，请参见使用NAS静态存储卷和使用NAS动态存储卷。
- 对于1.20及之后版本的Kubernetes集群，除了上述两种解决方法外，也可通过将fsGroupChangePolicy配置为OnRootMismatch，这时只有在首次启动时才会执行 `chmod` 或 `chown` 操作，导致存在挂载时间延长的问题，后续挂载NAS存储卷时挂载时间将恢复正常。关于fsGroupChangePolicy参数的更多信息，请参见为Pod或容器配置安全性上下文。

NAS存储卷挂载失败时出现 `timeout` 错误

问题现象：

NAS存储卷挂载失败时出现 `timeout` 错误。

问题原因：

NAS挂载点和集群不在同一VPC内。

解决方法：

选择与集群在同一VPC内的NAS挂载点。

使用NAS存储卷时，提示 `chown: option not permitted`

问题现象：

使用NAS存储卷时，提示 `chown: option not permitted` 。

问题原因：

您的容器没有权限使用该NAS存储卷。

解决方法：

您需要使用root权限启动容器。

挂载NAS存储卷失败

问题现象：

挂载NAS PV失败，并报以下错误：

```
Unable to mount volumes for pod "dp-earnings-pod_default(906172c6-3d68-11e8-86e0-00163e00****)": timeout expired waiting for volumes to attach/mount for pod "default"/"dp-earnings-pod". list of unattached/unmounted volumes=[vol1 vol2]
```

问题原因：

您没有安装Flexvolume插件。

解决方法：

您需要安装Flexvolume插件。具体操作，请参见安装与升级Flexvolume组件。

使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV

问题现象：

使用NAS动态存储卷时，若创建子目录写入速度快于Controller删除子目录速度，将可能导致Controller的任务队列阻塞且无法创建新的PV。

问题原因：

当集群使用动态NAS存储卷时，配置的StorageClass回收策略reclaimPolicy为Delete且archiveOnDelete为false。

解决方法：

将archiveOnDelete配置为true，当删除PV时只是修改NAS文件系统中子目录的名称，而不是真正删除文件。

具体的文件删除操作需要您自行处理，例如：在某个节点过载根目录启动定时删除机制，或启动多个Pod并发删除某种格式的子目录。

11.5. OSS存储卷

11.5.1. OSS存储卷概述

您可以在ACK集群中使用阿里云OSS数据卷。
目前，ACK仅支持OSS静态存储卷，不支持OSS动态存储卷。您可以通过使用创建PV和PVC的方式使用OSS静态存储卷。

前提条件

使用OSS静态存储卷之前，您需要先在OSS管理控制台上创建存储空间（Bucket）。具体操作，请参见创建存储空间。

使用说明

配置OSS静态存储卷的说明如下：

- OSS为共享存储，可以同时为多个Pod提供共享存储服务。
- bucket：目前只支持挂载存储空间Bucket，不支持挂载Bucket下面的子目录或文件。
- url：OSS的Endpoint，挂载OSS的接入域名，挂载节点和存储空间的地域相同时，可使用内网地址。
- akId：您的AccessKey ID值。
- akSecret：您的AccessKey Secret值。
- otherOpts：挂载OSS时支持定制化参数输入，格式为：`-o *** -o ***`。
- 您在使用OSS数据卷时，不要使用subpath的配置。
- 使用数据卷时，推荐每个应用使用独立的PV名字。
- 目前仅支持CentOS及Alibaba Cloud Linux 2操作系统。

注意事项

- OSS数据卷是使用OSSFS文件进行挂载的FUSE文件系统，适合于读文件场景。例如：读配置文件、视频、图片文件等场景。
- OSSFS不擅长于写文件的应用场景。如果您的业务是将文件写入存储的场景，推荐使用NAS存储卷服务。
- 内核态文件系统相较于用户态模拟的文件系统，在稳定性、性能上更具有优势，生产环境中推荐您使用NAS文件存储，而不推荐OSS对象存储。

- OSSFS可以通过调整配置参数的方式，优化其在缓存、权限等方面的表现，请参见常见问题、ossfs/README-CN.md、FAQ。

11.5.2. 使用OSS静态存储卷

本文为您介绍通过使用创建PV和PVC的方式使用OSS静态存储卷，及如何在PV中使用Secret。

注意事项

若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间延长。

> **说明** 若已配置securityContext.fsgroup参数，且需要减少挂载时间。具体操作，请参见OSS存储卷挂载时间延长。

使用创建PV和PVC的方式使用OSS静态存储卷

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: "LTAI4G6E3whAAKnzdRPx****"
        akSecret: "uRJeli0cbahgWOhxncpp54wR5b****"
        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. 在控制台左侧导航栏中，单击**集群**。

     c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

     d. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**。

     e. 在**存储卷**页面单击**创建**。

f. 在**创建存储卷**对话框中，配置存储卷的相关参数。

参数	描述
存储卷类型	本示例中为OSS。
名称	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` 。
存储驱动	本示例选择Flexvolume。有关存储插件详细信息，请参见CSI和Flexvolume存储插件的区别。
总量	所创建存储卷的容量。
访问模式	默认为ReadWriteMany。
访问证书	选择访问OSS所需的保密字典。 ■ **选择已有保密字典**：选择命名空间及保密字典。 ■ **新建保密字典**：配置命名空间、名称、AccessKey ID及AccessKey Secret。
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**公网域名**；若是经典网络，请选择**私网域名**。
标签	为该存储卷添加标签。

g. 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方式在PV中使用AccessKey

1. 执行以下命令，创建Secret。

```
kubectl create secret generic osssecret --from-literal=akId='LTAI4G6E3whAAKnzdRPx****' --from-literal=akSecret='uRJeIi0cbahgWOhxncpp54wR5b****' --type=alicloud/oss -n default
```

   - `osssecret`：Secret的名字，可以自行配置。
   - `akId`：AccessKey ID。
   - `akSecret`：AccessKey Secret。
   - Secret类型：配置为`alicloud/oss`名词空间，需要与期望的Pod在相同的命名空间。

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"
```

11.5.3. 使用OSS实现持久化存储-Flexvolume

当容器发生宕机故障时，有状态服务容器存储的业务数据存在着丢失和不可靠等风险。使用持久化存储可以解决该问题。本文介绍如何使用OSS实现持久化存储。

背景信息

阿里云对象存储服务（OSS）提供海量、安全、低成本、高可靠的云存储服务。OSS支持同时被多个Pod挂载。

OSS使用场景：

- 对磁盘I/O要求低。
- 配置文件、图片、小视频等共享业务。

OSS使用方式：

1. 手动创建Bucket。
2. 获取AccessKey ID和AccessKey Secret。
3. 手动创建PV及PVC。

前提条件

- 创建Kubernetes托管版集群。
- 通过kubectl工具连接集群。
- 您已在OSS管理控制台创建Bucket，请参见创建存储空间。

注意事项

- 容器服务Kubernetes集群升级会重启Kubelet，OSSFS驱动跟随一起重启，导致OSS目录不可用。这时使用OSS的Pod需要重建，可在YAML文件中增加健康检查的配置，在容器内OSS目录不可用时自动重启Pod，重新挂载OSS。
- 通过最新版本挂载的OSS已经解决了此问题。

创建PV

1. 创建*pv-oss.yaml*文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-oss
 labels:
  alicloud-pvname: pv-oss
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 storageClassName: oss
 flexVolume:
  driver: "alicloud/oss"
  options:
   bucket: "docker"                 //请替换为您的Bucket名称。
   url: "oss-cn-hangzhou.aliyuncs.com"   //请替换为您的URL。
   akId: "***"                  //请替换为您的AccessKey ID。
   akSecret: "***"                  //请替换为您的AccessKey Secret。
   otherOpts: "-o max_stat_cache_size=0 -o allow_other"  //请替换为您的otherOpts。
```

参数解释：

- `alicloud-pvname`：PV的名称，与PVC中的`selector`配合使用。
- `bucket`：Bucket名称，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 Bucket的访问域名（Endpoint），请参见访问域名和数据中心。也可以在OSS管理控制台查询：登录OSS管理控制台，在左侧导航栏选择目标Bucket，在**概览**页面的**访问域名**区域，查看**Endpoint（地域节点）**。
- `akId`：AccessKey ID。在容器服务管理控制台，单击右上角  ，选择**AccessKey 管理**，设置**AccessKey ID**和**AccessKey Secret**。
- `akSecret`：AccessKey Secret。获取方法同`akId`。
- `otherOpts`：挂载OSS时支持定制化参数，格式为：`-o *** -o ***`。

2. 执行以下命令，创建PV。

```
kubectl create -f pv-oss.yaml
```

预期结果：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**，可以看到刚刚创建的PV。



创建PVC

创建OSS存储声明PVC，使用 selector 筛选PV，精确配置PVC和PV的绑定关系。使用 storageClassName ，表示PVC只与OSS类型的PV绑定。

1. 创建*pvc-oss.yaml*文件。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-oss
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: oss
 resources:
  requests:
   storage: 5Gi
 selector:
  matchLabels:
   alicloud-pvname: pv-oss
```

2. 执行以下命令，创建PVC。

```
kubectl create -f pvc-oss.yaml
```

预期结果：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储卷**，可以看到刚刚创建的PVC。



名称	总量	访问模式	状态	存储类型	关联的存储卷	创建时间	操作
disk-ssd	20Gi	ReadWriteOnce	Bound	alicloud-disk-ssd-hangzhou-g	[redacted]	2018-12-19 14:48:40	查看Yaml | 删除
disk-ssd-web-0	20Gi	ReadWriteOnce	Bound	alicloud-disk-ssd	[redacted]	2018-12-20 10:23:57	查看Yaml | 删除
disk-ssd-web-1	20Gi	ReadWriteOnce	Bound	alicloud-disk-ssd	[redacted]	2018-12-20 10:24:13	查看Yaml | 删除
pvc-nas	5Gi	ReadWriteMany	Bound		pv-nas	2018-12-20 19:23:02	查看Yaml | 删除
pvc-oss	5Gi	ReadWriteMany	Bound	oss	pv-oss	2018-12-21 15:09:43	查看Yaml | 删除

创建应用

1. 创建*oss-static.yaml*文件。

```
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"
     - name: pvc-oss
       mountPath: "/data1"
    livenessProbe:
     exec:
      command:
      - sh
      - -c
      - cd /data
     initialDelaySeconds: 30
     periodSeconds: 30
   volumes:
    - name: pvc-oss
     persistentVolumeClaim:
      claimName: pvc-oss
```

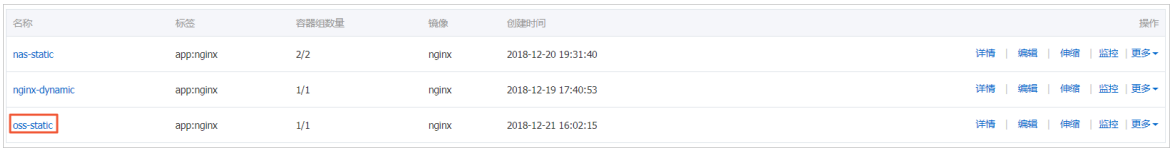
> **说明** 健康检查 livenessProbe 的详细内容及解释，请参见OSS存储卷概述。

2. 执行以下命令，创建Deployment。

```
kubectl create -f oss-static.yaml d
```

预期结果：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**，可以看到刚刚创建的Deployment。



OSS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
   ```

   预期输出：

   ```
   NAME                READY  STATUS  RESTARTS  AGE
   oss-static-66fbb85b67-dqbl2   1/1   Running  0     1h
   ```

2. 执行以下命令，查看*/data*路径下的文件。

   ```
   kubectl exec oss-static-66fbb85b67-dqbl2 ls /data | grep tmpfile
   ```

   > **说明** */data*路径下为空，没有文件。

3. 执行以下命令，在*/data*路径下创建文件*tmpfile*。

   ```
   kubectl exec oss-static-66fbb85b67-dqbl2 touch /data/tmpfile
   ```

4. 执行以下命令，查看*/data*路径下的文件。

   ```
   kubectl exec oss-static-66fbb85b67-dqbl2 ls /data | grep tmpfile
   ```

   预期输出：

   ```
   tmpfile
   ```

5. 执行以下命令，删除名称为*oss-static-66fbb85b67-dqbl2*的Pod。

   ```
   kubectl delete pod oss-static-66fbb85b67-dqbl2
   ```

   预期输出：

   ```
   pod "oss-static-66fbb85b67-dqbl2" deleted
   ```

6. 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
   ```

   预期输出：

   ```
   NAME                READY  STATUS  RESTARTS  AGE
   oss-static-66fbb85b67-dqbl2   1/1   Running  0     78m
   oss-static-66fbb85b67-dqbl2  1/1  Terminating  0   78m
   oss-static-66fbb85b67-zlvmw  0/1  Pending  0   <invalid>
   oss-static-66fbb85b67-zlvmw  0/1  Pending  0   <invalid>
   oss-static-66fbb85b67-zlvmw  0/1  ContainerCreating  0   <invalid>
   oss-static-66fbb85b67-dqbl2  0/1  Terminating  0   78m
   oss-static-66fbb85b67-dqbl2  0/1  Terminating  0   78m
   oss-static-66fbb85b67-dqbl2  0/1  Terminating  0   78m
   oss-static-66fbb85b67-zlvmw  1/1  Running  0   <invalid>
   ```

7. 执行以下命令，查看Kubernetes重建的Pod名称。

```
kubectl get pod
```

预期输出：

```
NAME                READY  STATUS  RESTARTS  AGE
oss-static-66fbb85b67-zlvmw   1/1   Running  0     40s
```

8. 执行以下命令，查看*/data*路径下的文件，刚刚创建的文件*tmpfile*并没有被删除，说明OSS上的数据可持久保存。

```
kubectl exec oss-static-66fbb85b67-zlvmw ls /data | grep tmpfile
```

预期输出：

```
tmpfile
```

11.5.4. OSS存储卷FAQ

本文为您介绍OSS存储卷常见问题的处理方法。

- OSS存储卷挂载失败
- 集群升级后容器内OSS挂载目录不可用
- OSS存储卷挂载时间延长

OSS存储卷挂载失败

问题现象：

OSS存储卷挂载失败。

问题原因：

- 使用的AccessKey不正确。
- OSS存储卷挂载使用的URL无法访问网络。

解决方法：

- 使用正确的AccessKey。
- 确认OSS存储卷挂载使用的URL可访问网络。

集群升级后容器内OSS挂载目录不可用

问题现象：

集群升级后，容器内OSS挂载目录不可用。

问题原因：

升级集群重启kubelet时，由于容器网络会重启，导致OSSFS进程重启，主机与容器目录映射失效。

解决方法：

需要重启容器，或重建Pod。您可以通过配置健康检查实现容器或Pod的自动重启。

关于使用OSS存储的更多信息，请参见OSS存储卷概述。

OSS存储卷挂载时间延长

问题现象：

OSS存储卷挂载时间延长。

问题原因：

若您在应用模板中配置了securityContext.fsgroup参数，kubelet在存储卷挂载完成后会执行 `chmod` 或 `chown` 操作，导致挂载时间延长。

解决方法：

- 若应用模板中配置了securityContext.fsgroup参数，请删除securityContext下的fsgroup参数。
- 若需要将挂载目录内文件变成期望的UID和mode，可以手动将Bucket挂载到一台ECS。关于ECS实例挂载OSS，请参见通过云存储网关挂载OSS。再通过命令行执行 `chown` 和 `chmod` ，完成后通过Flexvolume使用OSS存储卷。关于如何通过Flexvolume使用OSS存储卷，请参见使用OSS静态存储卷。
- 对于1.20及之后版本的Kubernetes集群，可通过将fsGroupChangePolicy配置为OnRootMismatch，这时只有在首次启动时才会执行 `chmod` 或 `chown` 操作。更多信息，请参见为Pod或容器配置安全性上下文。

11.6. CPFS存储卷

11.6.1. 如何在ACK集群中使用CPFS存储卷服务

您可以在阿里云容器服务Kubernetes版（ACK）中使用文件存储CPFS（Cloud Paralleled File System）。本文介绍如何在ACK中安装Flexvolume插件，并通过CPFS数据卷的方式为应用（Pod）提供CPFS服务。

前提条件

- 您已创建一个专有版Kubernetes集群，请参见创建Kubernetes托管版集群。
- 开放公网SSH访问，请参见SSH访问Kubernetes集群。

> 注意　ACK CPFS存储卷服务目前只支持CPFS 1.0版，不支持2.0版本。

背景信息

CPFS是一种并行文件系统。CPFS的数据存储在集群中的多个数据节点，并可由多个客户端同时访问，从而能够为大型高性能计算机集群提供高IOPS、高吞吐、低时延的数据存储服务。CPFS是共享存储服务类型，适合于容器服务场景对资源共享、高性能的要求，在大数据、AI、基因计算等高性能场景中使用容器服务和CPFS是一个推荐的解决方案。有关CPFS详情，请参见什么是文件存储CPFS。

步骤一、部署插件

ACK中使用CPFS依赖以下两个驱动类组件：

- CPFS容器驱动：即为Flexvolume-cpfs插件，对Centos各个版本都兼容，部署Flexvolume-cpfs即可完成安装。
- CPFS客户端驱动：为CPFS挂载时的客户端（类似于nfs-client），驱动与操作系统内核是强依赖。容器场景中安装CPFS客户端驱动有以下方式：
  - 手动安装驱动，请参见挂载文件系统。
  - Flexvolume-cpfs部署时自动安装驱动，但只支持部分内核版本。
    您可以通过在节点上执行命令 `uname -a` 查看内核版本。目前容器场景下您可以在以下内核版本中安装CPFS客户端驱动。

```
3.10.0-957.5.1
3.10.0-957.21.3
3.10.0-1062.9.1
```

> 说明
> - 目前Flexvolume只支持安装CPFS Client驱动，不支持cpfs-client驱动升级，即发现节点安装了CPFS客户端，不再继续安装驱动。
> - 升级Flexvolume版本，只会升级Flexvolume驱动（容器驱动），而不会升级cpfs-client版本。
> - 在已经部署了cpfs-client、lustre驱动的节点上安装cpfs flexvolume不会再安装新版本的cpfs-client。
> - cpfs-client驱动升级需要手动进行，具体操作请参见挂载文件系统。

1. 在集群节点中部署YAML模板。
   i. 在客户端计算机中通过kubectl工具连接集群。
   ii. 创建名为*flexvolume-cpfs.yaml*的文件。
   iii. 将以下模板拷贝至文件中。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: flexvolume-cpfs
 namespace: kube-system
 labels:
  k8s-volume: flexvolume-cpfs
spec:
 selector:
  matchLabels:
   name: flexvolume-cpfs
 template:
  metadata:
   labels:
    name: flexvolume-cpfs
  spec:
   hostPID: true
   hostNetwork: true
   tolerations:
   - operator: "Exists"
   priorityClassName: system-node-critical
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
       - key: type
         operator: NotIn
         values:
         - virtual-kubelet
   containers:
   - name: flexvolume-cpfs
    image: registry.cn-hangzhou.aliyuncs.com/acs/flexvolume:v1.14.8.96-0d85fd1-aliyun
    imagePullPolicy: Always
    securityContext:
     privileged: true
    env:
    - name: ACS_CPFS
     value: "true"
    - name: FIX_ISSUES
     value: "false"
    livenessProbe:
     exec:
      command:
      - sh
      - -c
      - ls /acs/flexvolume
     failureThreshold: 8
     initialDelaySeconds: 15
     periodSeconds: 60
     successThreshold: 1
     timeoutSeconds: 15
```

```
      timeoutSeconds: 15
      volumeMounts:
      - name: usrdir
        mountPath: /host/usr/
      - name: etcdir
        mountPath: /host/etc/
      - name: logdir
        mountPath: /var/log/alicloud/
      - mountPath: /var/lib/kubelet
        mountPropagation: Bidirectional
        name: kubeletdir
    volumes:
    - name: usrdir
      hostPath:
        path: /usr/
    - name: etcdir
      hostPath:
        path: /etc/
    - name: logdir
      hostPath:
        path: /var/log/alicloud/
    - hostPath:
        path: /var/lib/kubelet
        type: Directory
      name: kubeletdir
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 10%
    type: RollingUpdate
```

iv. 执行以下命令部署上述YAML文件至集群节点中。

```
kubectl create -f flexvolume-cpfs.yaml
```

2. 检查部署情况。

i. 执行以下命令查看存储插件的部署情况。

```
kubectl get pod -nkube-system | grep flex
```

返回结果如下：

```
flexvolume-97psk                1/1   Running  0      27m
flexvolume-cpfs-dgxfq             1/1   Running  0      98s
flexvolume-cpfs-qpbcb             1/1   Running  0      98s
flexvolume-cpfs-vlrf9             1/1   Running  0      98s
flexvolume-cpfs-wklls             1/1   Running  0      98s
flexvolume-cpfs-xtl9b             1/1   Running  0      98s
flexvolume-j8zjr                1/1   Running  0      27m
flexvolume-pcg4l                1/1   Running  0      27m
flexvolume-tjxxn                1/1   Running  0      27m
flexvolume-x7ljw                1/1   Running  0      27m
```

> **说明** 以flexvolume-cpfs开头的Pod表示部署的CPFS存储卷插件。不含CPFS字样的flexvolume Pod表示集群默认部署的NAS、云盘、OSS存储卷插件。两个插件可以同时部署。

ii. 执行以下命令查看cpfs-client驱动是否安装完成。

```
rpm -qa | grep cpfs
```

返回结果如下：

```
kmod-cpfs-client-2.10.8-202.el7.x86_64
cpfs-client-2.10.8-202.el7.x86_64
```

iii. 执行以下命令查看mount.lustre是否安装完成。

```
which mount.lustre
```

返回结果如下：

```
/usr/sbin/mount.lustre
```

步骤二、使用CPFS数据卷

在ACK中使用CPFS数据卷，需要您先到CPFS控制台创建一个CPFS卷和挂载点，参见创建文件系统。

> **注意** 创建CPFS挂载点时，选择的VPC网络需要和ACK集群在同一个VPC内。

在以下示例中，获取的挂载点为 *cpfs-*-alup.cn-shenzhen.cpfs.nas.aliyuncs.com@tcp:cpfs--ws5v.cn-shenzhen.cpfs.nas.aliyuncs.com@tcp*；文件系统ID为 *0237ef41*。

1. 创建PV。
   i. 创建名为*pv-cpfs.yaml*的文件。

ii. 拷贝以下模板至文件中。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-cpfs
 labels:
  alicloud-pvname: pv-cpfs
spec:
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 flexVolume:
  driver: "alicloud/cpfs"
  options:
   server: "cpfs-****-alup.cn-shenzhen.cpfs.nas.aliyuncs.com@tcp:cpfs-***-ws5v.cn-shenzhen.cpfs.nas.aliyuncs.com@tcp"
   fileSystem: "0237ef41"
   subPath: "/k8s"
   options: "ro"
```

参数	描述
server	配置为CPFS的挂载点。
fileSystem	配置为CPFS文件系统ID。
subPath	配置为期望挂载的CPFS子目录，相对于文件系统根目录。
options	可选，挂载配置选项。

iii. 执行以下命令创建PV。

```
kubectl create -f pv-cpfs.yaml
```

2. 创建PVC。

i. 创建*pvc-cpfs*文件，并拷贝以下模板至文件中。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: pvc-cpfs
spec:
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 5Gi
 selector:
  matchLabels:
   alicloud-pvname: pv-cpfs
```

ii. 执行命令 `kubectl create -f pvc-cpfs` 创建PVC。

3. 创建应用。

i. 创建*nas-cpfs*文件，并拷贝以下模板至文件中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nas-cpfs
 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-cpfs
       mountPath: "/data"
   volumes:
    - name: pvc-cpfs
      persistentVolumeClaim:
       claimName: pvc-cpfs
```

ii. 执行命令 `kubectl create -f nas-cpfs` 创建应用。

执行结果

执行以下命令检查Pod挂载情况。

```
kubectl get pod
```

返回结果如下：

```
NAME            READY  STATUS   RESTARTS  AGE
nas-cpfs-79964997f5-kzrtp  1/1   Running  0     45s
```

执行以下命令进入Pod查看挂载目录。

```
kubectl exec -ti nas-cpfs-79964997f5-kzrtp sh
mount | grep k8s
```

返回结果如下：

```
192.168.1.12@tcp:192.168.1.10@tcp:/0237ef41/k8s on /data type lustre (ro,lazystatfs)
```

执行以下命令进入Pod所在节点，查看挂载目录。

```
mount | grep cpfs
```

返回结果如下：

```
192.168.1.12@tcp:192.168.1.10@tcp:/0237ef41/k8s on /var/lib/kubelet/pods/c4684de2-26ce-11ea-abbd-00163e12e203/volumes/alicloud~cpfs/pv-cpfs type lustre (ro,lazystatfs)
```

相关文档

- CSI中如何使用CPFS服务

11.7. 创建持久化存储卷声明

本文介绍通过容器服务控制台创建持久化存储卷声明（PVC）。

前提条件

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建存储卷，本例中使用云盘创建一个云盘存储卷。具体操作，请参见云盘存储卷概述。
  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存储卷时，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。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
5. 在**存储声明**页面，单击**创建**。
6. 在**创建存储声明**对话框中配置参数，最后单击**创建**。
   - **存储声明类型**： 和存储卷一致， 包括云盘、NAS或OSS。
   - **名称**： 输入存储卷声明名称。
   - **分配模式**： 支持**使用存储类动态创建**、**已有存储卷**和**创建存储卷**。本例选择**使用存储类动态创建**或**已有存储卷**。
   - **已有存储类**：单击**选择存储类**，在目标存储类右侧操作列单击**选择**，选择存储类。

     > **说明** 仅分配模式选择**使用存储类动态创建**，才需设置该参数。

   - **已有存储卷**： 单击**选择已有存储卷**，在目标存储卷右侧操作列单击**选择**，选择存储卷。

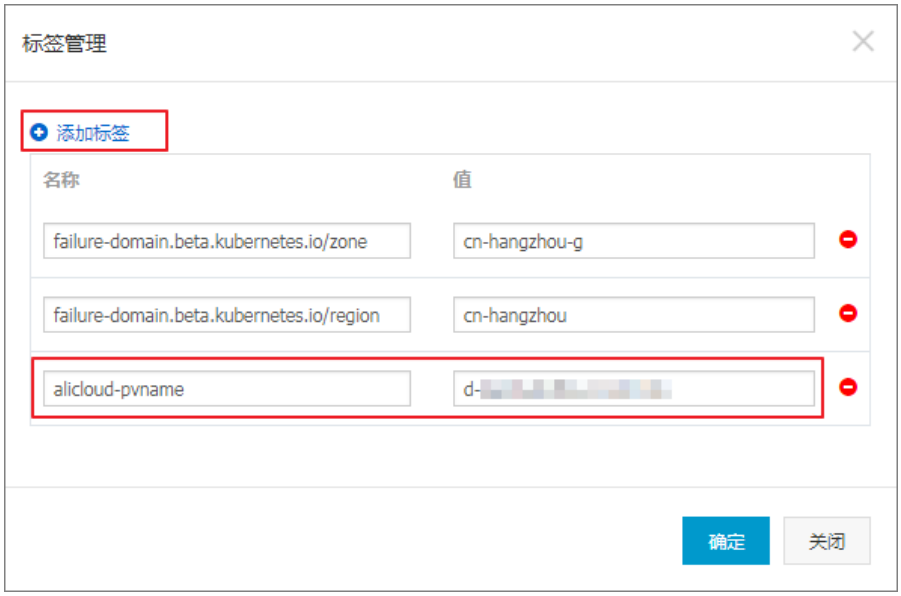
     > **说明** 仅分配模式选择**已有存储卷**，才需设置该参数。

   - **总量**： 声明使用量，不能大于存储卷的总量。
   - **访问模式**：默认为ReadWriteOnce。

     > **说明** 仅分配模式选择**使用存储类动态创建**，才需设置该参数。

   > **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选择**存储 > 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。



7. 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。

11.8. 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，请参见云盘存储卷概述。

前提条件

- 您已创建一个Kubernetes集群，请参见创建Kubernetes托管版集群。
- 您已创建一个存储卷声明，本例中使用云盘创建一个云盘存储卷声明pvc-disk，请参见创建持久化存储卷声明。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5.
6. 在**应用基本信息**页面，设置应用的基本信息。

   详细信息，请参见应用基本信息。



7. 在**容器配置**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**。详情请参见容器配置。
8. 在**高级配置**页面，配置test-nginx应用的服务，然后单击**创建**。
9. 在**创建完成**页面，单击**查看应用详情**查看应用任务。

   默认进入新建的test-nginx的详情页面。
10. 在**容器组**页签，找到该应用所属容器组，单击**详情**。
11. 在容器组详情页面，单击**存储**页签，您可看到该容器组正确绑定了pvc-disk。

11.9. 存储FAQ-Flexvolume

本文介绍您在使用云盘、NAS及OSS存储卷时常见问题的处理方法。

类型	问题
存储常见问题	• 如何解决存储卷挂载不上的问题？ • 如何查看存储相关日志？ • 如何解决Kubelet出现不受ACK管理的Pod日志的问题？
云盘存储卷常见问题	• 云盘挂载失败，出现timeout错误 • 云盘挂载失败时出现Zone错误 • 升级系统后，云盘报错input/output error • 卸载云盘时提示The specified disk is not a portable disk • 挂载云盘的Pod无法启动且提示had volume node affinity conflict • 启动挂载了云盘的Pod时，提示can't find disk • 动态创建PV失败，提示disk size is not supported

类型	问题
NAS存储卷常见问题	• NAS存储卷挂载时间延长 • NAS存储卷挂载失败时出现timeout错误 • 使用NAS存储卷时，提示chown: option not permitted • 挂载NAS存储卷失败 • 使用NAS动态存储卷时Controller的任务队列已满且无法创建新的PV
OSS存储卷常见问题	• OSS存储卷挂载失败 • 集群升级后容器内OSS挂载目录不可用 • OSS存储卷挂载时间延长

如何解决存储卷挂载不上的问题？

您需要检查Flexvolume和动态存储插件是否安装，如果没有，请安装Flexvolume和动态存储插件。

方式一：检查Flexvolume是否安装

执行以下命令获取Pod信息。

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

预期输出：

```
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。

如果运行状态不对，请参见插件运行日志分析。

方式二：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，执行以下命令查看Pod信息。

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

预期输出：

```
alicloud-disk-controller-8679c9fc76-lq6zb   1/1 Running   0   7d
```

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？

您可以查看Flexvolume日志、Provisioner插件日志和Kubelet日志。

方式一：查看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
```

> 说明 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。

查看云盘、NAS及OSS驱动日志。

执行以下命令查看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
```

预期输出：

```
Node: cn-hangzhou.i-bp19myla3uvnt6zi****/192.168.XX.XX
Node-Selectors: <none>
```

登录节点，查看云盘、NAS、OSS挂载的日志。

```
ssh 192.168.XX.XX
ls /var/log/alicloud/flexvolume*
```

预期输出：

```
flexvolume_disk.log flexvolume_nas.log flexvolume_o#ss.log
```

方式二：查看Provisioner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
```

> 说明 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。

方式三：查看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
```

预期输出：

```
Node: cn-hangzhou.i-bp19myla3uvnt6zi****/192.168.XX.XX
Node-Selectors: <none>
```

登录节点，查看kubelet日志。

```
ssh 192.168.XX.XX
journalctl -u kubelet -r -n 1000 &> kubelet.log
```

> **说明** -n的值表示期望看到的日志行数。

上述为获取Flexvolume、Provisioner、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。

如何解决Kubelet出现不受ACK管理的Pod日志的问题？

Pod异常退出，导致数据卷挂载点在卸载过程中没有清理干净，最终导致Pod无法删除。Kubelet的GC流程对数据卷垃圾回收实现并不完善，目前需要手动或脚本自动化实现垃圾挂载点的清理工作。

您需要在问题节点运行以下脚本，对垃圾挂载点进行清理。

```
wget https://raw.githubusercontent.com/AliyunContainerService/kubernetes-issues-solution/master/kubelet/kubelet.sh
sh kubelet.sh
```

12.安全

12.1. 安全体系概述

本文从运行时安全、可信软件供应链和基础架构安全三个维度介绍阿里云容器服务Kubernetes版的安全体系，包括安全巡检、策略管理、运行时监控和告警、镜像扫描、镜像签名、云原生应用交付链、默认安全、身份管理、细粒度访问控制等。



运行时安全

- 安全巡检
  集群的应用开发人员应该遵循权限的最小化原则配置应用部署模板，攻击者往往是利用应用Pod中开启的不必要的特权能力发起逃逸攻击，因此阿里云容器服务Kubernetes版ACK（Alibaba Cloud Container Service for Kubernetes）提供了应用运行时刻的安全配置巡检能力，帮助您实时了解当前状态下运行应用的配置是否存在安全隐患。
  巡检结果支持以报表化的方式展示，同时展示巡检对应扫描项的说明和修复建议。您还可以配置定期巡检，对应的扫描结果会写入到SLS指定的日志库中存储。具体操作，请参见使用配置巡检检查集群Workload安全隐患。
- 策略管理
  PSP（PodSecurityPolicy）是Kubernetes中Pod部署时重要的安全校验手段，能够有效地约束应用运行时的行为安全。它能够在应用部署时对Pod进行强制的安全校验，在集群的安全纵深防护中发挥了像AppArmor，SELinux这类策略引擎的作用。
  PSP是Kubernetes原生的集群维度资源模型，通过API Server的Admission Controller准入机制，在应用部署时刻对Pod的安全参数进行强制校验。如果参数配置不满足指定PSP策略的定义，API Server会禁止该Pod的部署请求。
  ACK集群默认开启了PSP特性，并且在控制台提供了一键化的PodSecurityPolicy Admission Controller开启关闭能力。同时基于原生的PSP模型提供了对应的策略定制编写和面向指定ServiceAccount的绑定能力。在降低了PSP使用门槛的同时，也能够避免因为应用PSP策略绑定导致已有应用无法再次部署的使用问题。更多信息，请参见使用PSP安全策略。
- 运行时监控和告警
  当容器应用通过API Server的认证鉴权和准入控制校验成功部署后，在云原生应用零信任的安全原则下，还需要在容器应用的运行时刻提供相应的安全监控和告警能力。因此，阿里云容器服务和云安全中心深度集成了告警处理和漏洞检测能力，集群管理员可以在应用运行时提供监控和告警能力，主要的容器侧攻击防护行为如下：
  - 恶意镜像启动告警。
  - 病毒和恶意程序的查杀。
  - 容器内部入侵告警。
  - 容器逃逸和高风险操作预警。

  您可以在集群管理页面的**安全管理 > 运行时刻安全监控**页面实时接收到相应告警，并根据页面提示查看和处理告警详情。具体操作，请参见使用运行时刻安全监控。
- 安全沙箱管理

相比于原有Docker运行时，安全沙箱为您提供的一种新的容器运行时选项，可以让您的应用运行在一个轻量虚拟机沙箱环境中，拥有独立的内核，具备更好的安全隔离能力。
安全沙箱特别适合于不可信应用隔离、故障隔离、性能隔离、多用户间负载隔离等场景。在提升安全性的同时，对性能影响非常小，并且具备与Docker容器一样的用户体验，例如日志、监控、弹性等。关于安全沙箱管理的详细介绍，请参见安全沙箱概述。

- ACK-TEE机密计算
  当面向诸如金融，政府等有很强安全诉求的应用场景时，ACK-TEE机密计算提供了基于硬件加密技术的云原生一站式机密计算容器平台，它可以帮助您保护数据使用（计算）过程中的安全性、完整性和机密性，同时简化了可信或机密应用的开发、交付和管理成本。
  机密计算可以让您把重要的数据和代码放在一个特殊的可信执行加密环境（Trusted Execution Environment，TEE）中，而不会暴露给系统其他部分，其他应用、BIOS、OS、Kernel、管理员、运维人员、云厂商，甚至除了CPU以外的其他硬件均无法访问机密计算平台数据，极大减少敏感数据的泄露风险，为您提供了更好的控制、透明度和隐秘性。更多信息，请参见ACK-TEE机密计算介绍。

可信软件供应链

- 镜像扫描
  容器镜像服务支持所有基于Linux的容器镜像安全扫描，可以识别镜像中已知的漏洞信息。您可以收到相应的漏洞信息评估和相关的漏洞修复建议，为您大幅降低使用容器的安全风险。容器镜像服务也接入了云安全的扫描引擎，可支持镜像系统漏洞、镜像应用漏洞和镜像恶意样本的识别。关于镜像扫描的详细介绍，请参见安全扫描容器镜像。
- 镜像签名
  在容器镜像管理中，您可以通过内容可信的机制保障镜像来源的安全性及不被篡改。镜像的创建者可以对镜像做数字签名，数字签名将保存在容器镜像服务中。通过在部署前对容器镜像进行签名验证可以确保集群中只部署可信授权方签名的容器镜像，降低在您的环境中运行意外或恶意代码的风险，确保从软件供应链到容器部署流程中应用镜像的安全和可溯源性。关于镜像签名和验签的配置使用流程，请参见使用kritis-validation-hook组件实现自动验证容器镜像签名。
- 云原生应用交付链
  在容器安全高效交付场景中，您可以使用容器镜像服务的云原生应用交付链功能，配置镜像构建、镜像扫描、镜像全球同步和镜像部署等，自定义细粒度安全策略，实现全链路可观测、可追踪的安全交付。保障代码一次提交，全球多地域安全分发和高效部署，将DevOps的交付流程全面升级成DevSecOps。关于云原生应用交付链的详细介绍，请参见创建交付链。

基础架构安全

- 默认安全
  阿里云容器服务Kubernetes版ACK（Alibaba Cloud Container Service for Kubernetes）集群节点，ControlPlane组件配置基于CIS kubernetes benchmark加固，且集群内所有系统组件均依据容器安全最佳实践进行了组件配置上的加固，同时保证系统组件镜像没有严重级别的CVE漏洞。
  - 每个新建的集群会默认分配一个与之对应的安全组，该安全组对于公网入方向仅允许ICMP请求。创建的集群默认不允许公网SSH连入，如果您需要配置通过公网SSH连入到集群节点，具体操作，请参见通过SSH连接ACK专有版集群的Master节点。
  - 集群节点通过NAT网关访问公网，可进一步减少安全风险。
  - 在托管集群的Worker节点上，遵循权限最小化原则，节点上绑定的RAM角色对应的阿里云资源访问权限经过了最小化收敛。更多信息，请参见【产品变更】托管集群节点RAM角色收敛公告。
- 身份管理
  ACK集群内所有组件之间的通讯链路均需要TLS证书校验，保证全链路通讯的数据传输安全，同时ACK管控侧会负责集群系统组件的证书自动更新。RAM账号或角色扮演用户均可以通过控制台或OpenAPI的方式获取连接指定集群API Server的Kubeconfig访问凭证，具体操作，请参见获取集群kubeconfig接口。ACK负责维护访问凭证中签发的身份信息，对于可能泄露的已下发Kubeconfig，可以及时进行吊销操作，具体操作，请参见吊销集群的KubeConfig凭证。
  在集群创建时，ACK支持**服务账户令牌卷投影**（Service Account Token Volume Projection）特性以增强在应用中使用ServiceAccount的安全性。具体操作，请参见部署服务账户令牌卷投影。

- 细粒度访问控制
  基于Kubernetes RBAC实现了对ACK集群内Kubernetes资源的访问控制，它是保护应用安全的一个基本且必要的加固措施。ACK在控制台的**授权管理**页面中提供了命名空间维度的细粒度RBAC授权能力，主要包括以下几点。
  - 根据企业内部不同人员对权限需求的不同，系统预置了管理员、运维人员、开发人员等对应的RBAC权限模板，降低了RBAC授权的使用难度。
  - 支持多集群和多个子账号的批量授权。
  - 支持RAM角色扮演用户的授权。
  - 支持绑定用户在集群中自定义的ClusterRole。

  更多信息，请参见配置子账号RBAC权限。
  ACK同时支持以组件管理的方式安装Gatekeeper组件，提供基于OPA策略引擎的细粒度访问控制能力。具体操作，请参见gatekeeper。
- 审计
  ACK和SLS日志服务进行了深度集成，支持多种审计日志的采集、检索和图表化展示功能，具体包括以下三种：
  - 集群API Server审计日志：帮助您记录或追溯集群访问者的日常操作，是集群安全运维中的重要环节。在**集群审计**页面，您可以查看内容丰富的审计报表，同时可基于日志内容设置对指定资源类型操作的实时告警。具体操作，请参见使用集群审计功能。
  - Ingress流量审计：通过不同的可视化流量报表帮助您了解集群Ingress的整体状态，比如服务访问的PV和UV，成功和失败比例，以及延迟信息等全方位的流量监控，同时支持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，提供问题发现的效率。具体操作，请参见Ingress访问日志分析与监控。
  - 事件监控审计：基于事件的监控可以帮助您通过事件获取，实时诊断集群的异常和安全隐患。更多信息，请参见事件监控。
- Secret落盘加密
  Kubernetes原生的Secret模型在Etcd落盘时只经过了Base64编码，为了保护Secret中敏感数据的落盘安全性，在ACK Pro托管集群中，您可以使用在阿里云密钥管理服务KMS（Key Management Service）中创建的密钥加密Kubernetes集群Secret，实现应用敏感数据的落盘加密。具体操作，请参见使用阿里云KMS进行Secret的落盘加密。

12.2. 基础设施安全

12.2.1. 在Kubernetes中实现HTTPS安全访问

容器服务ACK集群支持多种应用访问的形式，最常见形式如 `<SLB-Instance-IP>:<Port>` 、 `<NodeIP>:<NodePort>` 和域名访问等。ACK集群默认不支持HTTPS访问，如果您希望能够通过HTTPS进行应用的访问，容器服务ACK和阿里云负载均衡服务为您提供安全的HTTPS访问。本文通过实际案例演示的HTTPS访问配置，帮助您在容器服务ACK中配置自己的证书。

前提条件

- 创建Kubernetes托管版集群。
- 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集群的服务器证书。

  ```
  openssl genrsa -out tls.key 2048
  ```

  输出：

  ```
  Generating RSA private key, 2048 bit long modulus
  ..........................................................+++
  ..................................................................................+++
  e is 65537 (0x10001)
  ```

  ```
  ls
  ```

  输出：

  ```
  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
  ```

- 您也可以选择购买阿里云签发证书。具体操作，请参见选择阿里云签发证书。

背景信息

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。

在SLB上配置HTTPS证书

该方式有如下特点：

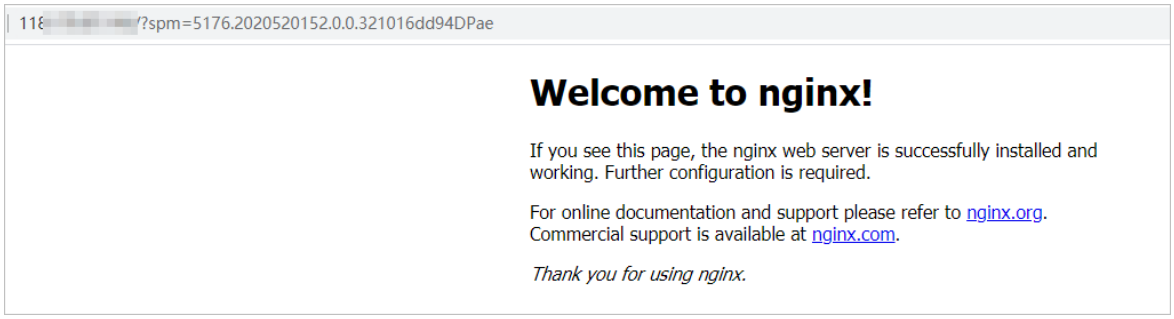
- 优点：证书配置在SLB上，为应用外部访问的入口，在集群内部进行应用的访问依然用的是HTTP访问方式。
- 缺点：需要维护较多的域名与IP地址的对应关系。
- 适用场景：应用不使用Ingress暴露访问方式，通过LoadBalancer类型的Service进行应用访问的暴露。

准备工作：

您已在该ACK集群中创建一个Nginx应用，该应用采用LoadBalancer类型的服务（Service）对外提供访问。更多信息，请参见创建无状态工作负载Deployment。

示例：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 选择集群的命名空间和服务，单击外部端点，您可通过 `<SLB IP>:<Port>` 的方式访问该应用。



6. 登录负载均衡管理控制台。
7. 配置**SSL证书**。
   - 如果您是通过命令方式创建集群的服务器证书，您需要使用前提条件中创建的公钥证书和私钥上传非阿里云签发证书。具体操作，请参见上传非阿里云签发证书。
   - 如果您是通过购买方式获取阿里云签发证书，请跳过此步骤。关于创建阿里云签发证书的操作，请参见选择阿里云签发证书。

     在证书列表中，找到目标证书名称下面的证书ID。
8. 在容器服务管理控制台的服务列表中，找到之前创建的服务，单击右侧**操作**列下的**更新**。
9. 在**更新服务**对话框中的注解区域，添加以下两个注解内容。



注解	名称	值
注解一	service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-port	https:443
注解二	service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id	${YOUR_CERT_ID}

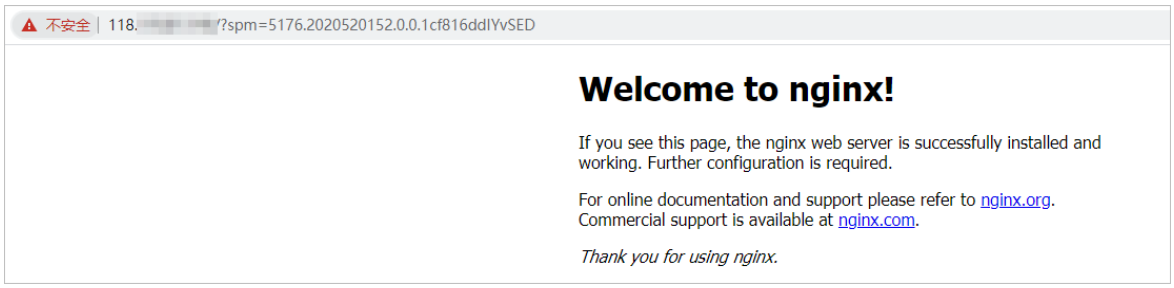
> **说明** 将${YOUR_CERT_ID}替换成步骤7配置SSL证书生成的证书ID。

您还可以使用YAML方式添加注解内容，完整YAML示例如下：

```
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:
 - name: https
  port: 443
  protocol: TCP
  targetPort: 80
 - name: http
  port: 80
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 type: LoadBalancer
```

> **说明** HTTPS的443端口对应的targetPort端口需要配置成HTTP的端口80。

10. 访问HTTPS的Nginx应用，在浏览器中输入 `https://<slb-instance-ip>` 并进行访问。



在Ingress上配置证书

该方法有以下特点：

- 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。更多信息，请参见创建无状态工作负载Deployment。

示例：

1. 根据前提条件中准备好的证书执行以下命令创建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. 在控制台左侧导航栏中，单击**集群**。

4. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
5. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
6. 在**路由**页面，单击页面右上角的**创建**。
7. 在**创建**路由对话框中，配置可HTTPS访问的路由，完成后单击**创建**。

   更多详细的路由配置信息，请参见创建路由（Ingress）。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。

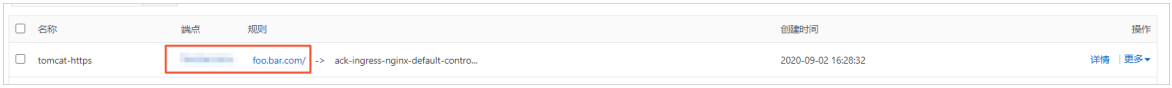
   您也可采用YAML文件的方式创建路由（Ingress），本例对应的YAML示例文件如下。

```
apiVersion: networking.k8s.io/v1
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
```

8. 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。

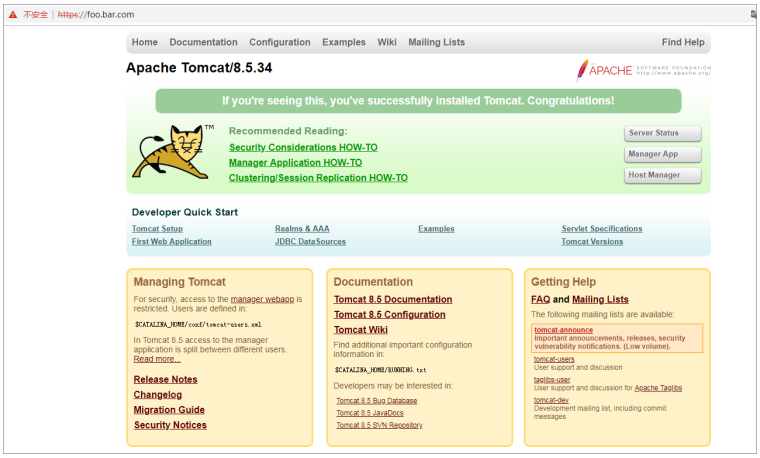
   说明 本例中以 foo.bar.com 作为测试域名，您需要在hosts文件中创建一条记录。

```
47.110.119.203 foo.bar.com          #其中IP地址即是路由的端点。
```



9. 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。



12.2.2. 使用集群审计功能

在Kubernetes集群中，API Server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API Server审计日志的相关配置，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，以及如何关闭集群审计功能。

使用说明

- 本文关于集群审计功能的配置适用于ACK托管版集群及ACK专有版集群。关于如何配置注册集群的集群审计功能，请参见在注册集群中使用集群审计功能。
- 注册集群不支持关闭集群审计功能。

配置介绍

当前创建Kubernetes集群时会默认选中**使用日志服务**，开启API Server审计功能，相关的参数配置功能如下。

> **说明** 登录到Master节点，API Server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	单个审计日志最大内存为100 MB。
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.
```

```
# 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*对于节点的get请求，kube组件在kube-system下对于endpoint的操作，以及API Server对于Namespaces的get请求等不作审计。
- 对于*/healthz**，*/version**及*/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authentication、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。
- RAM用户操作详细列表。
- 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**
> - 创建集群时候，默认选中使用日志服务，即会自动开通审计报表相关功能。日志服务的计费详情请参见计费说明。若未开通，请参见开通集群审计功能。
> - 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。

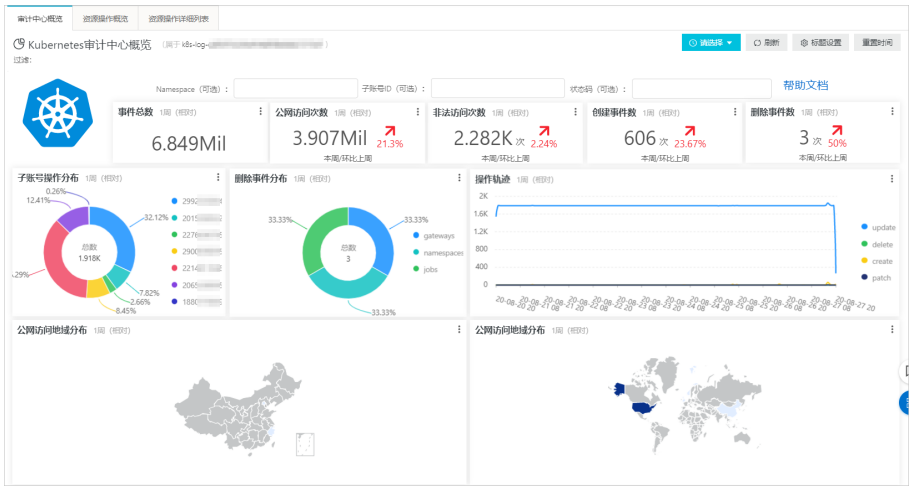
您可以通过以下两种方式访问审计报表：

- 登录容器服务管理控制台，在左侧导航栏单击**集群**，在集群列表目标集群的**操作**列中，单击**更多 > 集群审计**。
- 登录容器服务管理控制台，在左侧导航栏单击**集群**，在集群列表中单击集群名称，进入到集群信息页面。在集群管理页左侧导航栏中，单击**安全管理 > 集群审计**。

审计报表说明

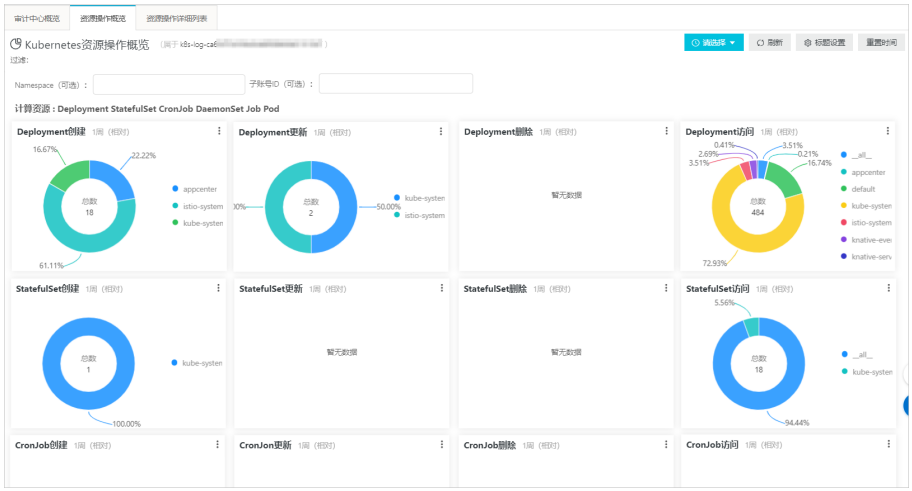
API Server审计共3个报表。分别是：审计中心概览、资源操作概览以及资源操作详细列表。

- 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。



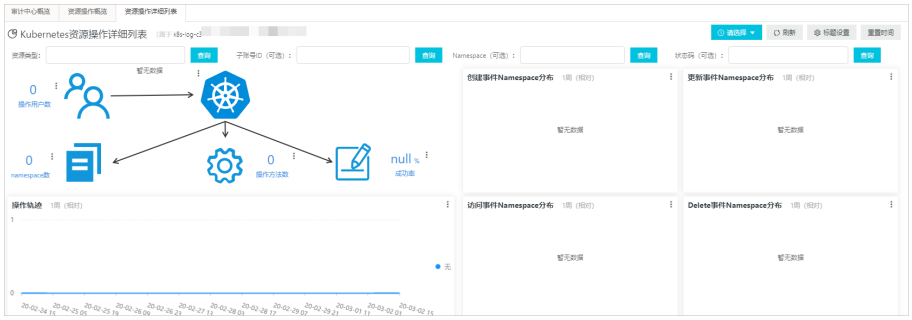
> **说明** 在该报表中，默认显示一周的统计信息。您可以自定义选择统计时间范围。此外，该报表支持指定Namespace、子账号ID（即RAM用户账号）、状态码进行筛选。您可以选择任意一项或多项组合筛选指定范围的事件。

- 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、RAM用户ID进行筛选。您可以选择任意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（即RAM用户账号）、状态码进行筛选。您可以选择任意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。

> **说明** 集群审计日志在日志服务中对应的日志库数据默认保存时间为30天，如果您需要修改日志的默认保存时间，请参见管理Logstore。

1. 登录日志服务控制台。
2. 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. 选择名称为audit-${clustered}的日志库（logstore），单击，选择**查询分析**，查看对应的审计日志。

> **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RAM用户的操作记录，直接输入RAM用户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，请参见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。详细操作方式请参见日志服务告警配置。

> **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**。详细操作方式请参见查看分析详情。

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:  ResponseStarted | 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：

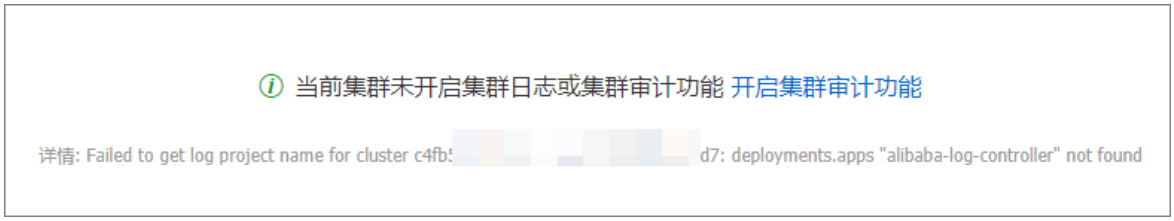
```
* | select ip as "源地址", total as "访问次数", round(rate * 100, 2) as "失败率%", failCount as "非法访问次数", CASE wh
en security_check_ip(ip) = 1 then 'yes' else 'no' end  as "是否高危IP",  ip_to_country(ip) as "国家", ip_to_province(
ip) as "省", ip_to_city(ip) as "市", ip_to_provider(ip) as "运营商" from (select CASE WHEN json_array_length(sourc
eIPs) = 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 =~ ".*"` 。

开通集群审计功能

创建Kubernetes集群时会默认选中**使用日志服务**，开启集群审计功能。若您还未开通审计报表，需手动开通审计报表。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在 Kubernetes 菜单下，单击左侧导航栏中的**集群 > 集群**，在目标集群右侧单击**管理**。
5. 在集群管理页左侧导航栏中，选择**安全管理 > 集群审计**。若您还未开通审计功能，会提示您需要手动开启。



6. 单击**开启集群审计功能**，选择**使用已有 Project** 或者**创建新Project**后，单击**确定**。

   当出现如下页面时，表示集群审计功能已开启。



更换日志Project

如果您想将集群审计日志数据迁移至另一个日志Project中，您可以使用集群审计中的**更换日志Project**功能。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**安全管理 > 集群审计**。
5. 在集群审计页面右上角，单击**更换日志Project**即可将集群审计日志的数据迁移至另一个日志服务的Project中。

关闭集群审计功能

如果您不再需要集群审计功能的话，可以通过以下方法关闭集群审计功能。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**安全管理 > 集群审计**。
5. 在**集群审计**页面右上角，单击**关闭集群审计**即可关闭当前集群的审计功能。

计费说明

- 您可以在费用账单的账单总览页面，查看计费明细，包含审计日志的费用信息。请参见费用账单。
- 审计日志计费方式，请参见按量付费。

支持第三方日志解决方案

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。

12.2.3. 部署服务账户令牌卷投影

您可以在容器服务Kubernetes版ACK（Container Service for Kubernetes）中开启**服务账户令牌卷投影**功能以降低在Pod中使用ServiceAccount遇到的安全性问题，此功能可使得kubelet支持基于Pod粒度的token签发，并且支持token audience和过期时间的配置。当token即将过期（过期时间的80%或者token存在大于24小时），kubelet支持对token的自动刷新。本文介绍如何配置及使用服务账户令牌卷投影。

背景信息

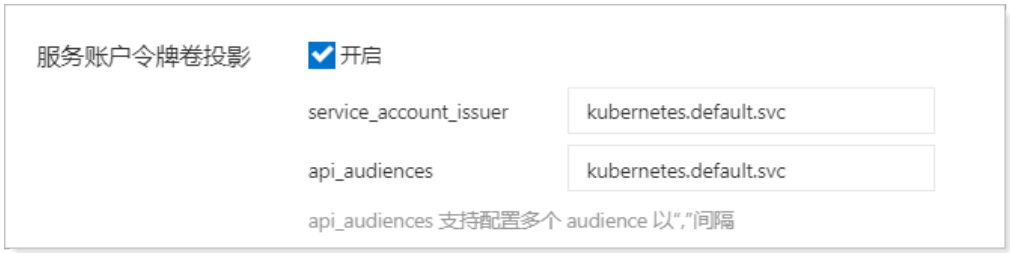
ServiceAccount是Pod和集群apiserver通讯的访问凭证。传统方式下，在Pod中使用ServiceAccount可能会遇到如下的安全挑战：

- ServiceAccount中的JSON Web Token (JWT) 没有绑定audience身份，因此所有ServiceAccount的使用者都可以彼此扮演，存在伪装攻击的可能。
- 传统方式下每一个ServiceAccount都需要存储在一个对应的secret中，并且会以文件形式存储在对应的应用节点上，而集群的系统组件在运行过程中也会使用到一些权限很高的ServiceAccount，其增大了集群管控平面的攻击面，攻击者可以通过获取这些管控组件使用的ServiceAccount非法提权。
- ServiceAccount中的JWT token没有设置过期时间，当上述ServiceAccount泄露情况发生时，您只能通过轮转ServiceAccount的签发私钥来进行防范，而client-go中还不支持这样的自动化流程，需要一个繁琐的手动运维流程。
- 每一个ServiceAccount都需要创建一个与之对应的secret，在大规模的应用部署下存在弹性和容量风险。

ACK同时配置了绑定服务账户令牌卷投影特性用于增强ServiceAccount的安全性，服务账户令牌卷投影可使Pod支持以卷投影的形式将ServiceAccount挂载到容器中从而避免了对secret的依赖。

步骤一、配置服务账户令牌卷投影

您可以通过在创建集群时选中**开启**，配置**服务账户令牌卷投影**功能。具体步骤，请参见创建ACK Pro版集群。



集群的系统组件apiserver和controller-manager中会自动开启绑定服务账户令牌卷投影feature-gate，同时在apiserver的启动参数中会增加以下配置。

参数	说明	默认值	控制台配置
service-account-issuer	serviceaccount token中的签发身份，即token payload中的iss字段。	kubernetes.default.svc	支持
api-audiences	合法的请求token身份，用于apiserver服务端认证请求token是否合法。	kubernetes.default.svc	支持，可以配置多个audience，通过`,`号分割。
service-account-signing-key-file	token签名私钥文件路径。	/etc/kubernetes/pki/sa.key	不支持，默认使用/etc/kubernetes/pki/sa.key，无需配置。

步骤二、部署服务账户令牌卷投影

当Pod中需要使用audience为vault并且有效期为2个小时的ServiceAccount时，您可以执行以下命令及使用以下模板配置PodSpec来使用服务账户令牌卷投影。

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
 name: build-robot
EOF
```

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
spec:
 containers:
 - image: nginx
   name: nginx
   volumeMounts:
   - mountPath: /var/run/secrets/tokens
     name: vault-token
 serviceAccountName: build-robot
 volumes:
 - name: vault-token
   projected:
     sources:
     - serviceAccountToken:
         path: vault-token
         expirationSeconds: 7200
         audience: vault
```

注意

- 请您确保Pod中能够实时获取到轮转后最新的token，也就是确保Pod逻辑中能够定期地重新加载目标token（建议为5分钟）。官方的Kubernetes在client-go 10.0.0版本后已经支持自动获取最新的token。
- 容器中ServiceAccount对应的token文件属性不再是644，在使用绑定服务账户令牌卷投影时，对应的token文件属性已经改为600（使用fsGroup特性时为640）。

相关文档

- Service Account Token Volume Projection
- Configure Service Accounts for Pods

12.2.4. 使用Pod安全策略

Kubernetes的Pod安全策略（Pod Security Policy）准入控制组件会基于您定义的规则验证在集群上创建和更新Pod的请求。如果创建或更新Pod的请求不符合定义的规则，系统将拒绝该请求并返回错误。本文将介绍如何在容器服务Kubernetes版ACK（Container Service for Kubernetes）中使用Pod安全策略。

前提条件

您已完成以下操作：

- 创建一个Kubernetes集群。
- 通过kubectl工具连接集群。

ACK默认的Pod安全策略

在ACK中，Kubernetes 1.16.6版本的标准专有集群和标准托管集群将默认启用Pod安全策略准入控制组件，并配置一个名为ack.privileged的Pod安全策略。这个安全策略将放行任意类型的Pod，效果等同于集群未开启Pod安全策略准入控制组件。

默认的Pod安全策略命令 详细规则的Pod安全策略命令

展开查看Pod安全策略、相应集群角色、集群角色绑定的完整YAML文件内容。 ❯

删除ACK默认Pod安全策略对应的集群角色绑定

警告 在删除ACK默认的Pod安全策略对应的集群角色绑定前必须先配置好自定义的Pod安全策略及其相应的RBAC绑定，否则所有用户、控制器、服务账号都将无法创建或更新Pod。

在配置好自定义的Pod安全策略及其相应的RBAC绑定后，您可以通过删除ACK默认Pod安全策略ack.privileged的集群角色绑定的方式来启用您自定义的Pod安全策略。

注意 请不要删除或修改名为ack.privileged的Pod安全策略以及名为ack:podsecuritypolicy:privileged的集群角色，ACK集群的正常运行需要依赖这两个资源。

展开查看删除ACK默认Pod安全策略ack.privileged的集群角色绑定命令。 ❯

配置或恢复ACK默认的Pod安全策略

展开查看配置或恢复使用ACK默认的Pod安全策略及其RBAC绑定命令。 ❯

相关文档

- Pod Security Policies

12.2.5. 安全巡检

CIS Kubernetes基线是CIS推出的一系列用于构建一个安全可靠的Kubernetes集群的安全配置建议。本文介绍如何在命令行下通过security-inspector组件提供的功能实现集群安全CIS基线检查。

前提条件

- 已创建一个ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 已在集群中安装security-inspector组件。具体操作，请参见管理组件。

CIS基线概述

CIS是互联网安全中心（Center for Internet Security）的缩写，它针对一些常见的系统制定了一套安全配置指南，称之为基线。CIS基线是由一组互联网安全专家共同讨论和制定的，这些安全指南被政府、企业和学术机构广泛认可和执行。

CIS Kubernetes基线是针对开源Kubernetes发行版本所编写的，旨在尽可能广泛应用于各个发行版本，同时不同的基线版本与特定的Kubernetes版本相关联。更多信息，请参见CIS Kubernetes 基线。

CIS也推出了更符合各大云厂商实际场景的CIS Kubernetes基线，例如针对ACK场景的CIS基线，即CIS Alibaba Cloud Container Service For Kubernetes（ACK）Benchmark。

使用security-inspector组件实现集群安全CIS基线检查

您可以通过以下步骤按照CIS Kubernetes基线标准扫描集群，并获取CSV格式的扫描结果报告。

1. 使用以下命令创建一个扫描任务。

   执行CIS基线检查，程序会根据集群版本选择合适的基线版本进行检查。

```
kubectl apply -f - <<EOF
apiVersion: securityinspector.alibabacloud.com/v1alpha1
kind: BenchmarkTask
metadata:
 name: cis-kubernetes-benchmark
spec:
 benchmarkVersion: 'cis-kubernetes-auto'
---
apiVersion: securityinspector.alibabacloud.com/v1alpha1
kind: BenchmarkJob
metadata:
 name: cis-kubernetes-benchmark
spec:
 taskName: cis-kubernetes-benchmark
EOF
```

   以上命令中benchmarkVersion字段的可用值如下，您可以根据需要选取合适的值（推荐使用cis-kubernetes-auto）。

benchmarkVersion字段的值	含义	适用集群
cis-kubernetes-auto	由组件根据集群版本自动选择最合适的CIS Kubernetes基线标准进行扫描。	Kubernetes 1.15及以上版本

benchmarkVersion字段的值	含义	适用集群
cis-kubernetes-ack-1.0	按照CIS Alibaba Cloud Container Service For Kubernetes（ACK）Benchmark v1.0.0基线标准进行扫描。	ACK专有版集群和托管版集群（Kubernetes 1.18及以上版本）
cis-kubernetes-1.20	按照CIS Kubernetes V1.20 Benchmark v1.0.0基线标准进行扫描。	Kubernetes 1.20及以上版本
cis-kubernetes-1.6	按照CIS Kubernetes Benchmark v1.6.0基线标准进行扫描。	Kubernetes 1.16~1.19版本
cis-kubernetes-1.5	按照CIS Kubernetes Benchmark v1.5.1基线标准进行扫描。	Kubernetes 1.15版本

2. 等待5分钟左右，通过以下命令确认本次扫描任务是否完成。

```
kubectl get benchmarkjobs.securityinspector.alibabacloud.com cis-kubernetes-benchmark -o 'jsonpath={.status.phase}'; echo
```

命令执行后返回 `Succeeded` 即表示扫描完成。

3. 确认扫描完成后，可以通过以下命令获取本次扫描结果的CSV格式报告。

```
for name in $(kubectl get benchmarkcsvresults.securityinspector.alibabacloud.com -l securityinspector.task.name=cis-kubernetes-benchmark -o name)
do
 filename="cis-$(echo $name | awk -F '/' '{print $2}')"; \
 kubectl get $name -o jsonpath='{.result.data}' > "$filename".csv; \
 echo "saved $filename.csv"
done
```

获取到报告后，对于报告中的结果您可以根据实际业务场景来判断是否需要采取措施，报告中各列的含义可以参考下文。

报告解读

上文扫描获取报告中的各列的说明如下表。

列名	含义	是否需要采取措施
Date	扫描时间	否
Result Schema	扫描时遵照的基线标准。取值如下： • cis-kubernetes-ack-1.0：CIS Alibaba Cloud Container Service For Kubernetes (ACK) Benchmark v1.0.0基线标准。 • cis-kubernetes-1.20：CIS Kubernetes V1.20 Benchmark v1.0.0基线标准。 • cis-kubernetes-1.6：CIS Kubernetes Benchmark v1.6.0基线标准。 • cis-kubernetes-1.5：CIS Kubernetes Benchmark v1.5.1基线标准。 关于各个基线标准的具体内容，请参见CIS Kubernetes基线标准文档。	否

列名	含义	是否需要采取措施
Node Name	当前报告是哪个集群节点的扫描报告。	否
Total Fail	计分检查项未通过数量。	关于说明信息，请参见下方Result列的说明。
Total Warn	需要关注的不计分检查项数量。	关于说明信息，请参见下方Result列的说明。
Total Pass	检查项通过数量。	否
Section Id	CIS基线标准中定义的章节编号。	否
Section Description	CIS基线标准中定义的章节简述。	否
Test Id	CIS基线标准中定义的检查项ID。	否
Test Description	CIS基线标准中定义的检查项简述。	否
Scored	检查项是否计分。取值如下： • Scored：计分。 • Not scored：不计分。	否
Test Remediation	检查项失败时推荐的调整/修复方法。 更多信息，请参见CIS Kubernetes基线标准文档。	关于说明信息，请参见下方Result列的说明。
Result	检查项结果。取值如下： • fail：计分检查项未通过。 • warn：检查项不计分需要关注。 • pass：检查通过。	各个值可采取的措施如下： • fail：强烈建议根据Test Remediation列的推荐方法进行调整或修复。您也可以根据您的实际业务场景来判断是否调整或修复。 • warn：保持关注，根据实际业务场景进行调整。 • pass：不需要采取措施。

相关文档

- CIS Kubernetes基线
- security-inspector

12.2.6. 使用ack-kubernetes-webhook-injector动态配置阿里云产品白名单策略

在对权限要求相对较高的云上的场景中，您需要将Pod的IP地址动态的加入或者移出指定RDS白名单，以实现对权限最细粒度的控制。使用ack-kubernetes-webhook-injector，您可以通过为Pod添加注解的方式，动态地将Pod的IP加入或者移出白名单。本文介绍如何安装及使用ack-kubernetes-webhook-injector组件。

前提条件

- 创建Kubernetes托管版集群
- 通过kubectl工具连接集群

背景信息

在云计算的使用场景中，某些资源需要配置相应的安全策略才能被外部访问，例如实例需要配置SLB访问控制策略，或者RDS数据库需要配置访问客户端的源IP地址白名单。虽然在创建ACK集群时可以将集群节点的IP网段加入到指定RDS的白名单中，但此方案有几个缺点：

- 白名单规则粒度太粗，加入白名单的IP网段覆盖了集群所有节点和Pod的IP地址。
- 集群删除后白名单规则需手动清理。

为了解决上述几点问题，阿里云容器服务ACK团队开发了ack-kubernetes-webhook-injector，它可以将指定Pod的IP地址动态加入RDS白名单或者安全组中，并且在Pod被删除时，将其IP从白名单或者安全组中移除，实现对安全策略的最细粒度的控制。

目前ack-kubernetes-webhook-injector支持的功能有：

- 随着Pod的创建与删除，自动将Pod的IP加入或移出指定的SLB访问控制策略组。
- 随着Pod的创建与删除，自动将Pod的IP加入或移出指定的阿里云Redis白名单。
- 随着Pod的创建与删除，自动将Pod的IP加入或移出指定的RDS白名单。

安装ack-kubernetes-webhook-injector

使用ack-kubernetes-webhook-injector组件前，您需要安装该组件。安装步骤如下：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 选择并单击**ack-kubernetes-webhook-injector**。
4. 在**参数**页签，将 `use_aksk` 的值设置为 `true` ，并设置AK信息。关于如何获取AK信息，请参见获取AccessKey。

```
openapi:
  # For test only. DO NOT use aksk authentication in production environments.
  use_aksk: true
  ak:
  sk:
```

5. 在页面右侧，选择目标集群后然后单击**创建**。

使用ack-kubernetes-webhook-injector动态添加RDS白名单示例

您只需要在Pod的副本控制器中的Pod Spec中使用注解（Annotation）指出Pod需加入的RDS实例ID和RDS白名单分组名称。在Pod创建的时候，ack-kubernetes-webhook-injector就会把Pod的IP地址加入到白名单或安全组的规则中，并且在Pod删除时移除规则。

Pod注解需包括：

- RDS白名单：
  - RDS实例ID：ack.aliyun.com/rds_id
  - RDS白名单分组名称：ack.aliyun.com/white_list_name

本文以自动添加RDS白名单规则为例，说明如何使用ack-kubernetes-webhook-injector为Pod动态添加安全规则。

1. 首先使用以下YAML示例创建一个Deployment，在其中的Pod定义中加入RDS实例ID和RDS白名单分组名称的注解。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  app: inject-test
 name: inject-test
spec:
 replicas: 1
 selector:
  matchLabels:
   app: inject-test
 template:
  metadata:
   annotations:
    ack.aliyun.com/rds_id: <rm-wz9nanjcud75bxxxx>
    ack.aliyun.com/white_list_name: <rds_group>
   labels:
    app: inject-test
  spec:
   containers:
   - command:
    - sleep
    - "3600"
    image: alpine:latest
    name: inject-test
```

2. 执行以下命令查看Pod的IP地址。

```
kubectl --kubeconfig .kube/config_sts_test -n inject-test get pod -o wide
```

输出：

```
NAME                READY     STATS    RESTARTS   AGE   IP          NODE
inject-test-68cc8f9bbf-gj86n   1/1      Running     0     22s  172.25.0.28   cn-hangzhou.xxx
```

可以看出Pod的IP地址是172.25.0.28。

3. 登录RDS控制台查看注解中RDS实例中的白名单。有关如何查看RDS的白名单，请参见高安全白名单模式IP白名单。
可以看到规则已经自动添加。



4. 将步骤一中的Deployment的副本数设置为0，再检查RDS白名单。
可以看到刚才创建的规则被自动删除。

> **说明** 用ack-kubernetes-webhook-injector将Pod IP地址加入安全组的步骤与加入RDS类似。

卸载ack-kubernetes-webhook-injector

如果您不再使用ack-kubernetes-webhook-injector，可以在ACK发布功能中将其删除，具体步骤，请参见删除发布。同时请执行以下命令清理配置信息。

```
kubectl -n kube-system delete secret kubernetes-webhook-injector-certs
kubectl delete mutatingwebhookconfigurations.admissionregistration.k8s.io kubernetes-webhook-injector
```

更多安全策略配置规则

ack-kubernetes-webhook-injector也支持以下安全策略配置：

- SLB访问控制：`ack.aliyun.com/access_control_policy_id`。
- 阿里云Redis白名单配置：
  - Redis ID：`ack.aliyun.com/redis_id`。
  - Redis白名单分组：`ack.aliyun.com/white_list_name`。

12.2.7. 自定义集群API Server证书SAN

ACK集群的API Server服务端证书中SAN（Subject Alternative Name）字段默认包括集群本地域名、API Server负载均衡的内网IP、API Server服务本地IP和公网EIP等字段。如果您有特殊的代理访问或跨域访问需求，可以通过自定义SAN字段实现。本文介绍新建集群如何自定义API Server证书SAN和已有集群如何自定义更新API Server证书SAN。

前提条件

目前支持托管版集群（ACK标准托管版集群、ACK Pro托管版集群等）及ASK集群。以下仅枚举三种托管版集群的创建方法，您只需要创建以下集群中的任意一种：

- 创建Kubernetes托管版集群
- 创建ACK Pro版集群
- 创建托管GPU集群

背景信息

SAN是一项对X.509标准的扩展，它允许在SSL安全证书中使用`subjectAltName`字段将多种值（包括IP地址、域名、URI和电子邮件等）与证书关联。

使用限制

- 创建ASK集群时不支持自定义API Server证书的SAN字段。
- 支持在已有ASK集群中自定义更新API Server证书的SAN字段。

自定义新建集群API Server证书SAN

本文以创建Kubernetes托管版集群为例说明新建集群如何自定义API Server证书SAN。

在创建集群的**集群配置**步骤，单击**显示高级选项**。在**自定义证书SAN**配置项中输入配置字段。具体操作，请参见创建Kubernetes托管版集群。

> 说明 自定义证书SAN字段的配置，支持输入符合规范的自定义IP、域名和URI，多个字段以英文逗号（,）分隔。



上图中的**自定义证书SAN**字段中包含两个域名和一个IP。

更新已有集群API Server证书SAN

> 注意 更新或修改自定义证书SAN会触发集群API Server短暂重启，请您在业务非高峰时段操作。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群信息管理页面选择**基本信息**页签，单击**自定义证书SAN**右侧的**更新**。
5. 在**更新自定义证书SAN**对话框中，配置**自定义证书SAN**字段。
6. 参数配置完成后，单击**确定**。

12.3. 容器应用安全

12.3.1. 使用PSP安全策略

PodSecurityPolicy（简称PSP）是Kubernetes中Pod部署时重要的安全校验手段，能够有效地约束应用运行时行为安全。本文主要介绍如何创建、管理和绑定PSP策略。

前提条件

- 创建一个Kubernetes托管版或专有版集群，且集群版本至少为1.14.8-aliyun.1，请参见创建Kubernetes托管版集群。
- 以主账号（即阿里云账号）登录控制台，或赋予子账号（即RAM用户）管理员权限，请参见配置RAM用户RBAC权限，然后以子账号登录控制台。
- 如果您使用子账号进行PSP策略管理，子账号需要拥有以下RAM授权条件：
  - `cs:DescribeClusterPSPState`：获取集群PSP开启状态。
  - `cs:DescribeServiceAccountBindingPSP`：获取指定`serviceaccount`绑定的PSP策略。
  - `cs:UpdateClusterPodSecurityPolicy`：更新集群中的PSP策略模型实例。
  - `cs:CreatePSPBinding`：创建对指定`serviceaccount`的PSP绑定。
  - `cs:UnbindingServiceAccountPSP`：解绑指定`serviceaccount`的PSP策略模型实例。

  更多自定义RAM授权策略，请参见自定义RAM授权策略。

背景信息

在Kubernetes Pod定义中，可以通过设定安全参数来约束应用容器运行时的行为。作为集群的管理员，则会希望在集群的安全纵深防护中具有像AppArmor，SELinux这样的策略引擎，能够在应用部署时对应用Pod进行强制的安全校验，而PSP正是这样的解决方案。

- 强制校验：PSP是Kubernetes原生的集群维度资源模型，通过API server的admission准入机制，在应用部署时刻对Pod的安全参数进行强制校验。如果参数配置不满足指定PSP策略的定义，API server会禁止该Pod的部署请求。
- PSP策略的RBAC绑定：创建PSP策略，需要开启PSP对应的admission plugin，同时建立指定Pod部署凭证对PSP策略的使用权限的RBAC绑定。
- 开启PSP特性：当前ACK新创建的托管和专有集群默认开启PSP特性，减少开启PSP带来的运维工作。
- 权限最小化原则：为了保证集群使用上的兼容性，默认给所有合法认证用户绑定了一个ack.privileged的特权策略，而在PSP的实际应用场景中，我们希望策略遵循权限最小化的安全原则，比如禁止部分用户部署特权容器，只能使用只读的根文件系统，或者只能挂载指定范围的host目录。

创建PSP策略

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群详情页面选择**安全管理 > 策略管理**。
5. 在**策略管理**页面单击**创建**，在**选择策略模板**编辑框中选择目标策略模板，单击目标策略模板上的**使用**。

   除了集群默认设置的特权策略（默认放开所有安全校验）ack.privileged之外，默认提供了两个PSP策略实例模板：

   - privileged-restricted：限制 `privileged` ，共用主机网络、端口、Namespace等特权容器的部署。
   - privileged-root-volumes-restricted：限制 `privileged` ，共用主机网络、端口、Namespace等特权容器的部署，同时限制挂载卷的类型和root权限的使用。
6. 在**创建YAML**页面根据不同的安全等级需求修改对应的配置项，从而创建不同应用场景下的PSP策略。PSP模板中的配置项详细说明请参见Pod Security Policies。

配置项	描述
privileged	启动特权容器。
hostPID，hostIPC	使用主机Namespaces。
hostNetwork，hostPorts	使用主机网络和端口。
volumes	允许使用的挂载卷类型。
allowedHostPaths	允许hostPath类型挂载卷在主机上挂载的路径，通过pathPrefix字段声明允许挂载的主机路径前缀组。
allowedFlexVolumes	允许使用的指定FlexVolume驱动。
fsGroup	配置Pod中挂载卷使用的辅组ID。
readOnlyRootFilesystem	约束启动Pod使用只读的root文件系统。
runAsUser，runAsGroup，supplementalGroups	指定Pod中容器启动的用户ID以及主组和辅组ID。
allowPrivilegeEscalation，defaultAllowPrivilegeEscalation	约束Pod中是否允许配置allowPrivilegeEscalation=true，该配置会控制setuid的使用，同时控制程序是否可以使用额外的特权系统调用。
defaultAddCapabilities，requiredDropCapabilities，allowedCapabilities	控制Pod中使用的Linux Capabilities。
seLinux	控制Pod使用seLinux配置。
allowedProcMountTypes	控制Pod允许使用的ProcMountTypes。
annotations	配置Pod中容器使用的AppArmor或seccomp。
forbiddenSysctls，allowedUnsafeSysctls	控制Pod中容器使用的sysctl配置。

7. 单击**确定**。

   策略实例创建成功后，您可以在**策略管理**页面中查看创建的PSP策略实例。

> 说明
> - 在**策略管理**页签，单击目标策略**操作**列的YAML，可以在弹出的窗口中修改PSP策略。
> - 在**策略管理**页签，单击目标策略**操作**列的**删除**，删除PSP策略。

绑定PSP策略

您需要给应用部署者赋予相应的PSP策略使用权限，否则如果开启了PSP特性且应用部署者没有绑定任何的PSP策略（已经删除了默认特权绑定），部署的Pod会遇到以下错误。

```
Error from server (Forbidden): error when creating xxx: pods "xxx" is forbidden: unable to validate against any pod s
ecurity policy: []
```

您可以对指定用户的服务账号或某个命名空间下的服务账号绑定PSP策略。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群详情页面选择**安全管理 > 策略管理**。
5. 选择**策略绑定**页签，单击**新增绑定规则**。
6. 在**新增绑定规则**编辑框中设置**命名空间**、**服务账号**、**绑定策略**。
7. 单击**确定**。

解除默认绑定

为了使用已经创建的PSP策略和绑定的约束能力，您需要删除默认的特权绑定。

- 为了确保系统组件和已部署应用的正常运行，在解除默认特权绑定的同时，后台会自动给kube-system命名空间下的所有serviceaccount对象和系统默认的system:nodes用户组绑定特权策略的使用权限，以确保系统组件的正常工作。
- 对于已经存在的命名空间，如果在命名空间下您没有通过策略管理控制台创建任何的PSP策略绑定，后台会给该命名空间下的所有serviceaccount对象绑定特权策略的使用权限，以确保该命名空间中已有应用的正常启动。

> 注意
> - 对于已经在控制台创建了策略绑定的命名空间。
> 为了使策略绑定的约束能力生效，在解除绑定时后台不会自动创建该命名空间下的特权策略绑定，因此在解除绑定前请您务必确认PSP策略和绑定的正确性。
> - 对于新创建的命名空间。
> 容器服务Kubernetes版后台不会自动为新命名空间下的serviceaccount对象创建对应的特权策略绑定，请您在新命名空间创建后及时创建对应的PSP策略和绑定对象。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群详情页面选择**安全管理 > 策略管理**。
5. 在**策略管理使用引导**区域中单击**解除默认绑定**。
6. 在**解绑**对话框中单击**确定**。

确保PSP的admission controller为开启状态

当前ACK新建的集群默认开启了PSP的admission controller。然而对于运行中的已有集群，为了使上述PSP操作生效，我们需要确保集群API server的PSP admission controller为开启状态。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群详情页面选择**安全管理 > 策略管理**。
5. 开启admission controller。
   - 如果在**策略管理**页面右上角显示，则单击**改变PSP状态**，在**PSP状态变更**对话框中单击**确定**，开启admission controller。
   - 如果在**策略管理**页面右上角显示  ，则表示已开启admission controller。

> **注意** 开启PSP会重启集群API server，如果集群中存在依赖API server的业务服务，请选择非业务非高峰期进行状态变更。

执行结果

下面将通过示例展示PSP策略管理的应用以及具体的安全校验应用效果。

1. 通过kubectl工具连接集群。
2. 创建privileged-restricted和privileged-root-volumes-restricted策略模板，详细介绍请参见创建PSP策略。
3. 执行以下命令，创建2个测试命名空间，同时在命名空间下创建测试用的serviceaccount实例并建立相应的RBAC权限绑定。

```
kubectl create ns ack-all-restrictive
kubectl create ns ack-restrictive
kubectl create sa ack-dev -n ack-restrictive
kubectl create sa ack-tester -n ack-all-restrictive
kubectl -n ack-restrictive create rolebinding ack-dev-editor \
    --clusterrole=edit \
    --serviceaccount=ack-restrictive:ack-dev
kubectl -n ack-all-restrictive create rolebinding ack-tester-editor \
    --clusterrole=edit \
    --serviceaccount=ack-all-restrictive:ack-tester
```

4. 在**策略绑定**页签中绑定PSP策略。
   i. 单击**新增绑定规则**，在**新增绑定规则**编辑框中设置**命名空间**为ack-restrictive，设置**服务账号**为 ack-dev，设置**绑定策略**为 privileged-restricted，单击**确定**。
   ii. 单击**新增绑定规则**，在**新增绑定规则**编辑框中设置**命名空间**为ack-restrictive，设置**服务账号**为 ack-dev，设置**绑定策略**为privileged-root-volumes-restricted，单击**确定**。
5. 执行以下命令，扮演使用ack-restrictive命名空间下的服务账号ack-dev和ack-all-restrictive命名空间下的服务账号ack-tester。

```
alias kubectl-dev='kubectl --as=system:serviceaccount:ack-restrictive:ack-dev -n ack-restrictive'
alias kubectl-tester='kubectl --as=system:serviceaccount:ack-all-restrictive:ack-tester -n ack-all-restrictive'
kubectl-dev auth can-i use  podsecuritypolicy/privileged-restricted
Warning: resource 'podsecuritypolicies' is not namespace scoped in group 'policy'
yes
```

6. 执行以下命令，创建共享主机网络的特权容器。

```
kubectl-dev apply -f- <<EOF
apiVersion: v1
kind: Pod
metadata:
 name: privileged
spec:
 hostNetwork: true
 containers:
  - name: busybox
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
EOF
Error from server (Forbidden): error when retrieving current configuration of:
Resource: "/v1, Resource=pods", GroupVersionKind: "/v1, Kind=Pod"
Name: "privileged", Namespace: "ack-restrictive"
Object: &{map["apiVersion":"v1" "kind":"Pod" "metadata":map["annotations":map["kubectl.kubernetes.io/last-applied-configuration":""] "name":"privileged" "namespace":"ack-restrictive"] "spec":map["containers":[map["command":["sh" "-c" "sleep 1h"] "image":"busybox" "name":"busybox"]] "hostNetwork":%!q(bool=true)]]}
from server for: "STDIN": pods "privileged" is forbidden: User "system:serviceaccount:ack-restrictive:ack-dev" cannot get resource "pods" in API group "" in the namespace "ack-restrictive"
```

系统显示报错，创建特权容器失败。

7. 执行以下命令，创建非特权容器。

```
kubectl-dev apply -f- <<EOF
apiVersion: v1
kind: Pod
metadata:
 name: non-privileged
spec:
 containers:
  - name: busybox
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
EOF
pod/non-privileged created
```

系统显示正常，创建非特权容器成功。

8. 执行以下命令，创建一个root权限Pod和一个非root权限Pod。

```
kubectl-tester apply -f- <<EOF
apiVersion: v1
kind: Pod
metadata:
 name: root-pod
spec:
 containers:
  - name: busybox
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
EOF
kubectl-tester apply -f- <<EOF
apiVersion: v1
kind: Pod
metadata:
 name: non-root-pod
spec:
 containers:
  - name: busybox
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    securityContext:
     runAsUser: 1000
     runAsGroup: 3000
EOF
```

9. 执行以下命令，查看Pod状态。

   使用root权限的容器创建失败，使用的ack-tester sa绑定的PSP策略会约束pod使用非root权限启动容器。
   命令如下：

   ```
   kubectl-tester get po
   ```

   结果如下：

   ```
   NAME       READY  STATUS                 RESTARTS  AGE
   non-root-pod  1/1    Running                0      115s
   root-pod     0/1    CreateContainerConfigError  0      24s
   ```

10. 执行以下命令，查看错误详情。

    命令如下：

    ```
    kubectl describe pod root-pod
    ```

    结果如下：

    ```
    Warning  Failed    28s (x5 over 2m1s)   kubelet, cn-shenzhen.192.168.1.52  Error: container has runAsNonRoot a
    nd image will run as root
    ```

12.3.2. 使用配置巡检检查集群Workload安全隐患

本文将介绍如何使用配置巡检功能来扫描集群中Workload配置的安全隐患和介绍巡检报告相关信息，帮助您实时了解当前状态下运行应用的配置是否有安全隐患。

前提条件

您已经成功创建一个Kubernetes 1.14.8或以上版本的托管版或专有版集群。具体操作，请参见创建Kubernetes托管版集群。

子账号（即RAM用户）RAM授权

如果您当前使用的是子账号，请先完成ACK容器服务配置巡检页面的RAM授权操作，确保当前子账号拥有操作当前集群配置巡检页面的权限，否则会出现权限不足无法操作配置巡检页面功能的问题。具体操作，请参见自定义RAM授权策略。

```
{
 "Statement": [
  {
   "Action": [
    "cs:DescribePolarisConfig",
    "cs:DescribePolarisJob",
    "cs:DescribePolarisCronJob",
    "cs:UpdatePolarisJob",
    "cs:UpdatePolarisCronJob"
   ],
   "Effect": "Allow",
   "Resource": [
    "acs:cs:*:*:cluster/<yourclusterID>"
   ]
  }
 ],
 "Version": "1"
}
```

如果您当前使用的是子账号，且需要使用巡检报告功能，请先完成日志服务指定 logproject （当前集群logtail-ds组件所使用的 logproject ）的RAM授权，确保当前子账号拥有日志服务指定 logproject 的数据读取权限，否则会出现权限不足无法查看巡检报告的问题。具体操作，请参见RAM自定义授权场景。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:Get*",
        "log:List*"
      ],
      "Resource": "acs:log:*:*:project/<指定的Project名称>/*",
      "Effect": "Allow"
    }
  ]
}
```

子账号RBAC授权

如果您当前使用的是子账号，请先完成ACK容器服务配置巡检页面涉及资源的RBAC授权，需要授予子账号指定集群的管理员权限，以确保子账号拥有操作配置巡检页面中涉及的Kubernetes资源的权限。具体操作，请参见配置RAM用户RBAC权限。

扫描集群Workload

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏选择**安全管理 > 配置巡检**。
5. 如果您未安装巡检组件的话，在**安装组件**下方单击**开始安装**。如果您已经安装请忽略此步骤。

6. 在**配置巡检**页面右上方，单击**立即执行巡检**。
7. 等待巡检扫描完成后，单击刷新图标查看相应的巡检报告。
8. （可选）在**配置巡检**页面右上方，单击**巡检配置**，开启或禁用定期自动执行配置巡检的功能以及配置巡检时执行的检查项。

巡检详情

巡检详情页面通过表格化的方式展示不同工作负载的详细巡检结果，主要包括以下功能：

- 直观展示工作负载的巡检**通过项**和**风险项**数量。
- 在检查结果页面展示巡检结果的各个检查项的通过状态、检查项详细说明以及加固建议。
- 支持为工作负载设置检查项白名单。
- 支持**命名空间**、**工作负载类别**、**是否有风险**等条件对结果进行过滤查询。

巡检报告

巡检报告中主要展示最近执行的巡检扫描结果，主要包括以下几类信息：

- 扫描结果概览，包括总的检查条目数、检查各个资源项数目及百分比、整体的健康度。
- 各大类扫描结果统计，包括报告健康检查、镜像、网络、资源、安全等大类的结果。
- 各个Workload配置的详细扫描结果，包括资源类别、资源名称、命名空间、检查类型、检查项、检查结果等内容。

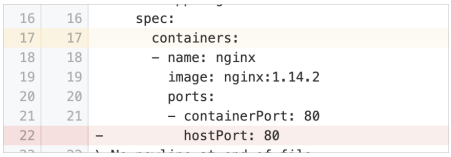
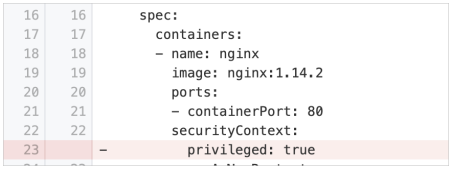
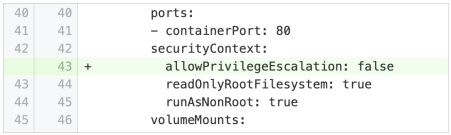
配置巡检功能会扫描并展示以下检查项的扫描结果。

检查项名称	检查的内容及安全风险	修复建议

hostNetworkSet	通过检查Workload的Pod Spec中是否配置了hostNetwork：true，检查是否配置了共享使用主机的网络namespace。如果配置了，存在Pod中容器攻击主机网络、嗅探主机网络数据的风险。	修改Pod Spec，删除hostNetwork字段。
		示例：
		`@@ -14,7 +14,6 @@ spec:`
		`labels:`
		`app: nginx`
		`spec:`
		`- hostNetwork: true`
		`containers:`
		`- name: nginx`
		`image: nginx:1.14.2`

hostIPCSet	通过检查Workload的Pod Spec中是否配置了hostIPC：true，检查是否配置了共享使用主机的IPC namespace。如果配置了，存在Pod中容器攻击主机上进程、嗅探主机上进程数据的风险。	修改Pod Spec，删除hostIPC字段。
		示例：
		`@@ -14,7 +14,6 @@ spec:`
		`labels:`
		`app: nginx`
		`spec:`
		`- hostIPC: true`
		`containers:`
		`- name: nginx`
		`image: nginx:1.14.2`

hostPIDSet	通过检查Workload的Pod Spec中是否配置了hostPID：true，检查是否配置了共享使用主机的PID namespace。如果配置了，存在Pod中容器攻击主机上进程、采集主机上进程数据的风险。	修改Pod Spec，删除hostPID字段。
		示例：
		`@@ -14,7 +14,6 @@ spec:`
		`labels:`
		`app: nginx`
		`spec:`
		`- hostPID: true`
		`containers:`
		`- name: nginx`
		`image: nginx:1.14.2`

检查项名称	检查的内容及安全风险	修复建议
hostPortSet	通过检查Workload的Pod Spec中是否配置了hostPort，检查是否配置了把容器中监听的端口映射到主机指定端口上。如果配置了，存在挤占主机可用端口以及被非预期的请求方请求容器端口的风险。	修改Pod Spec，删除hostPort字段。 示例： 
runAsRootAllowed	通过检查Workload的Pod Spec中是否未配置runAsNonRoot：true，检查是否未配置使用非root用户运行容器。如果未配置，存在被容器中的恶意进程入侵用户应用、入侵主机甚至入侵整个集群的风险。	修改Pod Spec，增加runAsNonRoot：true。 示例： 
runAsPrivileged	通过检查Workload的Pod Spec中是否配置了privileged：true，检查是否配置了允许以特权模式运行容器。如果配置了，存在被容器中的恶意进程入侵用户应用、入侵主机甚至入侵集群的风险。	修改Pod Spec，删除privileged字段。 示例： 
privilegeEscalationAllowed	通过检查Workload的Pod Spec中是否未配置allowPrivilegeEscalation：false，检查是否未配置禁止容器中的子进程拥有比父进程更高的权限。如果未配置，存在被容器中的恶意进程实现越权操作的风险。	修改Pod Spec，增加allowPrivilegeEscalation：false字段。 示例： 

检查项名称	检查的内容及安全风险	修复建议

capabilitiesAdded	通过检查Workload的Pod Spec中capabilities字段，检查是否配置了允许容器中的进程拥有SYS_ADMIN、NET_ADMIN、ALL等特权Linux Capabilities。如果配置了，存在被容器中的恶意进程通过这些特权入侵用户应用、入侵或破坏组件/集群的风险。	修改Pod Spec，根据实际需求只添加必需的Linux Capabilities，删除不需要的Linux Capabilities。
		不依赖额外Linux Capabilities，删除所有不需要的Linux Capabilities。示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`  capabilities:`
		`- add:`
		`-  - SYS_ADMIN`
		`-  - NET_ADMIN`
		`-  - KILL`
		`+ drop:`
		`+  - ALL`
		只添加必需的Linux Capabilities，删除所有不需要的Linux Capabilities。示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`  capabilities:`
		`  add:`
		`-  - SYS_ADMIN`
		`-  - NET_ADMIN`
		`-  - KILL`
		`+  - CHOWN`
		`+ drop:`
		`+  - ALL`

检查项名称	检查的内容及安全风险	修复建议

notReadOnlyRootFileSystem	通过检查Workload的Pod Spec中是否未配置readOnlyRootFilesystem：true，检查是否未配置容器中的文件系统是不可修改的。如果未配置的话，存在被容器中的恶意进程恶意修改系统文件的风险。	修改Pod Spec，增加readOnlyRootFilesystem: true，如果有需要修改某个目录下文件的需求，可以通过volumeMounts实现。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`+  readOnlyRootFilesystem: true`
		如果有需要修改某个目录下文件的需求，通过volumeMounts实现修改文件的需求。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`+  readOnlyRootFilesystem: true`
		`  runAsNonRoot: true`
		`+  volumeMounts:`
		`+  - mountPath: /path/to/write`
		`+  name: writeable`
		`+  volumes:`
		`+  - emptyDir: {}`
		`+  name: writeable`

cpuRequestsMissing	通过检查Workload的Pod Spec中是否未配置resources.requests.cpu字段，检查是否未配置运行容器所需的最少CPU资源。如果未配置的话，Pod有被调度到资源紧张的节点上的风险，可能会出现容器内进程运行缓慢的风险。	修改Pod Spec，增加resources.requests.cpu字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`+  resources:`
		`+  requests:`
		`+  cpu: 100m`

cpuLimitsMissing	通过检查Workload的Pod Spec中是否未配置resources.limits.cpu字段，检查是否未配置运行容器所需的最大CPU资源。如果未配置的话，存在被容器内的异常进程消耗大量节点资源，甚至把整个节点或集群的资源销耗殆尽的风险。	修改Pod Spec，增加resources.limits.cpu字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  resources:`
		`  requests:`
		`  cpu: 100m`
		`+  limits:`
		`+  cpu: 100m`

检查项名称	检查的内容及安全风险	修复建议

memoryRequestsMissing	通过检查Workload的Pod Spec中是否未配置resources.requests.memory字段，检查是否未配置运行容器所需的最少内存资源。如果未配置，Pod有被调度到资源紧张的节点上的风险，可能会出现容器内进程OOM的风险。	修改Pod Spec，增加resources.requests.memory字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  resources:`
		`  requests:`
		`  cpu: 100m`
		`+ memory: 128Mi`
		`  limits:`

memoryLimitsMissing	通过检查Workload的Pod Spec中是否未配置resources.limits.memory字段，检查是否未配置运行容器所需的最大内存资源。如果未配置的话，存在被容器内的异常进程消耗大量节点资源，甚至把整个节点或集群的资源销耗殆尽的风险。	修改Pod Spec，增加resources.limits.memory字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  resources:`
		`  requests:`
		`  cpu: 100m`
		`  memory: 128Mi`
		`  limits:`
		`  cpu: 100m`
		`+ memory: 128Mi`
		`  ports:`

readinessProbeMissing	通过检查Workload的Pod Spec中是否未配置readinessProbe字段，检查是否未配置检测容器内应用能否正常处理请求的探针。如果未配置的话，存在在容器内应用异常无法处理请求时仍旧有请求发过来导致业务异常的风险。	修改Pod Spec，增加readinessProbe字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`+ readinessProbe:`
		`+ httpGet:`
		`+ path: /health`
		`+ port: 8080`
		`+ initialDelaySeconds: 5`
		`+ periodSeconds: 20`
		`  resources:`

livenessProbeMissing	通过检查Workload的Pod Spec中是否未配置livenessProbe，检查是否未配置检测容器内应用是否出现异常需要重启容器的探针。如果未配置，存在在容器内应用异常需要重启容器才能恢复时未及时重启导致业务异常的风险。	修改Pod Spec，增加livenessProbe字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`+ livenessProbe:`
		`+ httpGet:`
		`+ path: /health`
		`+ port: 8080`
		`+ initialDelaySeconds: 5`
		`+ periodSeconds: 20`
		`  readinessProbe:`

检查项名称	检查的内容及安全风险	修复建议

tagNotSpecified	通过检查Workload的Pod Spec中的image字段的值是否未包含镜像Tag或者使用了latest作为镜像Tag，检查是否未配置运行容器时使用指定Tag的容器镜像。如果未配置，存在在运行容器时运行了非预期的容器镜像版本导致业务异常的风险。	修改Pod Spec，修改image字段，使用指定的镜像Tag并且不要使用latest作为镜像Tag。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`-  image: nginx:latest`
		`+  image: nginx:1.14.2`

anonymousUserRBACBinding	通过检查集群内的RBAC（Role-based access control）绑定找出配置了匿名用户访问权限的配置项。如果配置了允许匿名用户访问集群资源的配置项，则存在被恶意匿名用户窃取集群敏感信息、攻击和入侵集群的风险。	修改扫描出来的RBAC绑定，根据实际情况删除允许匿名用户访问集群资源的权限配置项。
		示例：
		`- apiGroup: rbac.authorization.k8s.io`
		`  kind: Group`
		`  name: 'foo-group'`
		`- - apiGroup: rbac.authorization.k8s.io`
		`-  kind: User`
		`-  name: 'system:anonymous'`
		`- - apiGroup: rbac.authorization.k8s.io`
		`-  kind: Group`
		`-  name: 'system:unauthenticated'`

相关文档

- Configure a Security Context for a Pod or Container
- Configure Liveness, Readiness and Startup Probes

12.3.3. 使用运行时刻安全监控

运行时刻安全监控提供监控和告警能力，包括恶意镜像启动、病毒和恶意程序的查杀、容器内部入侵行为、容器逃逸和高风险操作预警等主要的容器侧攻击行为。本文介绍如何使用针对指定ACK集群的运行时刻安全监控。

前提条件

- 已创建一个Kubernetes集群，具体操作，请参见创建Kubernetes托管版集群。
- 已开启云安全中心服务，具体操作，请参见购买云安全中心。
- 如果您使用的是子账号（即RAM用户），请确保子账号有云安全中心的RAM只读访问权限AliyunYundunSASReadOnlyAccess。

背景信息

当容器应用通过API Server的认证鉴权和准入控制校验成功部署后，在云原生应用零信任的安全原则下，还需要在容器应用的运行时刻提供相应的安全监控和告警能力。为此，容器服务和云安全中心深度集成告警处理和漏洞检测能力，集群管理员可以在应用运行时提供监控和告警能力，包括恶意镜像启动，病毒和恶意程序的查杀，容器内部入侵行为，容器逃逸和高风险操作预警等主要的容器侧攻击行为。您可以在集群详情页实时接收到相应告警，并根据页面提示查看和处理告警详情。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页面左侧导航栏中，选择**安全管理 > 运行时刻安全监控**，在**运行时刻安全监控**页面查看监控和告警信息。
   - 在**运行时刻安全监控**页面上方显示了集群的安全概况，包括以下几个方面：
     - 安全检测：显示检测到的安全风险告警事件。您可以单击**查看更多**，跳转到云安全中心总览页面，查看详细事件。

- 节点状态：显示健康节点和存在风险节点。
- 防御能力：显示待处理的告警事件、后台扫描agent中病毒库更新时间和上次系统漏洞的扫描时间。

- 在**运行时刻安全监控**页面下方，单击**安全告警处理**页签，显示了集群中实时监控到的安全事件，包括在容器中或在主机层面发生的病毒和恶意程序攻击，容器内部的入侵行为，容器逃逸和高风险操作预警等主要的容器侧攻击行为，安全告警的详细说明请参见安全告警概述。
  - 在**安全告警处理**页面单击目标安全告警**操作**列的**处理**，您可以在弹出的对话框中将该安全告警加入白名单或者忽略该安全告警。
  - 在**安全告警处理**页面单击目标安全告警**操作**列的**详情**，在告警事件的详情页面显示安全告警事件详细信息，包括事件的发生时间，受影响资产信息，进程ID等相关信息。在**详情**页面单击**溯源**页签，在溯源页面可以对攻击事件进行自动化溯源并提供原始数据预览。
- 在**运行时刻安全监控**页面下方，单击**漏洞信息**页签，显示集群节点上存在的CVE漏洞。
  单击详细漏洞页签，单击目标漏洞**操作**列的**修复**，跳转到漏洞修复详情页，您可以选择目标资产进行一键修复。

> **说明** 漏洞信息页面显示的CVE漏洞包含Linux软件漏洞、Windows系统漏洞、Web-CMS漏洞、应用漏洞、应急漏洞，详细介绍请参见以下内容：
>
> - Linux软件漏洞
> - Windows系统漏洞
> - Web-CMS漏洞
> - 应用漏洞
> - 应急漏洞

12.4. 容器安全FAQ

本文主要为您介绍安全组常见问题。

- 为什么容器之间网络不通？
- 如何给Kubernetes集群指定安全组？
- 集群审计功能是否可以取消或者在创建集群后再部署？
- Kubernetes专有版本集群怎么更换证书有效期，以及更换各个组件的证书？
- Pod无法创建，报错详情：no providers available to validate pod request
- Secret在新建的Namespace下面无法使用
- 无法挂载default-token
- 审计日志查询方法
- 如何收集Kubernetes集群的诊断信息？
- 如何收集边缘Kubernetes集群节点的诊断信息？
- CentOS 7.6系统的Kubernetes集群中kubelet日志含有“Reason:KubeletNotReady Message:PLEG is not healthy:”信息

为什么容器之间网络不通？

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。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv. 选择**集群资源**页签，单击**虚拟专有网络 VPC**右侧的链接。

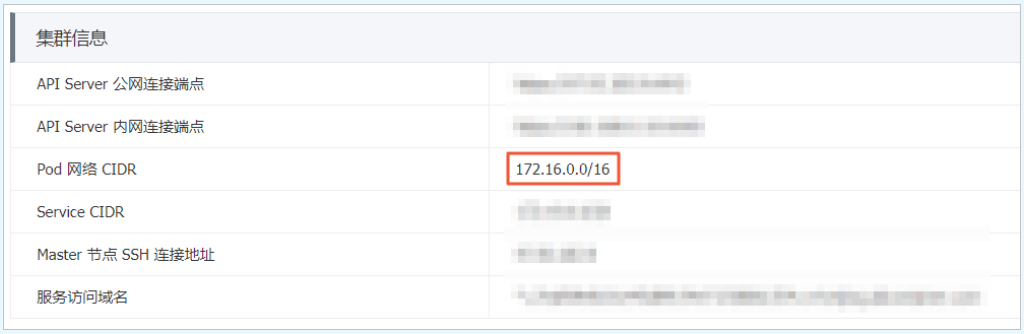


v. 在**资源管理**页签，单击**安全组**下的数字。

vi. 单击目标安全组**操作列**的**配置规则**。

vii. 在**入方向**页面，单击**手动添加**。

viii. 填写**协议类型**、**端口范围**、**授权对象**，并单击**保存**。

> **说明**
>
> - **协议类型**请选择**全部**。
> - **授权对象**填写为Pod的网段地址。
>
> Pod的网段地址（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，可参见安全组应用案例ECS安全组配置操作指南。

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。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv. 选择**集群资源**页签，单击**虚拟专有网络 VPC**。



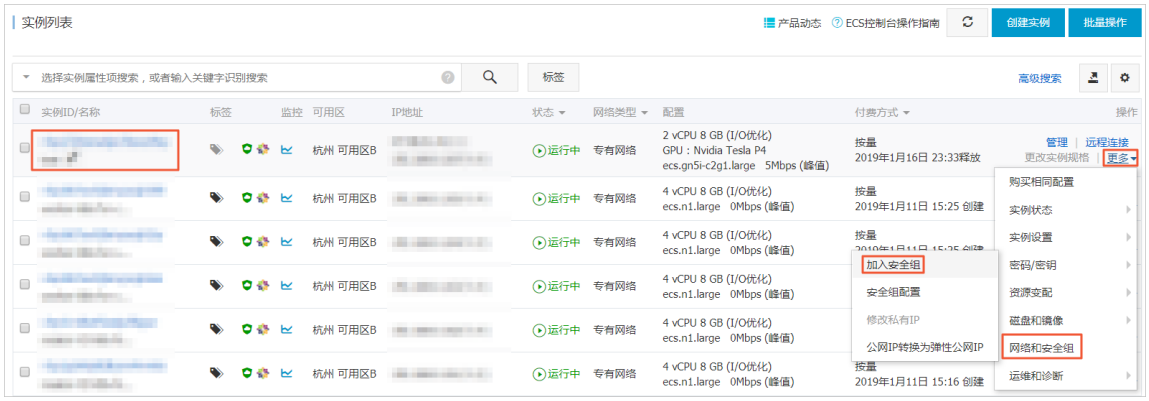
v. 在**专有网络详情**页面，单击**资源管理**页签**安全组**下的数字，跳转到云服务器ECS管理控制台的**安全组列表**页面查看安全组的详细信息。

vi. 在**安全组列表**页面，查看安全组的名称。

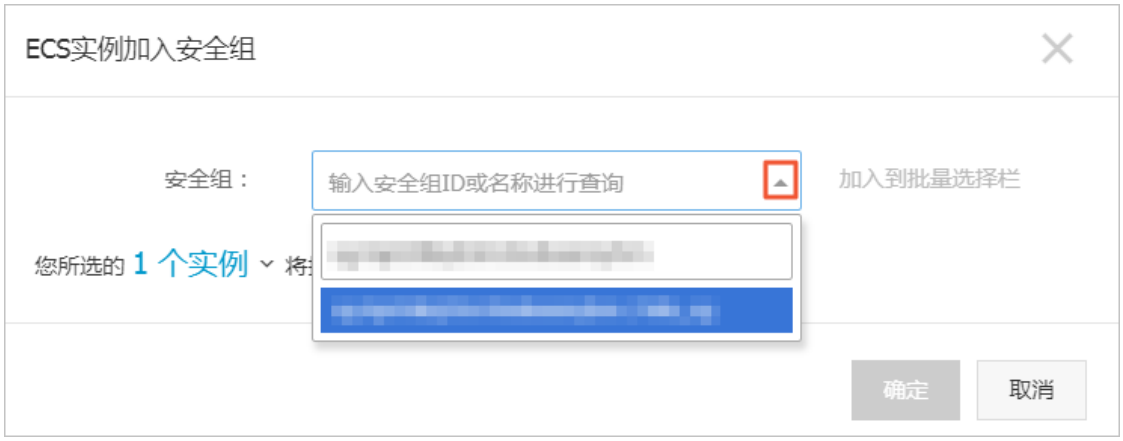


vii. 在云服务器ECS管理控制台，选择左侧导航栏**实例与镜像 > 实例**。

viii. 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。



ix. 单击**安全组**复选框右侧的下拉箭头，输入步骤6查询到的集群安全组名称。



x. 单击**确定**。

执行结果：

i. 在云服务器管理控制台左侧导航栏，选择**实例与镜像 > 实例**，在**实例列表**页面，单击目标实例的名称。

ii. 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。

集群审计功能是否可以取消或者在创建集群后再部署？

可以。具体操作，请参见使用集群审计功能。

Kubernetes专有版本集群怎么更换证书有效期，以及更换各个组件的证书？

- 集群临近过期前两个月左右您会收到站内和短信通知，收到通知后在控制台集群列表页面单击证书更新按钮即可。具体操作，请参见更新Kubernetes集群即将过期的证书。
- 如果Kubernetes集群证书已过期，具体操作，请参见更新Kubernetes集群已过期的证书。

Pod无法创建，报错详情：no providers available to validate pod request

- 如果您没有自定义PSP，出现该报错是因为您把默认PSP删除了，恢复使用默认的PSP规则即可。具体操作，请参见使用Pod安全策略。
- 如果您需要自定义PSP，具体操作，请参考使用PSP安全策略。

Secret在新建的Namespace下面无法使用

Secret是命名空间级别的，您需要在新建的Namespace下新建Secret。

无法挂载default-token

无法挂载default-token，具体报错信息如下：

```
Normal Scheduled 13m default-scheduler Successfully assigned dev/alibaba-demo-67fcdbfb8-zklnp to cn-hangzhou.10.7.3.16  Warning FailedMount 13m (x2 over 13m) kubelet, cn-hangzhou.10.7.3.16 MountVolume.SetUp failed for volume 'default-token-8twx9' : mount failed: exit status 1 Mounting command: systemd-run Mounting arguments: --description=Kubernetes transient mount for /var/lib/kubelet/pods/62d39b35-9a4d-11ea-9870-c24d56a0e904/volumes/kubernetes.io~secret/default-token-8twx9 --scope -- mount -t tmpfs tmpfs /var/lib/kubelet/pods/62d39b35-9a4d-11ea-9870-c24d56a0e904/volumes/kubernetes.io~secret/default-token-8twx9 Output: Failed to start transient scope unit: Argument list too long  Warning FailedCreatePodContainer 3m40s (x49 over 13m) kubelet, cn-hangzhou.10.7.3.16 unable to ensure pod container exists: failed to create container for [kubepods burstable pod62d39b35-9a4d-11ea-9870-c24d56a0e904] : Argument list too long
```

Systemd版本太老旧。

- 升级Systemd，具体操作，请参见systemd。
- 执行 `systemctl daemon-reload` 命令重启清零。更多信息，请参见systemd。

审计日志查询方法

RBAC相关变更操作的审计日志查询方法

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理信息页面，选择**集群资源**页签，单击日志服务Project右侧的链接。
5. 在**日志存储 > 日志库**页面，选择对应的 `audit-<cluster_id>` ，单击**查询分析**。



6. 单击右上角**15分钟（相对）**，选择需要查询的时间段。

> **说明** 时间段覆盖的范围是正常到出现问题时的这段时间，比如：3 天、7天或15天。

7. 在**查询/分析**文本框中输入以下SQL查询命令，然后单击**查询/分析**。

```
requestURI: "rbac.authorization.k8s.io" not (verb: get or verb: watch)
```

8. 单击  图标，在弹出的**日志下载**对话框中，选中**通过Cloud Shell下载**，单击**确定**。



ConfigMap相关变更操作的审计日志查询方法

在**查询/分析**文本框中输入以下SQL查询命令，然后单击**查询/分析**。更多操作，请参见审计日志查询方法。

```
requestURI: "configmaps" and <configmap_name> not (verb: get or verb: watch or verb: list)
```

> **说明** 上述查询命令在查询时，需要将*<configmap_name>*替换为实际的ConfigMap名称。

Deployment的Pod扩缩容相关操作的审计日志查询方法

在**查询/分析**文本框中输入以下SQL查询命令，然后单击**查询/分析**。更多操作，请参见审计日志查询方法。

```
requestURI: deployments and (verb: update or verb: patch) and replicas and deployments and <deployment_name
> not deployment-controller
```

> **说明** 上述查询命令在查询时，需要将*<deployment_name>*替换为实际的Deployment名称。

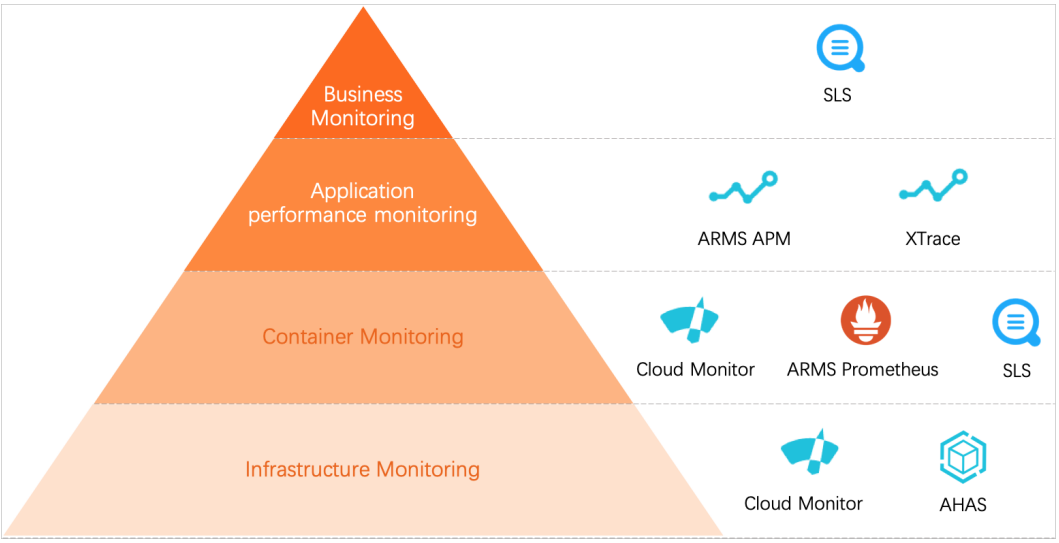
13.可观测性

13.1. 可观测性体系概述

可观测性指如何从外部输出推断及衡量系统内部状态。Kubernetes可观测性体系包含监控和日志两部分，监控可以帮助开发者查看系统的运行状态，而日志可以协助问题的排查和诊断。本文介绍阿里云容器服务ACK可观测性生态分层和各层的可观测能力，以帮助您更好地对容器服务可观测性生态有一个全面的认识。

容器服务可观测生态概述

从可观测性的角度，以ACK为基础的系统架构可以粗略分为4个层次。自下而上分别是：基础设施层、容器性能层、应用性能层、用户业务层。



以下分别是基础设施层、容器性能层、应用性能层、用户业务层的可观测性介绍。

基础设施层可观测性 | 容器性能层可观测性 | 应用性能层可观测性 | 用户业务层可观测性

相关文档

- 架构感知监控
- 基础资源监控
- 阿里云Prometheus监控
- 开源Prometheus监控
- 事件监控
- 应用性能监控
- 在ASM中实现分布式跟踪
- 快速上手链路追踪
- 通过日志服务采集Kubernetes容器日志

13.2. 日志管理

13.2.1. 日志概述

Kubernetes日志可以协助问题的排查和诊断。本文介绍阿里云容器服务ACK如何通过多种方式对应用进行日志管理。

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。

- 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。具体操作，请参见通过日志服务采集Kubernetes容器日志。
- 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。具体操作，请参见为Kubernetes和日志服务配置Log4JAppender。
- 支持收集托管集群控制平面组件日志，将日志从ACK控制层采集到您账号中的SLS日志服务的Log Project中。具体操作，请参见收集托管集群控制平面组件日志。
- 部署了CoreDNS作为集群内的DNS服务器，可以通过查看CoreDNS日志来分析CoreDNS解析慢、访问高危请求域名等问题。具体操作，请参见分析和监控CoreDNS日志。

13.2.2. 通过日志服务采集Kubernetes容器日志

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。

步骤一：启用日志服务组件Logtail

您可以在创建集群时选中**使用日志服务**，启用Logtail组件；也可以为已有集群启用Logtail组件。

创建集群时启用Logtail

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**创建集群**。

   以下仅介绍开启日志服务的关键步骤。关于创建集群的具体操作，请参见创建Kubernetes托管版集群。
4. 在**组件配置**配置项页中，选中**使用日志服务**，表示在新建的Kubernetes集群中安装日志插件。

   当选中**使用日志服务**后，会出现创建项目（Project）的提示。关于日志服务管理日志的组织结构，请参见项目（Project）。有以下两种创建Project方式。
   - 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是您新建的Kubernetes集群的唯一标识。

5. 配置完成后，单击右下角**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开启了Logtail的ACK集群。

为已有集群启用Logtail

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**，并在**日志与监控**区域找到**logtail-ds**。
5. 在**logtail-ds**组件右侧，单击**安装**。
6. 在**安装组件**对话框中单击**确认**。

如果您已安装旧版本的日志服务组件，可以在**logtail-ds**组件右侧，单击**升级**。

步骤二：创建应用时配置日志服务

您可以在创建应用的同时配置日志服务，从而对容器的日志进行采集。目前支持控制台向导和YAML模板两种方式创建应用。

通过控制台向导创建应用并配置日志服务

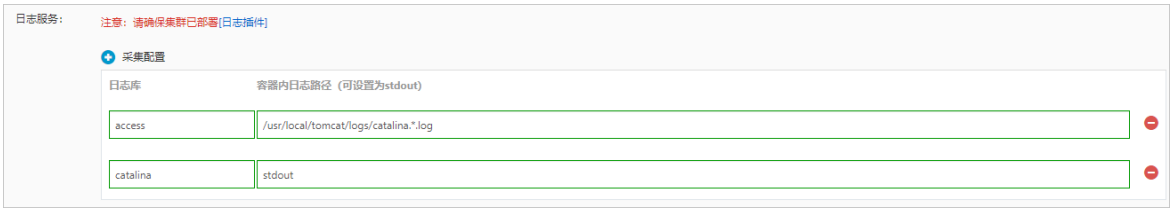
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面上方的**命名空间**下拉框中设置命名空间，然后单击页面右上角的**使用镜像创建**。
6. 在**应用基本信息**页签，设置**应用名称**、**副本数量**和**类型**，单击**下一步**，进入**容器配置**页面。

   以下仅介绍日志服务相关的配置，其他的应用配置，可参见创建无状态工作负载Deployment。
7. 进行**日志配置**。单击**+**图标创建新的采集配置，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该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。



8. 设置**自定义Tag**。

   单击**+**图标创建新的自定义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，例如版本号。



9. 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，可参见创建无状态工作负载Deployment。

通过YAML模板创建

1. 在集群管理页左侧导航栏中，单击**工作负载**。
2. 在**无状态**页面上方的**命名空间**下拉框中设置命名空间，然后单击页面右上角的**使用YAML创建资源**。
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`来为容器增加**采集配置**和**自定义Tag**，并根据采集配置，创建对应的volumeMounts和volumes。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volume mount ###########
  volumeMounts:
  - name: volumn-sls-mydemo
   mountPath: /var/log
 volumes:
 - name: volumn-sls-mydemo
  emptyDir: {}
 ################################
```

根据您的需求按照以下顺序进行配置。

> **说明** 如果您有更多的日志采集需求，请参见步骤三：配置环境变量的高级参数。

i. 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（_），可以使用短划线（-）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。

ii. 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volumeMounts。
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。

（可选）

步骤三：配置环境变量的高级参数

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。

> **注意** 通过容器环境变量配置采集日志的方式不适用边缘场景。

字段	说明	示例	注意事项
aliyun_logs_{key}	• 必选项。{key}只能包含小写字母、数字和-。 • 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，并参见通过DaemonSet-控制台方式采集Kubernetes文件、通过DaemonSet-控制台方式采集Kubernetes标准输出或通过DaemonSet-CRD方式采集容器日志进行配置。 • {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_logstore 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	-

字段	说明	示例	注意事项
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，指定日志的保存时间为永久保存。 • 当不存在该环境变量时，默认指定日志的保存时间为90天。	- name: aliyun_logs_catalina_ttl 3650	-
aliyun_logs_{key}_machinegroup	可选。值为应用的机器组。当不存在该环境变量时与安装Logtail的默认机器组一致。	- name: aliyun_logs_catalina_machinegroup my-machine-group	-

- **定制需求1：将多个应用数据采集到同一Logstore**
  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：
  i. 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。
  ii. 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
```

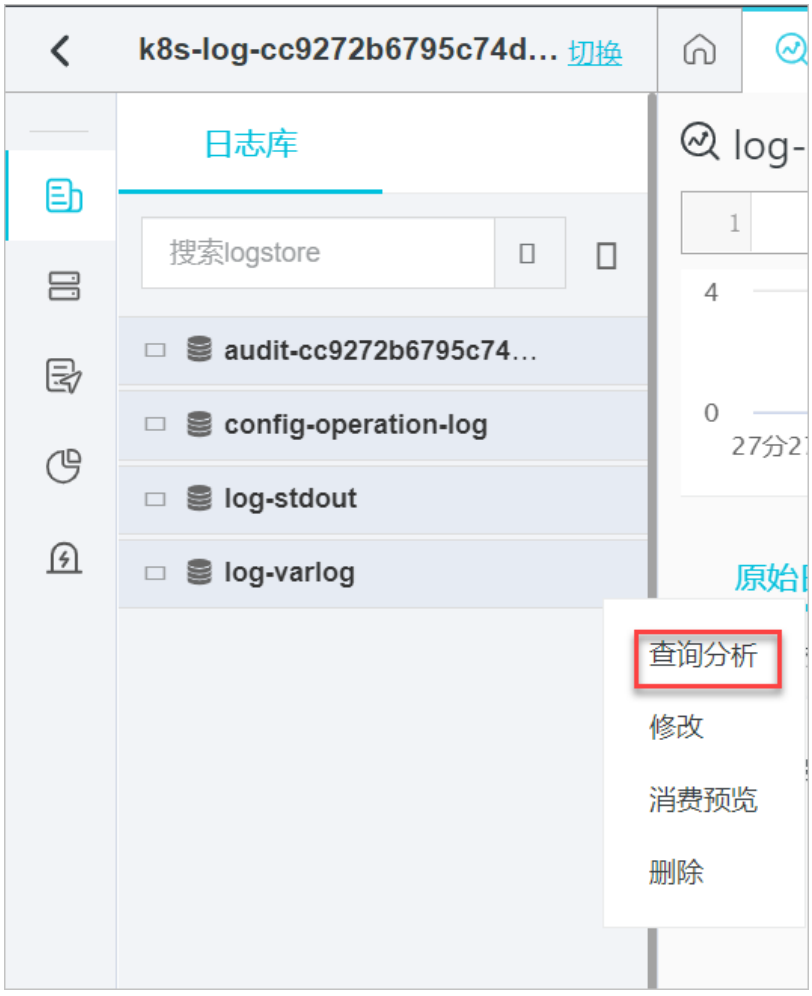
步骤四：查看日志

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，您可以在日志服务控制台查看容器日志。操作步骤如下：

1. 安装成功后，进入日志服务控制台。
2. 在进入控制台后，在**Project列表**区域选择Kubernetes集群对应的Project（默认为k8s-log-{Kubernetes集群ID}），进入**日志库**列表页签。
3. 在列表中找到相应的Logstore（采集配置中指定），将鼠标悬浮在相应的Logstore名称的右侧，单击

   图标，并单击**查询分析**。
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。



在为应用接入日志服务后，您也可以在ACK控制台查看容器的日志。操作如下：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 日志中心**。
5. 在**日志中心**页面，单击**应用日志**页签，设置检索条件后单击**选择日志库**即可查看应用容器的相关日志。

更多信息

1. 默认情况下，日志服务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，可以参见以下日志服务文档并前往日志服务控制台进行配置修改。
   - 通过DaemonSet-控制台方式采集Kubernetes文件
   - 通过DaemonSet-控制台方式采集Kubernetes标准输出
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，具体可参见通过DaemonSet-CRD方式采集容器日志。
3. 关于如何进行异常排查，请参见Logtail排查简介。

13.2.3. 为Kubernetes和日志服务配置Log4JAppender

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（杭州）。
- 启用AccessKey或RAM，确保有足够的访问权限。本例中使用AccessKey。

背景信息

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。

操作步骤

1. 在阿里云日志服务上配置Log4jAppender。
   i. 创建一个日志服务Project。

      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。具体操作，请参见创建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。

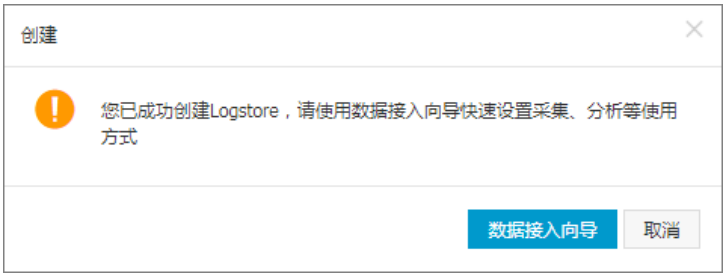

创建Project
* Project名称:
注释:
不支持<>'\"\\,最多包含64个字符
* 所属地域:
cn-qingdao-env17-d01
开通服务日志:
cn-qingdao-env17-d01
重要日志（计量，消费组延迟和Logtail心跳日志等，免费）
开通服务日志会在您选择的存储位置创建对应的Logstore和仪表盘。
确定
取消

ii. 为k8s-log4j的Project创建一个Logstore。

本示例创建名为k8s-logstore的日志库。具体操作，请参见创建Logstore。



iii. 在k8s-logstore创建完成后，页面会提示您创建数据接入向导。



iv. 选择**自定义代码**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。

本示例使用demo-deployment和demo-Service示例YAML文件进行演示。

i. 连接到您的Kubernetes集群。

具体操作参见SSH访问Kubernetes集群或通过kubectl工具连接集群。

ii. 获取*demo-deployment.yaml*文件并配置环境变量 `JAVA_OPTS` 。

*demo-deployment.yaml*文件的示例编排如下。

```
apiVersion: apps/v1
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
```

> **说明** 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。

iii. 在命令行中执行以下命令，创建deployment。

```
kubectl create -f demo-deployment.yaml
```

iv. 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. 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。
命令如下：

```
kubectl get svc
```

结果如下：

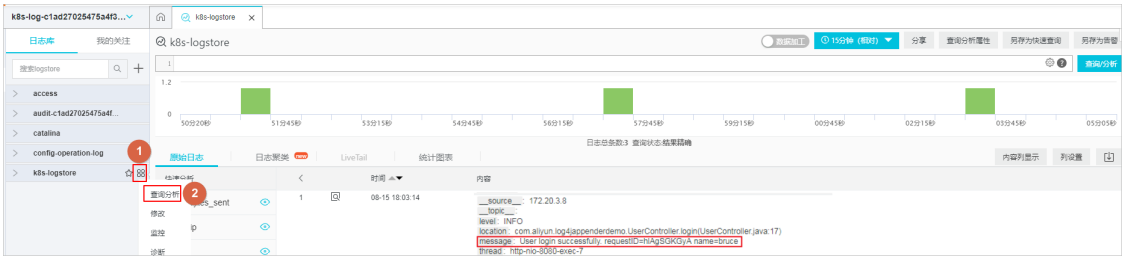
```
NAME                TYPE       CLUSTER-IP   EXTERNAL-IP    PORT(S)      AGE
log4j-appender-demo-spring-boot-svc  LoadBalancer  172.21.XX.XX  120.55.XXX.XXX  8080:30398/TCP  1h
```

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
```

4. 在阿里云日志服务中查看日志。
   i. 在**Project列表**中，单击目标Project，进入对应的Project详情页面。
   ii. 在对应的日志库k8s-logstore右侧的图标，选择**查询分析**，查看Kubernetes集群输出的日志，如下所示。



13.2.4. 收集托管集群控制平面组件日志

阿里云容器服务Kubernetes版ACK（Alibaba Cloud Container Service for Kubernetes）支持收集托管集群控制平面组件日志，将日志从ACK控制层采集到您账号中的SLS日志服务的Log Project中。本文主要介绍如何开启收集托管集群控制平面组件日志功能和查看集群控制平面组件日志。

前提条件

您账号下SLS日志服务中的日志库配额充足。

> **说明** 一个账号默认情况下日志库配额为50。

背景信息

相关控制平面control plane组件日志可以帮助您更安全有效地运维集群，您可以在创建ACK标准托管集群或ACK Pro托管集群的时候通过选中**收集控制平面组件日志**的选项开启相关组件的日志收集能力，日志将以日志流的形式发送到您账号下指定的SLS日志服务的Log Project中，且将以SLS日志服务统一的按量付费方式进行计费。更多信息，请参见按量付费。

开启收集控制平面组件日志

在创建集群时的**组件配置**的配置向导页面，在**控制平面组件日志**区域选中**开启**选项。关于如何创建集群的详细信息，请参见创建Kubernetes托管版集群或创建ACK Pro版集群。

> **说明**
> - 创建ACK Pro托管集群时会默认选中**开启**选项，ACK标准托管版集群默认不选中。
> - **控制平面组件日志**支持选择使用已有Project。



查看集群控制平面组件日志

当托管集群创建完成后，有以下两种方式查看集群控制平面组件日志。

方式一：通过日志服务控制台查看目标集群控制平面组件日志。

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标集群对应的日志服务Project名称。
3. 在**日志存储**页面左侧的**日志库**列表选择目标控制平面组件的日志库，当前支持**kube-apiserver**，**kube-scheduler**和**kube-controller-manager**三种控制平面组件日志查询方式。具体操作，请参见SLS官方文档。

方式二：通过容器服务控制台查看目标集群控制平面组件日志。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 查看三种控制平面组件。
   - 通过**集群信息**入口查看三种控制平面组件。
     a. 在集群信息管理页面单击**集群资源**页签，在列表中单击控制平面组件日志对应的Project链接。
     b. 在**日志存储**页面左侧的**日志库**列表选择目标控制平面组件的日志库，当前支持**kube-apiserver**，**kube-scheduler**和**kube-controller-manager**三种控制平面组件的日志查询。具体操作，请参见SLS官方文档。
   - 通过**运维管理**入口查看三种控制平面组件。
     a. 在集群管理左侧导航栏，选择**运维管理 > 日志中心**。
     b. 单击**控制平面组件日志**页签，然后选择目标的组件查看相应的组件日志信息。

控制平面组件Logstore说明

当前ACK支持收集以下三种类型的控制平面组件日志，每种日志服务Logstore对应一个Kubernetes控制层面组件。关于这些组件的更多信息，请参见Kubernetes组件。

组件	Logstore	说明
kube-apiserver	apiserver	kube-apiserver组件是暴露Kubernetes API接口的控制层面的组件。更多信息，请参见kube-apiserver。
kube-controller-manager	kcm	kube-controller-manager组件是Kubernetes集群内部的管理控制中心，内嵌了Kubernetes发布版本中核心的控制链路。更多信息，请参见kube-controller-manager。
kube-scheduler	scheduler	kube-scheduler组件是Kubernetes集群的默认调度器。更多信息，请参见kube-scheduler。

相关文档

- 按量付费
- 查询概述
- 创建Kubernetes托管版集群
- 创建ACK Pro版集群

13.2.5. 分析和监控CoreDNS日志

阿里云容器服务ACK部署了CoreDNS作为集群内的DNS服务器，您可以通过查看CoreDNS日志来分析CoreDNS解析慢、访问高危请求域名等问题。本文介绍如何开启CoreDNS日志分析与监控能力。

前提条件

- 已安装日志组件logtail-ds。
  在创建集群时，ACK默认选中日志组件logtail-ds。如果您未安装日志组件logtail-ds，关于如何手动安装的操作，请参见通过日志服务采集Kubernetes容器日志。
- 请确保日志应用alibaba-log-controller版本不低于0.2.0.0-76648ee-aliyun。



如果alibaba-log-controller版本低，可直接升级logtail-ds组件。关于如何升级组件，请参见管理组件。

步骤一：开启CoreDNS日志服务

> **说明** 开启CoreDNS日志会消耗额外的CPU（约10%，与实际请求有关）及网络流量。如果当前已经部署的CoreDNS副本CPU使用量较高，您可以对CoreDNS进行扩容。关于如何扩容CoreDNS副本数，请参见手动伸缩容器组。

在命名空间kube-system下，ACK集群有一个**coredns**配置项。您可以通过在**coredns**配置项中的Corefile字段中加上log插件，开启CoreDNS每次域名解析的日志。关于如何修改配置项的具体步骤，请参见修改配置项。

默认日志格式的**coredns**的配置如下。

```
Corefile: |
 .:53 {
    errors
    log // 此处添加Log插件。
    health {
      lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
      pods insecure
      upstream
      fallthrough in-addr.arpa ip6.arpa
      ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
 }
 // 如果有其它的域，且该域容器需要被采集日志，也需要增加Log插件，格式一致。
 demo.com:53 {
    ...
    log //此处添加Log插件。
 }
```

步骤二：配置CoreDNS日志采集

日志采集配置针对ACK进行了自定义资源CRD扩展，您可以直接部署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。关于如何创建自定义资源AliyunLogConfig的具体步骤，请参见管理自定义资源。

创建自定义资源AliyunLogConfig使用的YAML配置文件如下。

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
 #    Your config name, must be unique in you k8s cluster.
 name: k8s-coredns-log
 namespace: kube-system
spec:
 # logstore name to upload log
 logstore: coredns-log
 # logtail config detail
 productCode: k8s-coredns
 logtailConfig:
  inputType: plugin
  # logtail config name, should be same with [metadata.name]
  configName: k8s-coredns-log
  inputDetail:
    plugin:
      inputs:
      - type: service_docker_stdout
        detail:
          IncludeLabel:
            io.kubernetes.container.name: coredns
          Stderr: true
```

```
      Stdout: true
    processors:
    - type: processor_regex
      detail:
        KeepSource: false
        KeepSourceIfParseError: true
        Keys:
        - level
        - remote
        - port
        - id
        - type
        - class
        - name
        - proto
        - size
        - do
        - bufsize
        - rcode
        - rflags
        - rsize
        - duration
        NoKeyError: true
        NoMatchError: false
        FullMatch: false
        Regex: \[([^]]+)]\s([^:]+):(\S+)\s+-\s+(\S+)\s+"(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+([^"]+)"\s+(\S+)\s+(\S+)\s+(\S+)\s+([\d\.]+).*
        SourceKey: content
    - type: processor_regex
      detail:
        KeepSource: false
        KeepSourceIfParseError: true
        Keys:
        - error
        - rcode
        - name
        - type
        - errorMsg
        NoKeyError: false
        NoMatchError: false
        FullMatch: false
        Regex: \[ERROR]\s+(plugin/errors):\s+(\S)+\s+(\S+)\s+([^:]*):\s+(.*)
        SourceKey: content
```

> **注意**
>
> - 请确保日志组件alibaba-log-controller版本不低于0.2.0.0-76648ee-aliyun。如果在更新alibaba-log-controller版本前，您已经部署了该CRD配置，请删除该配置并重新部署。
> - 上述配置只针对默认CoreDNS中的日志格式生效。如果您自定义了CoreDNS的日志格式，请修改上述CRD配置中的正则表达式Regex部分。
>   关于自定义CoreDNS的日志格式的更多信息，请参见log。
>   关于更详细的日志采集流程及采集配置，请参见通过DaemonSet-CRD方式采集容器日志。

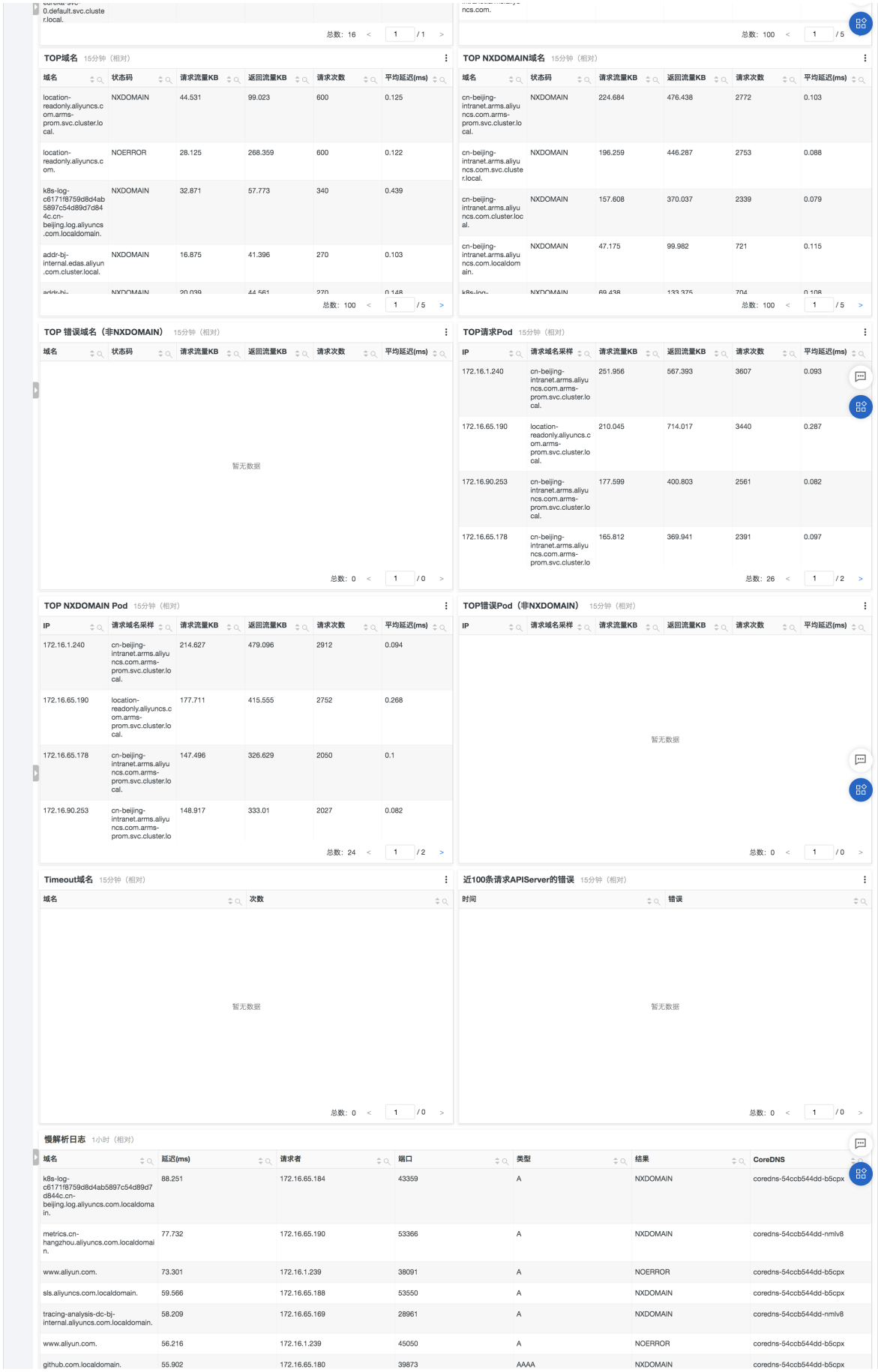
关于如何配置存储、网络、弹性伸缩等资源的AliyunLogConfig的CRD，请参见步骤一：创建系统插件的日志配置文件。

步骤三：查看CoreDNS日志采集

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面，单击**集群资源**页签，然后单击**日志服务Project**右侧的日志服务Project名称链接。
5. 在日志库页面左侧导航栏，单击图标，然后单击**仪表盘**，在仪表盘列表中找到并单击**Kubernetes CoreDNS日志分析**。

   在**Kubernetes CoreDNS日志分析**页面，您可以查看CoreDNS请求数、请求成功率和请求延迟等聚合信息，同时提供了各类Top域名请求列表、错误域名列表、慢解析域名列表和高危请求域名访问分析。



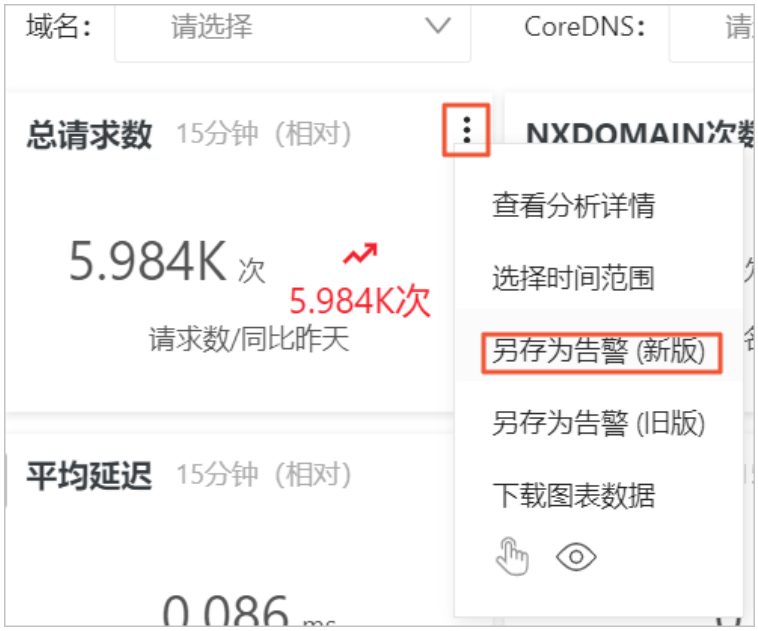

0.default.svc.cluster.local.
总数：16 < 1 / 1 >
ncs.com.
总数：100 < 1 / 5
TOP域名 15分钟（相对）
域名 状态码 请求流量KB 返回流量KB 请求次数 平均延迟(ms)
location-readonly.aliyuncs.com.arms-prom.svc.cluster.local. NXDOMAIN 44.531 99.023 600 0.125
location-readonly.aliyuncs.com. NOERROR 28.125 268.359 600 0.122
k8s-log-c617118759d8d4ab5897c54d89d7d844c.cn-beijing.log.aliyuncs.com.localdomain. NXDOMAIN 32.871 57.773 340 0.439
addr-bj-internal.edas.aliyun.com.cluster.local. NXDOMAIN 16.875 41.396 270 0.103
总数：100 < 1 / 5 >
TOP NXDOMAIN域名 15分钟（相对）
域名 状态码 请求流量KB 返回流量KB 请求次数 平均延迟(ms)
cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.local. NXDOMAIN 224.684 476.438 2772 0.103
cn-beijing-intranet.arms.aliyuncs.com.svc.cluster.local. NXDOMAIN 196.259 446.287 2753 0.088
cn-beijing-intranet.arms.aliyuncs.com.cluster.local. NXDOMAIN 157.608 370.037 2339 0.079
cn-beijing-intranet.arms.aliyuncs.com.localdomain. NXDOMAIN 47.175 99.982 721 0.115
总数：100 < 1 / 5 >
TOP 错误域名（非NXDOMAIN） 15分钟（相对）
域名 状态码 请求流量KB 返回流量KB 请求次数 平均延迟(ms)
暂无数据
总数：0 < 1 / 0 >
TOP请求Pod 15分钟（相对）
IP 请求域名采样 请求流量KB 返回流量KB 请求次数 平均延迟(ms)
172.16.1.240 cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.local. 251.956 567.393 3607 0.093
172.16.65.190 location-readonly.aliyuncs.com.arms-prom.svc.cluster.local. 210.045 714.017 3440 0.287
172.16.90.253 cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.local. 177.599 400.803 2561 0.082
172.16.65.178 cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.lo 165.812 369.941 2391 0.097
总数：26 < 1 / 2 >
TOP NXDOMAIN Pod 15分钟（相对）
IP 请求域名采样 请求流量KB 返回流量KB 请求次数 平均延迟(ms)
172.16.1.240 cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.local. 214.627 479.096 2912 0.094
172.16.65.190 location-readonly.aliyuncs.com.arms-prom.svc.cluster.local. 177.711 415.555 2752 0.268
172.16.65.178 cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.local. 147.496 326.629 2050 0.1
172.16.90.253 cn-beijing-intranet.arms.aliyuncs.com.arms-prom.svc.cluster.lo 148.917 333.01 2027 0.082
总数：24 < 1 / 2 >
TOP错误Pod（非NXDOMAIN） 15分钟（相对）
IP 请求域名采样 请求流量KB 返回流量KB 请求次数 平均延迟(ms)
暂无数据
总数：0 < 1 / 0 >
Timeout域名 15分钟（相对）
域名 次数
暂无数据
总数：0 < 1 / 0 >
近100条请求APIServer的错误 15分钟（相对）
时间 错误
暂无数据
总数：0 < 1 / 0 >
慢解析日志 1小时（相对）
域名 延迟(ms) 请求者 端口 类型 结果 CoreDNS
k8s-log-c617118759d8d4ab5897c54d89d7d844c.cn-beijing.log.aliyuncs.com.localdomain. 88.251 172.16.65.184 43359 A NXDOMAIN coredns-54ccb544dd-b5cpx
metrics.cn-hangzhou.aliyuncs.com.localdomain. 77.732 172.16.65.190 53366 A NXDOMAIN coredns-54ccb544dd-nmlv8
www.aliyun.com. 73.301 172.16.1.239 38091 A NOERROR coredns-54ccb544dd-b5cpx
sls.aliyuncs.com.localdomain. 59.566 172.16.65.188 53550 A NXDOMAIN coredns-54ccb544dd-b5cpx
tracing-analysis-dc-bj-internal.aliyuncs.com.localdomain. 58.209 172.16.65.169 28961 A NXDOMAIN coredns-54ccb544dd-nmlv8
www.aliyun.com. 56.216 172.16.1.239 45050 A NOERROR coredns-54ccb544dd-b5cpx
github.com.localdomain. 55.902 172.16.65.180 39873 AAAA NXDOMAIN coredns-54ccb544dd-b5cpx



步骤四：创建CoreDNS日志告警

在Kubernetes CoreDNS日志分析仪表盘页面中，您可以根据仪表盘中单个图表进行告警设置。

在目标图表卡片右上方，选择 ⋮ > **另存为告警（新版）**。关于告警配置的参数设置，请参见创建日志告警监控规则。



创建完成告警规则后，您可以查看、修改告警规则，关闭告警规则等。更多信息，请参见告警监控规则相关操作。

相关文档

- 基于DNS的服务发现概述
- ACK集群内DNS原理和配置说明

13.2.6. 收集系统插件日志

系统插件（Addon）使用Kubernetes资源（DaemonSet、Deployment等）实现集群的存储、网络等功能，这些资源部署于kube-system命名空间之下。本文介绍如何配置收集存储、网络、弹性伸缩等主要系统插件的日志功能。

前提条件

- 已安装日志组件logtail-ds。
  在创建集群时，ACK默认选中日志组件logtail-ds。如果您未安装日志组件logtail-ds，关于如何手动安装的操作，请参见通过日志服务采集Kubernetes容器日志。
- 请确保日志应用alibaba-log-controller版本不低于0.2.0.0-76648ee-aliyun。



如果alibaba-log-controller版本低，可直接升级logtail-ds组件。关于如何升级组件，请参见管理组件。

步骤一：创建系统插件的日志配置文件

日志采集配置针对ACK进行了自定义资源CRD扩展，您可以直接部署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。关于如何创建自定义资源，请参见管理自定义资源。

根据以下示例创建存储、Terway网络、Flannel网络及弹性伸缩的AliyunLogConfig配置文件。

- 存储CS插件的日志配置示例：

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
 #您的配置名称在K8s集群中必须是唯一的。
 name: k8s-csi-log
 namespace: kube-system
spec:
 #用于上传日志的logstore名称。
 logstore: k8s-csi-log
 #logtail配置详细信息。
 logtailConfig:
  #Docker标准输出的输入类型为“plugin”。
  inputType: plugin
  #logtail配置名称，应与[metadata.name]相同。
  configName: k8s-csi-log
  inputDetail:
   plugin:
    inputs:
     -
      #输入类型。
      type: service_docker_stdout
      detail:
       #收集stdout和stderr。
       Stdout: true
       Stderr: true
       K8sNamespaceRegex: "^kube-system$"
       K8sPodRegex: "^csi.*$"
```

- Terway网络插件的日志配置示例：

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
 #您的配置名称在K8s集群中必须是唯一的。
 name: k8s-terway-log
 namespace: kube-system
spec:
 #用于上传日志的logstore名称。
 logstore: k8s-terway-log
 #logtail配置详细信息。
 logtailConfig:
 #Docker标准输出的输入类型为“plugin”。
  inputType: plugin
  #logtail配置名称，应与[metadata.name]相同。
  configName: k8s-terway-log
  inputDetail:
   plugin:
    inputs:
     -
      #输入类型。
      type: service_docker_stdout
      detail:
       #收集stdout和stderr。
       Stdout: true
       Stderr: true
       K8sNamespaceRegex: "^kube-system$"
       K8sPodRegex: "^terway-eniip.*$"
```

- Flannel网络插件的日志配置示例：

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
 #您的配置名称在K8s集群中必须是唯一的。
 name: k8s-flannel-log
 namespace: kube-system
spec:
 #用于上传日志的logstore名称。
 logstore: k8s-flannel-log
 #logtail配置详细信息。
 logtailConfig:
  #Docker标准输出的输入类型为“plugin”。
  inputType: plugin
  #logtail配置名称，应与[metadata.name]相同。
  configName: k8s-flannel-log
  inputDetail:
   plugin:
    inputs:
     -
      #输入类型。
      type: service_docker_stdout
      detail:
       #收集stdout和stderr。
       Stdout: true
       Stderr: true
       K8sNamespaceRegex: "^kube-system$"
       K8sPodRegex: "^kube-flannel-ds.*$"
```

- 弹性伸缩插件的日志配置示例：

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
 #您的配置名称在K8s集群中必须是唯一的。
 name: k8s-cluster-autoscaler-log
 namespace: kube-system
spec:
 #用于上传日志的logstore名称。
 logstore: k8s-cluster-autoscaler-log
 #logtail配置详细信息。
 logtailConfig:
  #Docker标准输出的输入类型为“plugin”。
  inputType: plugin
  #logtail配置名称，应与[metadata.name]相同。
  configName: k8s-cluster-autoscaler-log
  inputDetail:
   plugin:
    inputs:
     -
      #输入类型。
      type: service_docker_stdout
      detail:
       #收集stdout和stderr。
       Stdout: true
       Stderr: true
       K8sNamespaceRegex: "^kube-system$"
       K8sPodRegex: "^cluster-autoscaler.*$"
```

步骤二：查看日志

关于如何查看日志，请参见步骤四：查看日志。

13.3. 监控管理

13.3.1. 基础资源监控

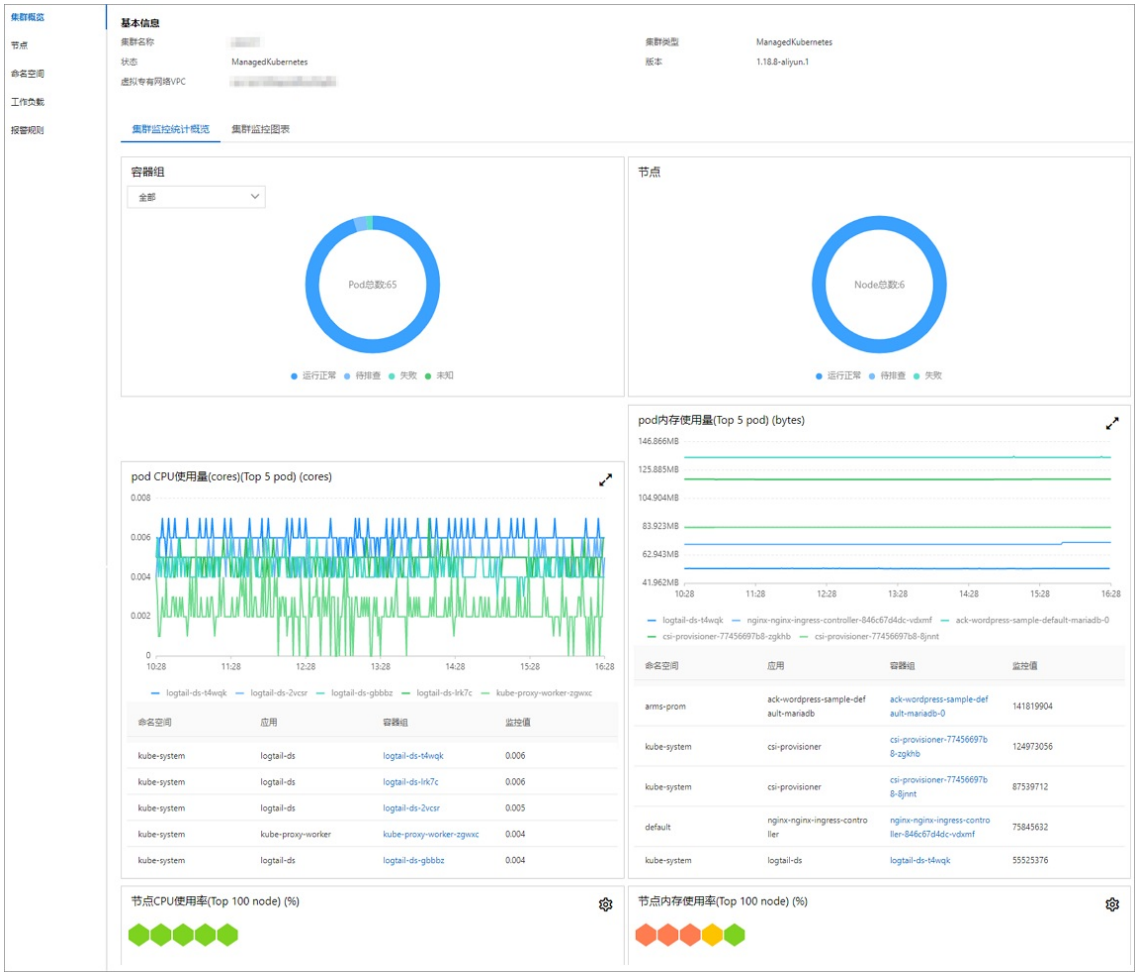
资源监控是Kubernetes中最常见的监控方式，通过资源监控可以快速查看负载的CPU、内存、网络等指标的使用率。在阿里云容器服务中，资源监控已经与云监控互通，新建的集群默认安装与集成云监控。本文介绍如何在控制台查看基础资源的监控及如何配置报警。

前提条件

若需要使用最新版本的云监控，容器服务Kubernetes版集群的metrics-server组件必须升级到V0.3.8.5及以上版本。具体操作，请参见升级Kubernetes集群的metrics-server组件。

功能特性

- 提供全集群视角指标，洞悉集群概况。

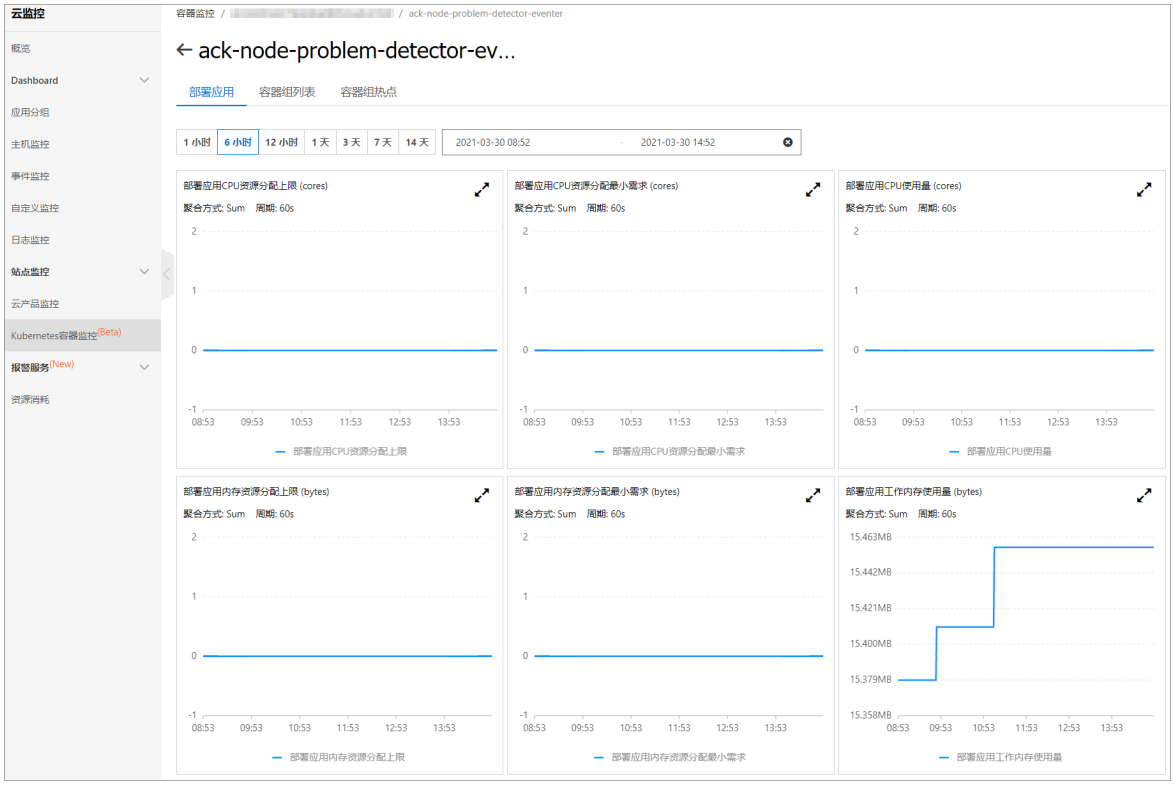


- 更专业的监控与报警能力。
  老版云监控容器监控功能全面升级，提供更加专业的容器场景基础监控能力。从集群命名空间、节点、工作负载及Pod等原生Kubernetes概念视角提供关键指标监控能力。报警功能全面升级，支持Kubernetes场景不同视角下配置对应报警规则。
- 更合适的容器场景指标。
  在宿主机基础设施层、容器PaaS层及Kubernetes调度层不同场景下使用最合适的指标。例如，容器中影响Kubernetes调度的内存指标，会使用容器工作内存的专用指标，与宿主机的内存Usage区分。

进入资源监控

方式一：从容器服务控制台进入资源监控

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面，单击目标应用右侧**操作**列下的**监控**，进入云监控的**容器监控**页面。



6. 您可以单击**部署应用**、**容器组列表**、**容器组热点**页签查看监控数据。

方式二：从云监控控制台进入资源监控

1.
2. 在控制台左侧导航栏中，单击**Kubernetes容器监控**。
3. 在**容器监控**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**查看详情**。

> **说明** 首次访问时，页面会出现授权提示，单击**授权**后才可进入目标集群页面。

4. 在目标集群页面，可从**集群概览**、**节点**、**命名空间**及**工作负载**中目标应用等视角查看监控数据。



配置指标报警的场景

场景	场景描述	指标报警配置
集群水位健康状况，以集群或节点配置水位报警。	当集群或集群节点发生水位等指标异常时，第一时间上报集群资源异常，避免影响业务。此场景推荐以集群或集群节点的方式配置水位报警规则。	配置报警规则时，选择**资源范围**为**集群**或**节点**，配置以整个集群、集群任一节点出现指标异常时的报警规则。选择节点时配置**全部**节点，则当该集群的任一节点出现规则描述中的指标异常时，则触发报警。
异常容器水位状况，以对应集群下的任一Pod统一报警。	当集群发生资源水位异常情况时，通常需要分解问题，找到具体Pod。此场景推荐以该集群中的任一Pod的方式配置水位报警规则。	配置报警规则时，选择**资源范围**为**容器组Pod**，选择命名空间及容器组Pod为**全部**，则当该集群的任一Pod出现规则描述中的指标异常时，则触发报警。
按命名空间多租户使用集群场景，针对对应集群下的指定命名空间的Pod设置报警。	通常一个集群会被多个应用共享，通过命名空间来拆分应用是一种Kubernetes的常用应用多租户方式。当应用所在的命名空间发生水位异常时，您可通过报警第一时间感知。此场景推荐以指定命名空间下任一Pod的方式配置水位报警规则。	配置报警规则时，选择**资源范围**为**容器组Pod**，选择**名字空间**为对应应用所在命名空间，选择**容器组Pod**为**全部**，则当该命名空间下的任一Pod出现规则描述中的指标异常时，则触发报警。
应用指标水位状况预警，针对对应集群下指定命名空间的指定应用（工作负载）的所属Pod设置报警。	通常一个集群会被多个应用共享，通过工作负载来拆分应用是一种Kubernetes的常用应用多租户方式，例如一个应用对应一个Deployment。当应用所对应的Deployment发生水位异常时，您可通过报警第一时间感知。此场景推荐以指定工作负载下任一Pod的方式配置水位报警规则。	配置报警规则时，选择**资源范围**为**容器组Pod**，选择**名字空间**为对应应用所在命名空间，选择应用的工作负载类型。目前支持的Kubernetes工作负载类型有：无状态（Deployment）、有状态（StatefulSet）、守护进程集（DaemonSet）、任务（Job）、定时任务（CronJob）。选择**容器组（Pod）**为任一Pod，则当该工作负载下的任一Pod出现规则描述中的指标异常时，则触发报警。

配置指标报警规则

步骤一：创建报警联系人并关联报警联系组

1. 登录云监控控制台。
2. 在控制台左侧导航栏中，选择**报警服务 > 报警联系人**。
3. 创建报警联系人，并添加到对应的报警联系组。

   具体操作，请参见创建报警联系人或报警联系组。

步骤二：创建指标报警规则

1. 登录云监控控制台。
2. 在控制台左侧导航栏中，单击**Kubernetes容器监控**。
3. 在**容器监控**页面，单击目标集群对应**操作**列的**报警规则**。
4. 在**报警规则**页面，单击**创建报警规则**。
5. 在**创建报警规则**面板，设置报警规则相关参数。

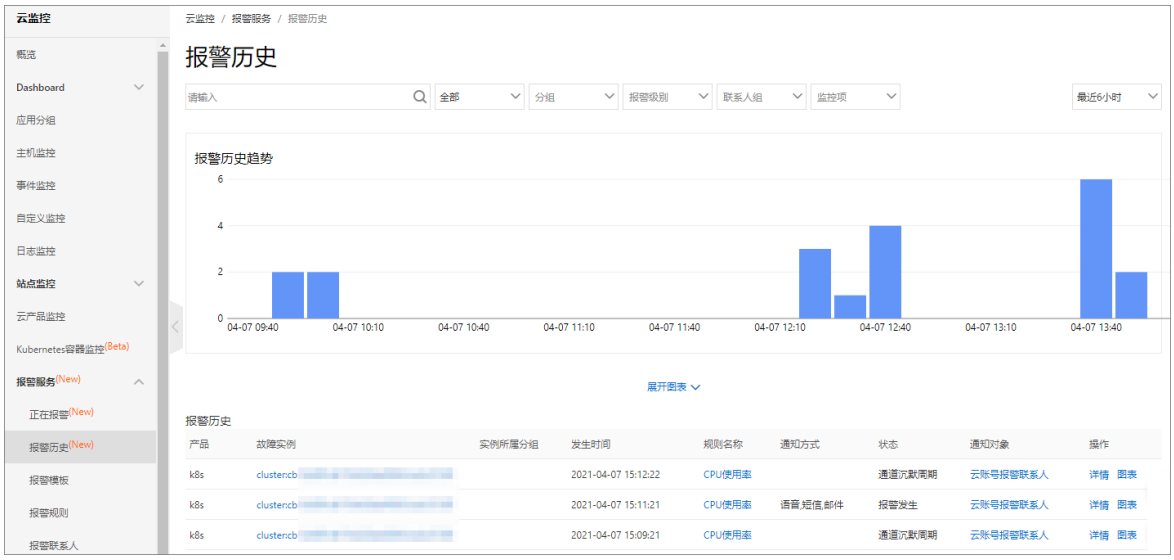
参数	描述
资源范围	报警规则的作用范围。取值： ○ **集群**：报警规则作用于目标集群。 ○ **节点**：报警规则作用于目标集群的全部节点或指定节点。 ○ **容器组（Pod）**：报警规则作用于目标集群的指定命名空间下应用中的全部容器组或指定容器组。选择**全部**，则表示任意该类型的实例触发了规则描述中的阈值规则，则报警。

参数	描述
规则描述	报警规则的主体。 配置具体报警指标、阈值及报警级别。关于容器组指标的监控项，请参见容器服务Kubernetes版（新版）。 当监控数据满足指定条件时，触发报警规则。
通道沉默周期	报警发生后如果未恢复正常，间隔多久重复发送一次报警通知。
生效时间	报警规则的生效时间。报警规则只在生效时间内才会检查监控数据是否需要报警。
报警回调	云监控会将报警信息通过POST请求推送到该地址，目前仅支持HTTP协议。 **说明** 建议您填写公网可访问的URL。
报警联系组	用于接收报警通知的联系人组。

6. 单击**确定**，完成报警规则配置。
   在**报警规则**页面，可以看到已创建的报警规则。

结果验证

1. 在控制台左侧导航栏中，选择**报警服务 > 报警历史**。
2. 在**报警历史**页面可查看报警历史趋势及报警历史。



旧版资源监控

若容器服务Kubernetes版集群的metrics-server组件未升级到V0.3.8.5及以上版本，可按以下操作进入旧版资源监控页面。

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 选择所需的Deployment，单击右侧的**监控**，进入到云监控的相应的**监控视图**页面。
5. 您可以单击**部署应用**、**容器组列表**、**容器组热点**页签查看监控数据。
6. （可选）如需设置告警，您可以在左侧导航栏，选择**报警服务 > 报警规则**。
   分组级别的指标以group开头，实例级别的指标以pod开头。更多信息，请参见管理报警规则。

13.3.2. 应用性能监控

对于部署在容器服务Kubernetes版中的Java及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

- 创建Kubernetes集群
- 开通和升级ARMS

> **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应用监控概述。

安装ARMS应用监控组件

1. 登录容器服务管理控制台。
2. 在左侧导航栏选择**市场 > 应用目录**，在**阿里云应用**页签下搜索并单击**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，并单击**创建**。

为容器服务Kubernetes版授权

执行以下操作步骤为容器服务Kubernetes版授予ARMS资源的访问权限。

1. 使用阿里云账号登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面中，单击**集群资源**页签中的Worker RAM角色**KubernetesWorkerRole-******。
5. 在**RAM访问控制**控制台的**RAM角色管理**页面上，单击**权限管理**页签上的权限策略名称**k8sWorkerRolePolicy-******。
6. 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**确认**。

```
{
  "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。

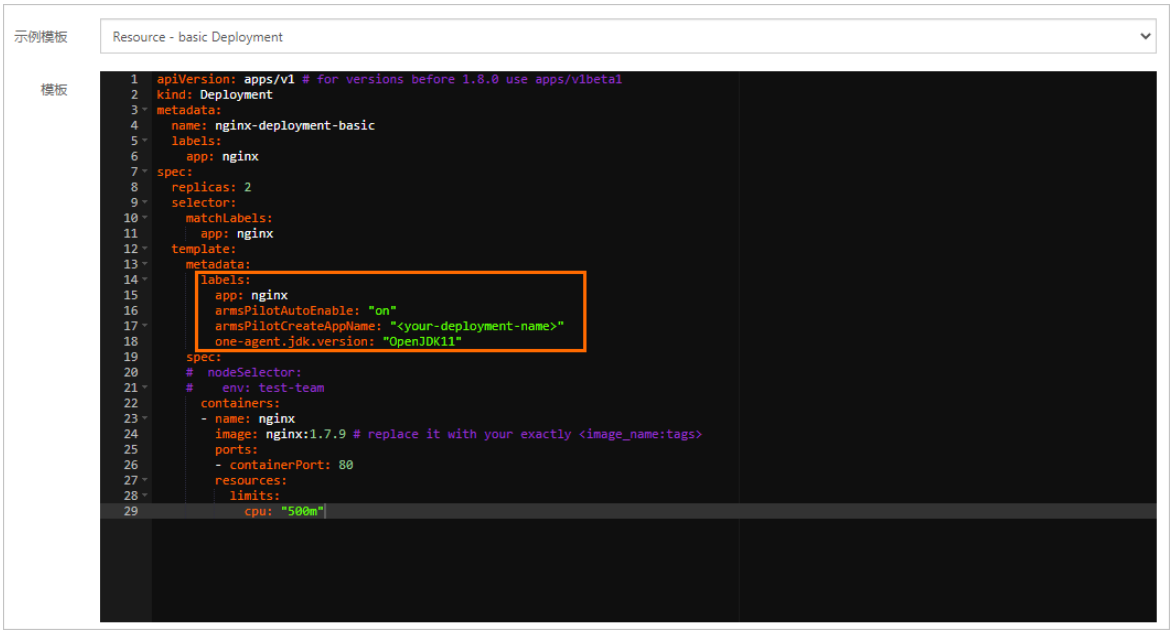
如需在创建新应用的同时开启ARMS应用监控，请按以下步骤操作：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

5. 在**无状态**页面右上角单击**使用YAML创建资源**。
6. 在**使用YAML创建资源**页面上选择**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：

展开查看完整示例YAML文件（Java）

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. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面右上角单击**使用YAML创建**。
6. 在**使用模板创建**页面上选择**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
```

7. （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.in*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。

8. 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.in*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：

展开查看完整示例YAML文件（PHP） ▼

如需为现有应用开启ARMS应用监控，请按以下步骤操作：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 选择**命名空间**，在目标应用右侧**操作**列中选择**更多 > 查看Yaml**。
6. 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
```

7. 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
```

执行结果

在**无状态**或**有状态**页面，目标应用的**操作**列将出现**ARMS控制台**操作。

> **说明** 若**操作**列没有出现**ARMS控制台**操作，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。



13.3.3. 架构感知监控

AHAS提供针对容器服务Kubernetes环境的可视化展示能力。本文介绍应用高可用服务AHAS（Application High Availability Service）架构感知。

前提条件

- 已创建容器服务Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已开通AHAS。具体操作，请参见开通AHAS。

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，及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，可以快速低成本地提升应用可用性。更多信息，请参见什么是应用高可用服务AHAS。

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。

   i. 登录容器服务管理控制台。

   ii. 在左侧导航栏选择**市场 > 应用目录**，在右侧选中 **ack-ahas-pilot**。

   iii. 在**应用目录 - ack-ahas-pilot** 页面右侧的**创建**区域，选择集群名称。

   iv. 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。 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

   v. 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。
   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，请检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：

i. 登录容器服务管理控制台。
ii. 在控制台左侧导航栏中，单击**集群**。
iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
iv. 在左侧导航栏，选择**应用 > Helm**。
v. 找到发布名称为**ahas**的Helm应用，单击**操作**列的**详情**。
vi. 单击**参数**页签，查看地域参数 `region_id` 的值。

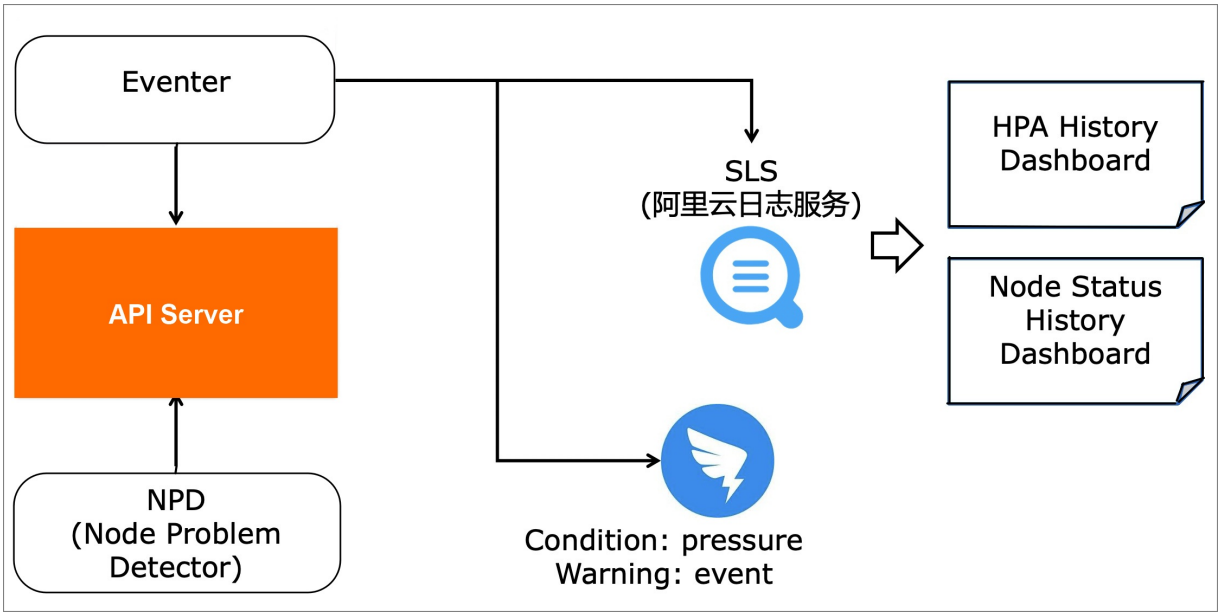
13.3.4. 事件监控

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、准确性和场景上的欠缺。您可以通过使用NPD结合SLS的Kubernetes事件中心、配置NPD集群检查项以及异常事件离线功能、使用钉钉、使用SLS离线Kubernetes事件及使用EventBridge离线Kubernetes事件，实时监控集群的异常与问题。

背景信息

Kubernetes的架构设计基于状态机，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。

ACK提供开箱即用的容器场景事件监控方案，通过ACK维护的NPD（node-problem-detector）以及包含在NPD中的kube-eventer提供容器事件监控能力。



- NPD（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，结合kube-eventer可以实现节点事件告警的闭环。更多信息，请参见NPD。
- kube-eventer是ACK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、EventBridge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。更多信息，请参见kube-eventer。

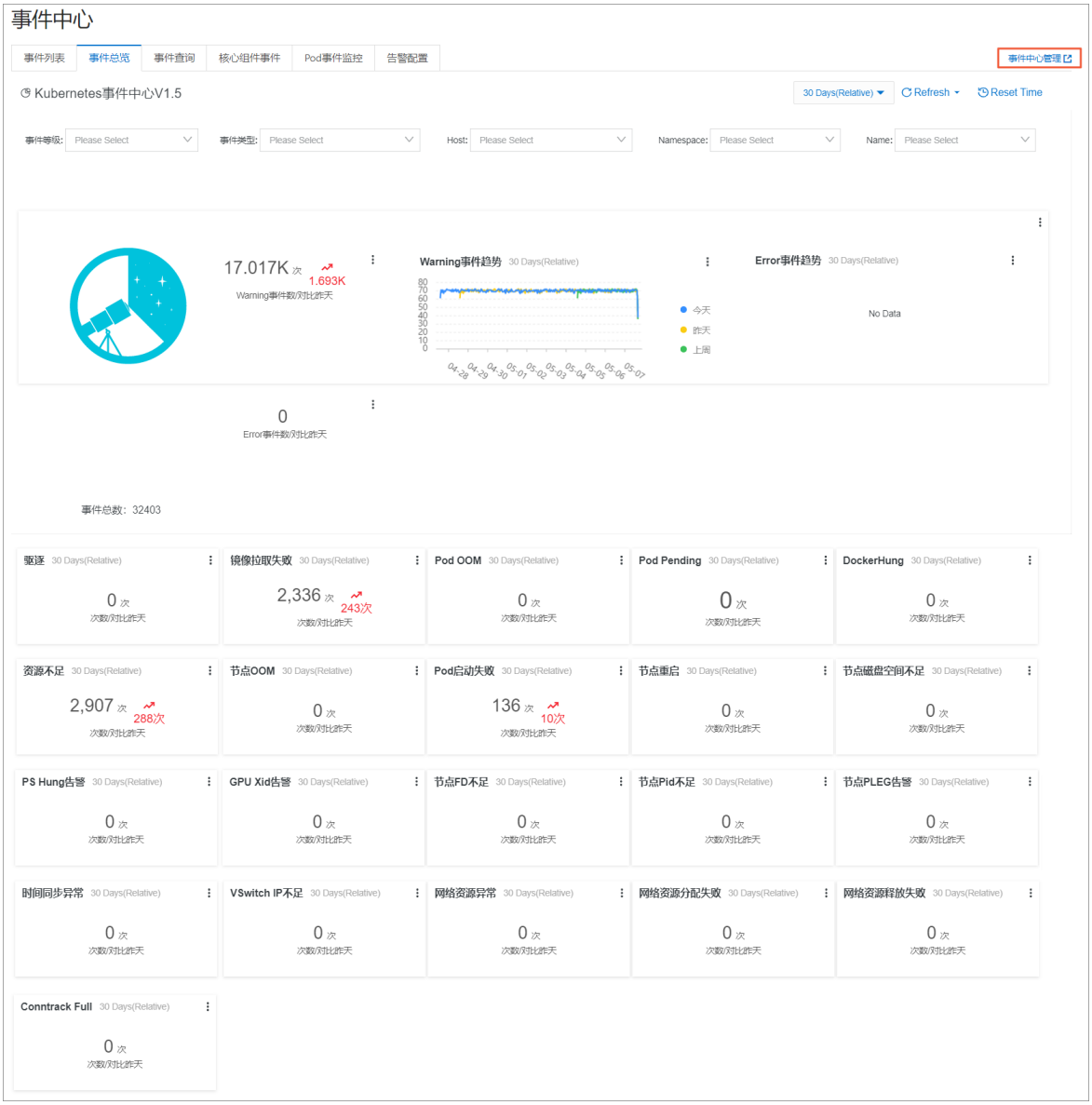
本文通过以下五种场景为您介绍事件监控。

场景一：使用NPD结合SLS的Kubernetes事件中心监控集群事件

NPD根据配置与第三方插件检测节点的问题或故障，生成相应的集群事件。而Kubernetes集群自身也会因为集群状态的切换产生各种事件。例如Pod驱逐，镜像拉取失败等异常情况。日志服务SLS（Log Service）的Kubernetes事件中心实时汇聚Kubernetes中的所有事件并提供存储、查询、分析、可视化、告警等能力。将集群事件接入日志服务的Kubernetes事件中心操作步骤如下。

方法一：如果您在创建集群时，已选中**安装node-problem-detector并创建事件中心**，请按照以下步骤查看Kubernetes事件中心。关于通过创建集群，安装node-problem-detector并创建事件中心的详细信息，请参见创建Kubernetes托管版集群。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 选择**运维管理 > 事件中心**。
5. 单击右上角的**事件中心管理**，在**K8s事件中心**页面左侧导航栏，单击目标集群名称左边的展开图标 ，查看Kubernetes事件中心相关信息。
   您可以查看Kubernetes事件的总览及详情、Pod生命周期，还可以进行自定义查询及告警配置。



方法二：如果您在创建集群时，未配置事件中心功能，请按照以下步骤使用Kubernetes事件中心。

1. 对被监控集群安装NPD并激活SLS功能。具体安装步骤，请参见场景三：使用钉钉实现Kubernetes监控告警。

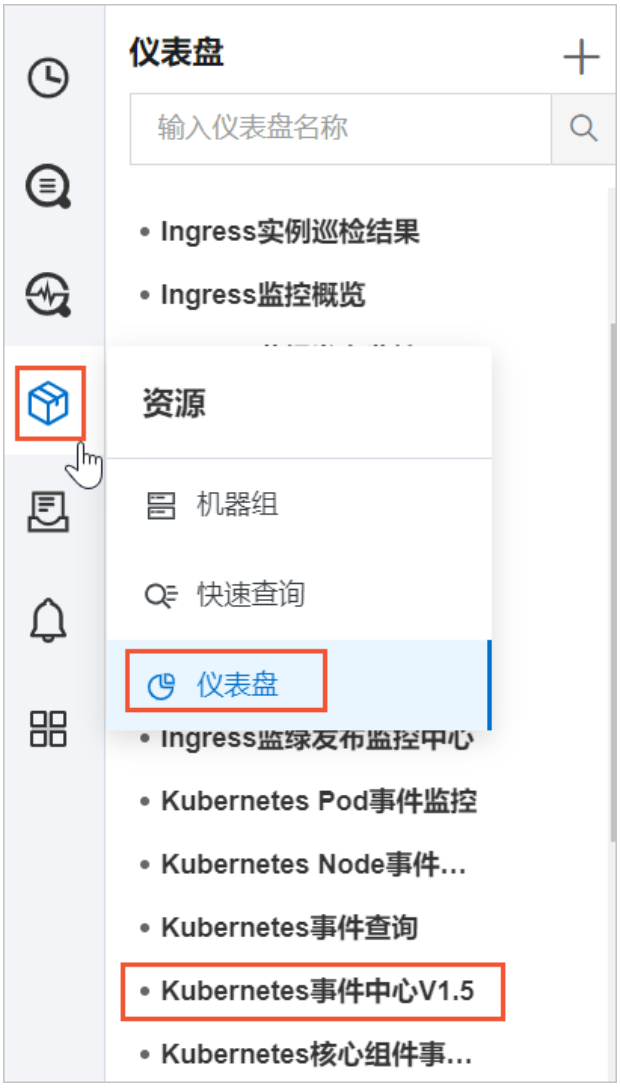
> **说明** 如果您已安装NPD但未激活SLS，请先卸载旧版本。
>
> i. 在控制台左侧导航栏中，单击**集群**。
> ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
> iii. 选择**应用** > Helm。
> iv. 在Helm页面，删除原有的ack-node-problem-detector组件以卸载旧版本NPD。

您在配置NPD参数时，需将 `eventer.sinks.sls.enabled` 设置为true，以创建SLS日志项目。

```
sinks:
   sls:
    # If you want the monitoring results to be notified by sls, set enabled to true.
     # sls_log_project
    enabled: true
     # You can view the project information by logging in to the
```

成功部署NPD后，NPD会自动在SLS控制台内创建与集群对应的项目。

2. 在日志服务管理控制台为集群配置Kubernetes事件中心。
   i. 在**接入数据**区域，单击**Kubernetes-标准输出**。
   ii. 设置**项目Project**为配置NPD时默认的SLS Project，选择**日志库Logstore**为自动创建的**k8s-event**。
   iii. 单击**下一步**，然后单击**确认安装完毕**。
3. 在日志服务管理控制台**Project列表**区域，单击目标Project名称。
4. 在左侧导航栏中，单击**仪表盘**，然后单击**Kubernetes事件中心V1.5**。



在Kubernetes事件中心大盘中即可看到从集群中汇总的事件。

场景二：配置NPD集群检查项以及异常事件离线功能

NPD（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，结合kube-eventer可以实现节点事件告警的闭环。具体的操作步骤如下。

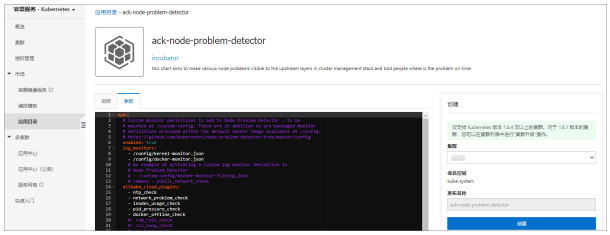
1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，在左侧导航栏选择**市场 > 应用目录**，在右侧查找并选中**ack-node-problem-detector**。

   > **说明** 若已经开启事件中心，请先删除原有的ack-node-problem-detector组件。
   >
   > i. 在控制台左侧导航栏中，单击**集群**。
   > ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   > iii. 选择**应用 > Helm**。
   > iv. 在**Helm**页面，删除原有的ack-node-problem-detector组件。

3. 在**应用目录 - ack-node-problem-detector**页面，在右侧的**创建**区域选择对应的集群。

   同时可以看到命名空间已设定为*kube-system*，发布名称已设定为*ack-node-problem-detector*，在**参数**页

签，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。
参数说明

参数	说明	默认值
npd.image.repository	NPD的镜像地址。	registry.aliyuncs.com/acs/node-problem-detector
npd.image.tag	NPD的镜像版本。	v0.6.3-28-160499f
alibaba_cloud_plugins	检测节点问题的插件列表，请参见下文的问题检查插件。	包括 fd check 、 ntp check 、 network problem_check 、 inode_usage_check
plugin settings.check_fd_warning_percentage	fd_check 监控插件的百分比报警阈值。	80
plugin settings.inode_warning_percentage	inode 监控插件的百分比报警阈值。	80
controller.regionId	监控插件所在集群的地域。目前仅支持 cn-hangzhou 、 cn-beijing 、 cn-shenzhen 及 cn-shanghai 。	集群所在区域
controller.clusterType	监控插件所在集群的类型。	ManagedKubernetes
controller.clusterId	监控插件所在集群的ID。	所在集群的ID
controller.clusterName	监控插件所在集群的名称。	所在集群的名称
controller.ramRoleType	访问控制角色类型。若值为 restricted ，表示使用默认的Token权限。	集群默认的访问控制角色类型
eventer.image.repository	eventer 的镜像地址。	registry.cn-hangzhou.aliyuncs.com/acs/eventer
eventer.image.tag	eventer 的镜像版本。	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 日志服务的离线链路。	false
eventer.sinks.sls.project	日志服务的Project名称。	无

参数	说明	默认值
eventer.sinks.sls.logstore	日志服务的Project下Logstore的名称。	无
eventer.sinks.dingtalk.enabled	是否开启 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。	无
eventer.sinks.dingtalk.token	钉钉机器人的Token。	无
eventer.sinks.dingtalk.monitorkinds	过滤的资源类型。	无
eventer.sinks.dingtalk.monitornamespaces	过滤的资源命名空间。	无
eventer.sinks.eventbridge.enable	是否开启 eventer 离线 eventBridge 数据链路。	false

NPD支持的问题检查插件如下表。

插件名	功能	说明
fd_check	检查集群节点系统的已打开的文件描述符是否超过最大上限的80%。	80%为默认值，可通过参数修改。该检查项消耗资源较多，不建议打开。
ram_role_check	检查集群节点是否具有RAM Role以及相应的鉴权密钥。	无
ntp_check	检查集群节点上的NTP时间同步服务是否正常运行。	默认开启项。
nvidia_gpu_check	检查集群节点上的NVIDIA GPU计算卡是否具有有效的 xid 。	无
network_problem_check	检查集群节点上的 conntrack 表用量是否超过了总量的90%。	默认开启项。
inodes_usage_check	检查集群节点系统盘的 inode 是否超过了总量的80%。	80%为默认值，可通过参数修改。默认开启项。
csi_hang_check	检查集群节点上的CSI存储插件状态是否正常。	无
ps_hang_check	检查集群节点系统中是否有状态为D（挂起且无法唤醒）的进程。	无
public_network_check	检查集群节点是否可以访问公网。	无
irqbalance_check	检查集群节点系统中的 irqbalance 服务是否正常。	无
pid_pressure_check	检查集群节点系统中的进程 pid 数与系统内核中允许的 pid 最大数占比是否超过了85%。	默认开启项。

插件名	功能	说明
docker_offline_check	检测集群节点上的 docker daemon 服务是否正常。	默认开启项。

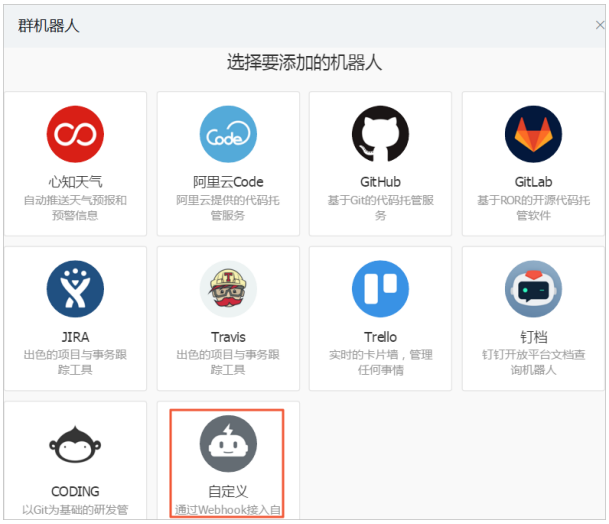
> **说明** 默认开启项：创建集群时**日志服务**选中**安装node-problem-detector并创建事件中心**或在集群**组件管理**中一键安装ack-node-problem-detector组件时的默认开启检查项。从**应用目录**中安装ack-node-problem-detector组件时还需要手动开启检查项。

4. 在右侧的**创建**区域，单击**创建**。
   在集群列表页，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**，然后单击**守护进程集**页签，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。

场景三：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**智能群助手**，单击**添加机器人**，单击**+**，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. 根据以下信息配置群机器人后，阅读并同意服务及免责条款，然后单击**完成**。

参数	描述
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。
安全设置	安全设置目前支持3种方式：自定义关键词、加签和IP地址（段）。目前集群的事件监控仅支持第一种方式，即**自定义关键词**。 选中**自定义关键词**，填入*Warning*可接收所有监控报警。如果发现机器人消息发送过于频繁，可增加关键词进行过滤，最多支持设置10个关键词。ACK发送消息时也会同步此关键词。

5. 单击**复制**，复制**webhook**地址。

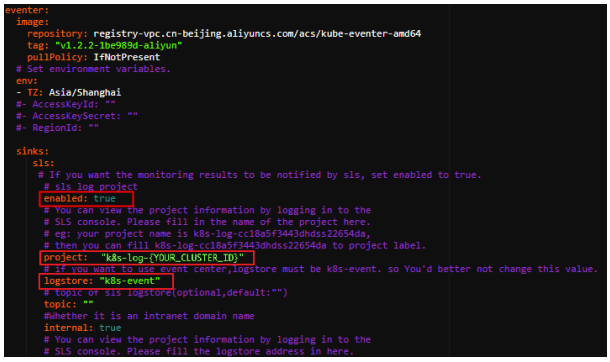


> **说明** 在**群机器人**页面，选择目标群机器人，单击右侧
>
> 图标可以进行以下操作：
>
> - 修改群机器人的头像及机器人名字。
> - **开启**或**关闭**消息推送。
> - 重置**webhook**地址。
> - 删除群机器人。

6. 登录容器服务管理控制台。
7. 在左侧导航栏选择**市场 > 应用目录**，在**阿里云应用**页面选中**ack-node-problem-detector**。

> **说明** 若已经开启事件中心，请先删除原有的ack-node-problem-detector组件。
>
> i. 在控制台左侧导航栏中，单击**集群**。
> ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
> iii. 选择**应用 > Helm**。
> iv. 在**Helm**页面，删除原有的ack-node-problem-detector组件。

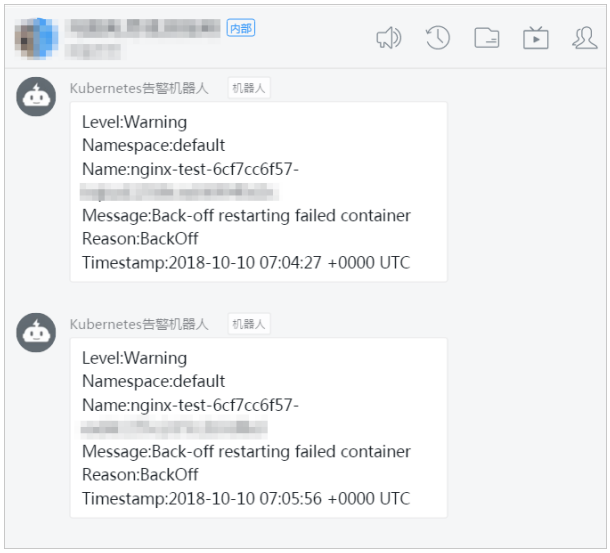
8. 在**应用目录 - ack-node-problem-detector**页面的**参数**页签，修改以下内容。
   - 将 `npd` 下方的 `enabled` 设置为false。
   - 将 `eventer.sinks.dingtalk.enabled` 设置为true。
   - 通过步骤5创建**webhook**地址查看Token，填入Token字段。



9. 在右侧的创建页面，选择对应的集群，同时可以看到命名空间已设定为kube-system，发布名称已设定为ack-node-problem-detector，然后单击**创建**。

预期结果：

部署成功后30秒，**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。



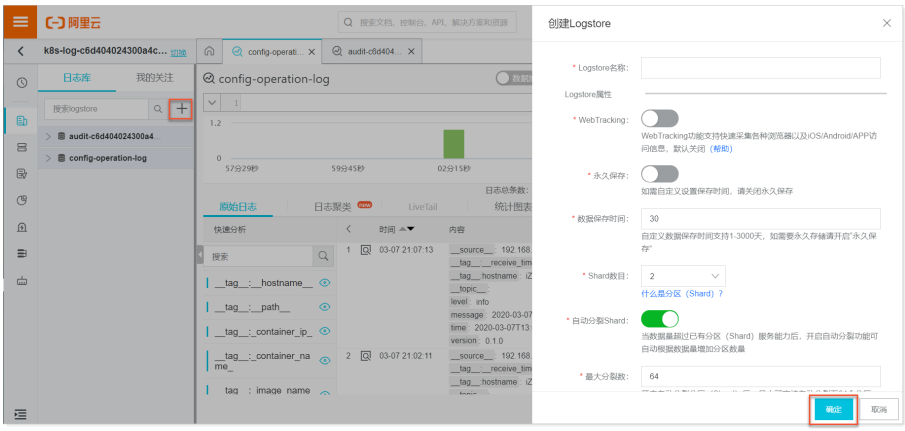
场景四：使用SLS离线Kubernetes事件

阿里云日志服务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。详情请参见创建并使用Kubernetes事件中心。

1. 创建Project与Logstore。
   i. 登录日志服务管理控制台。
   ii. 在**Project列表**区域，单击**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。

   iii. 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。
   iv. 默认进入**日志库**页面，在页面左侧，单击加号**+**，弹出**创建Logstore**对话框。
   v. 填写日志库配置信息并单击**确定**。
   本示例创建名为k8s-logstore的日志库。



   vi. 创建完毕后，页面会提示您使用数据接入向导。单击**数据接入向导**，弹出**接入数据**对话框。

vii. 选择**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。

i. 登录容器服务管理控制台。

ii. 在左侧导航栏选择**市场 > 应用目录**，在右侧选中**ack-node-problem-detector**。

> **说明** 若已经开启事件中心，请先删除原有的ack-node-problem-detector组件。
>
> a. 在控制台左侧导航栏中，单击**集群**。
>
> b. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
>
> c. 选择**应用 > Helm**。
>
> d. 在**Helm**页面，删除原有的ack-node-problem-detector组件。

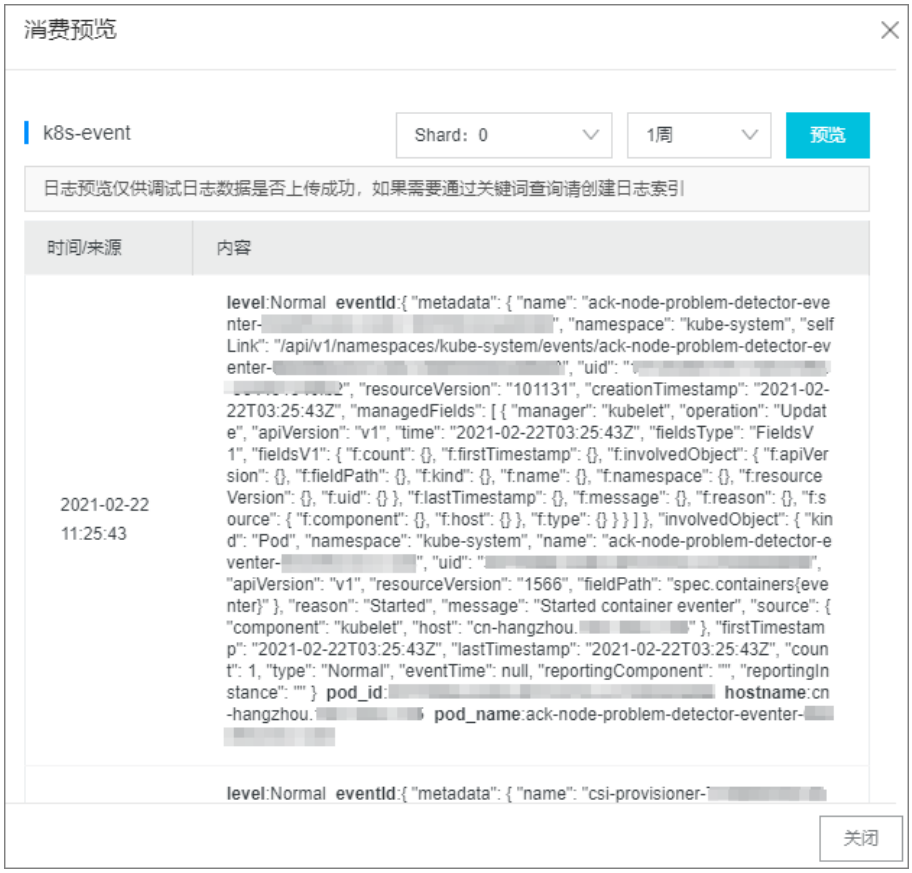
iii. 在**应用目录 - ack-node-problem-detector**中，在**参数**页签，修改以下内容。

- 将 `npd` 下方的 `enabled` 设置为false。
- 将 `eventer.sinks.sls.enabled` 设置为true。
- 将步骤1创建好的 `Project` 和 `Logstore` 分别填入各字段。
  若未在创建集群时自定义 `Project`，默认 `Project` 为k8s-log-{YOUR_CLUSTER_ID}

```
eventer:
  image:
    repository: registry-vpc.cn-beijing.aliyuncs.com/acs/kube-eventer-amd64
    tag: "v1.2.2-1be989d-aliyun"
    pullPolicy: IfNotPresent
  # Set environment variables.
  env:
  - TZ: Asia/Shanghai
  #- AccessKeyId: ""
  #- AccessKeySecret: ""
  #- RegionId: ""

  sinks:
    sls:
     # If you want the monitoring results to be notified by sls, set enabled to true.
     # sls log project
     enabled: true
     # You can view the project information by logging in to the
     # SLS console. Please fill in the name of the project here.
     # eg: your project name is k8s-log-cc18a5f3443dhdss22654da,
     # then you can fill k8s-log-cc18a5f3443dhdss22654da to project label.
     project: "k8s-log-{YOUR_CLUSTER_ID}"
     # If you want to use event center,logstore must be k8s-event. so You'd better not change this value.
     logstore: "k8s-event"
     # topic of sls logstore(optional,default:"")
     topic: ""
     #whether it is an intranet domain name
     internal: true
     # You can view the project information by logging in to the
     # SLS console. Please fill the logstore address in here.
```

iv. 配置完成后，单击**创建**，将**eventer**部署到目标集群中。

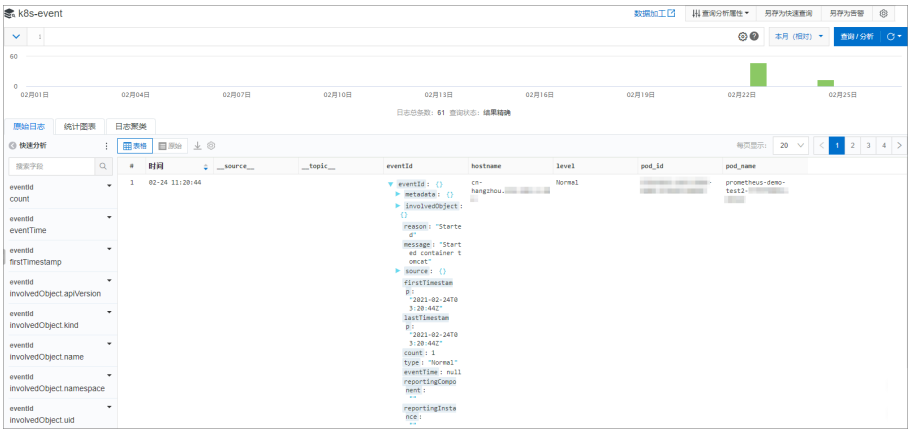
3. 操作集群（例如，删除Pod或者创建应用等）产生事件后，登录日志服务控制台查看数据采集。请参见普通消费。



4. 设置索引与归档。请参见配置索引。
   i. 在日志服务控制台**Project列表**区域，单击Project名称。
   ii. 单击日志库名称后的图标，选择**查询分析**。
   iii. 单击右上角的**开启索引**。
   iv. 在**查询分析**对话框中配置索引。

v. 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。

vi. 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**。请参见通过日志服务投递日志到MaxCompute和投递日志到OSS。

场景五：使用EventBridge离线Kubernetes事件

事件总线EventBridge是阿里云提供的一款无服务器事件总线服务，支持阿里云服务、自定义应用及SaaS应用以标准化、中心化的方式接入，并能够以标准化的CloudEvents 1.0协议在这些应用之间路由事件。容器服务事件可支持通过离线到EventBridge实现构建松耦合、分布式的事件驱动架构。关于EventBridge的详情，请参见什么是事件总线EventBridge。

1. 开通事件总线EventBridge。具体操作，请参见开通事件总线EventBridge并授权。
2. 登录容器服务管理控制台。
3. 在控制台左侧导航栏选择**市场 > 应用目录**，在右侧选中**ack-node-problem-detector**。

> **说明** 若已经开启事件中心，请先删除原有的ack-node-problem-detector组件。
>
> i. 在控制台左侧导航栏中，单击**集群**。
>
> ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
>
> iii. 选择**应用 > Helm**。
>
> iv. 在**Helm**页面，删除原有的ack-node-problem-detector组件。

4. 在**应用目录 - ack-node-problem-detector**中，在**参数**页签，修改以下内容。

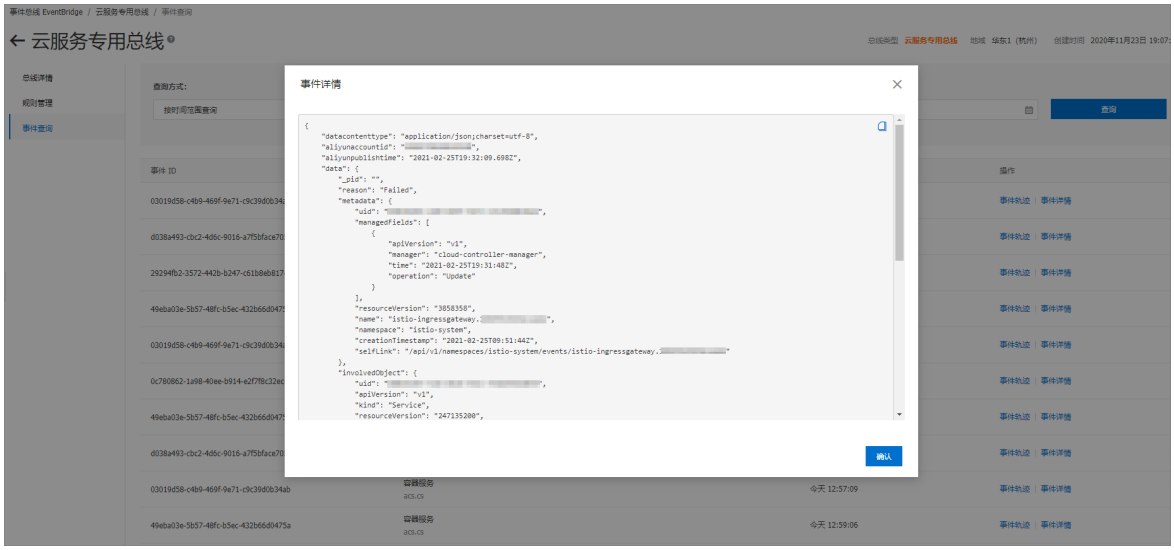
   配置事件中心并开启EventBridge事件离线数据链路。

   - 将 `npd` 下方的 `enabled` 设置为true。
   - 将 `eventer.sinks.eventbridge.enable` 设置为true。

```
eventbridge:
  # If you want the monitoring results to be notified by EventBridge,
  # first, you need to access to EventBridgeset (https://eventbridge.console.aliyun.com)
  # then set enabled to true.
  # You can visit EventBridge website to get more information. https://eventbridge.console.aliyun.com
  enabled: true
```

5. 配置完成后，单击**创建**，完成ack-node-problem-detector事件中心组件的安装。

6. 数据链路开启成功后，可在事件总线EventBridge控制台查看容器事件。
   i. 登录事件总线EventBridge控制台。
   ii. 在控制台左侧导航栏中，单击**云服务专用总线**。
   iii. 在**云服务专用总线**页面左侧导航栏中，单击**事件查询**。
   iv. 选择事件查询方式及范围，单击**查询**。
   v. 在事件列表**操作**列单击**事件详情**。



13.3.5. 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，参见创建Kubernetes托管版集群。
- 您已连接到集群，方便快速查看节点标签等信息，参见通过kubectl工具连接集群。

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。

i. 登录容器服务管理控制台。

ii. 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**。

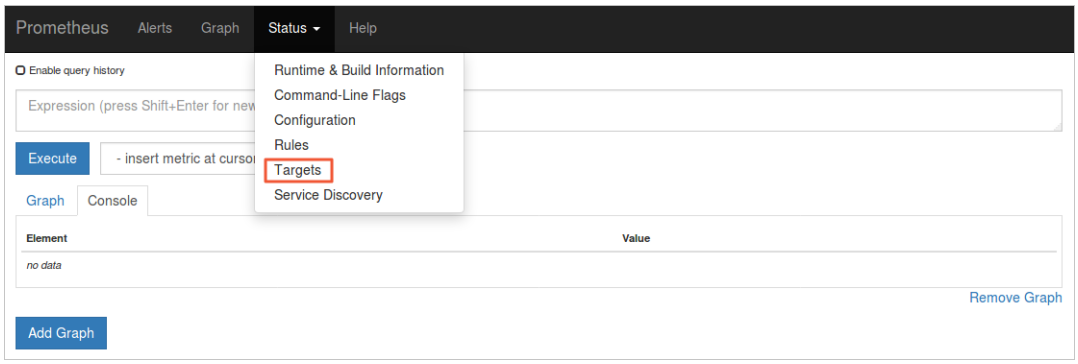
iii. 在**应用目录-ack-prometheus-operator**页面右侧的**创建**区域，选择目标集群后，单击**创建**，添加Prometheus-operator。
查看部署结果：

a. 执行以下命令，将集群中的Prometheus映射到本地9090端口。

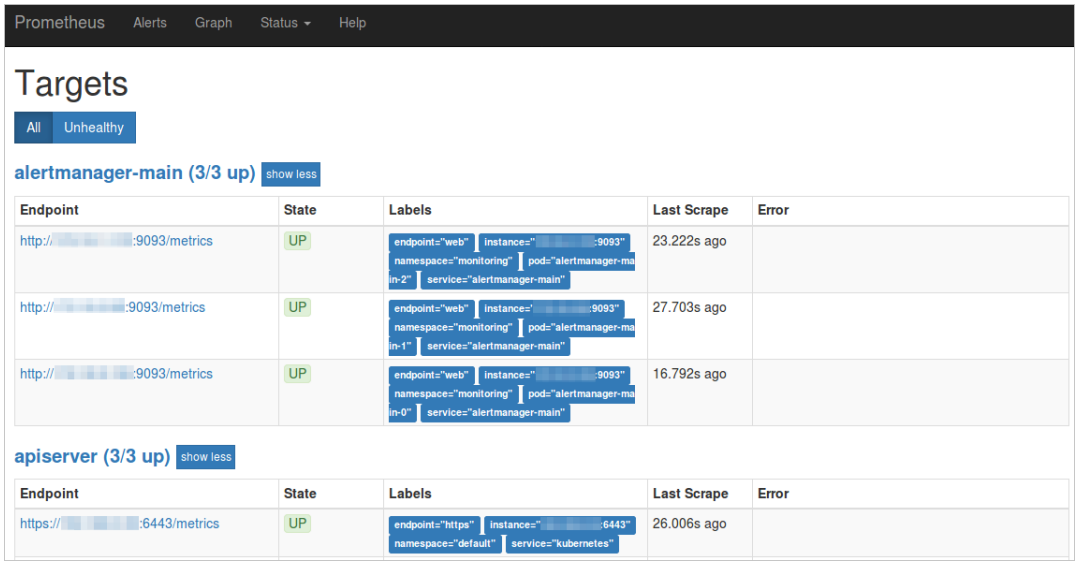
```
kubectl port-forward svc/ack-prometheus-operator-prometheus 9090:9090 -n monitoring
```

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。

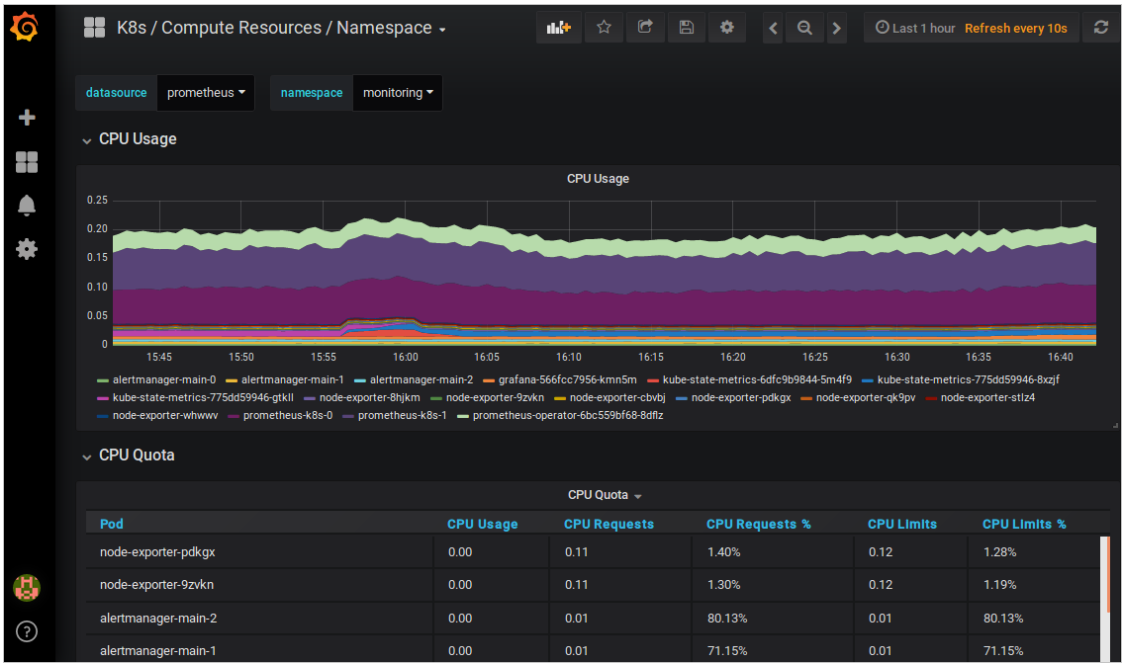


2. 查看与展示数据聚合。

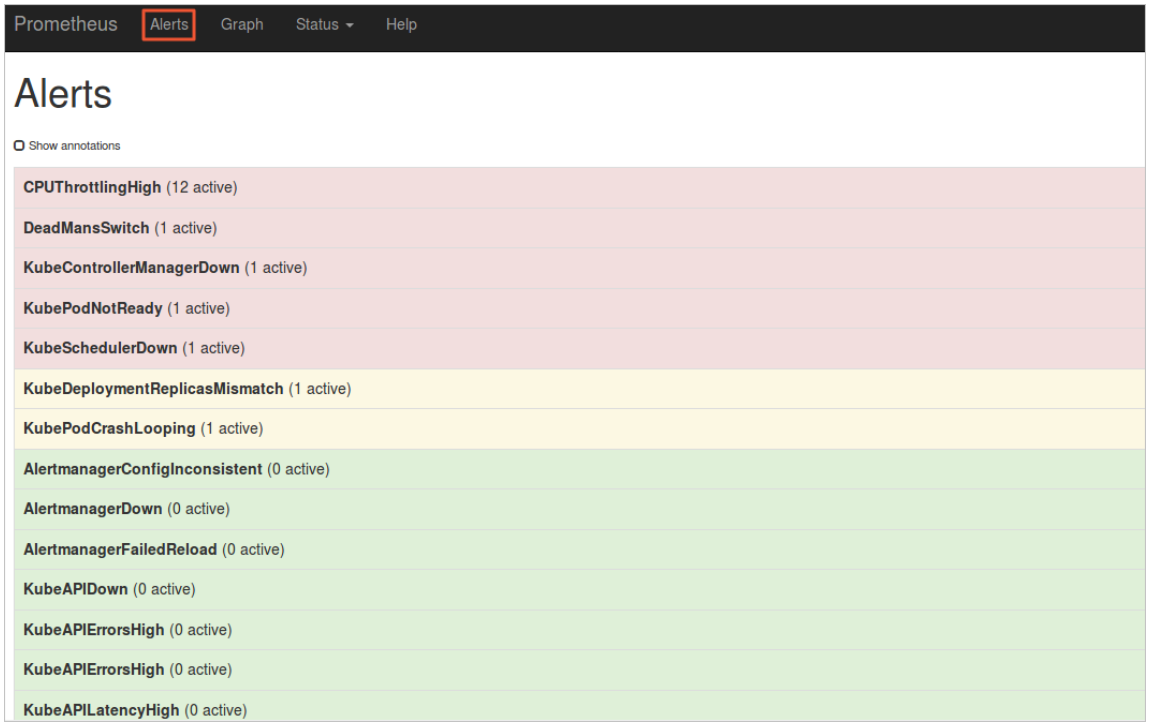
i. 执行以下命令，将集群中的Grafana映射到本地3000端口。

```
kubectl -n monitoring port-forward svc/ack-prometheus-operator-grafana 3000:80
```

ii. 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。

使用容器服务的Prometheus完整的最佳实践请参见云上Prometheus监控运维最佳实践。

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：
    a. 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=xxxxxxxxx*。

```
##
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
```

    b. 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*。

c. 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*。

```yaml
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）。

```yaml
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
```

dingtalk

- 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-xxxxxxxxxxxx.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-xxxxxxxxxxxx.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，storage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。

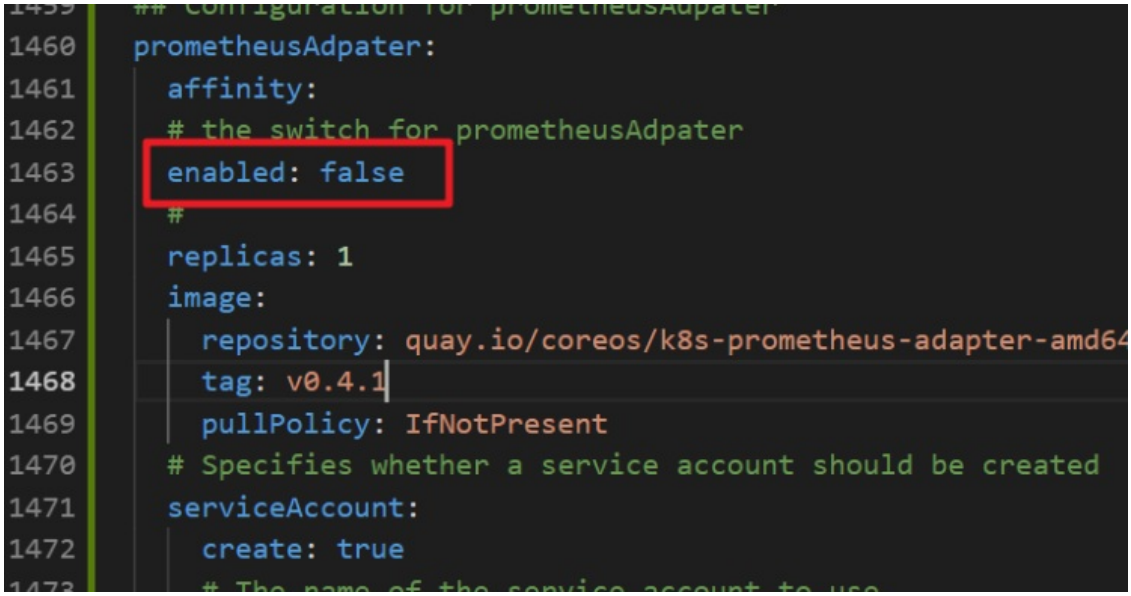
```
storageSpec:
#  volumeClaimTemplate:
#    spec:
#      storageClassName: alicloud-disk-ssd
#      accessModes: ["ReadWriteOnce"]
#      resources:
#        requests:
#          storage: 50Gi
#    selector: {}
```

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，请参见云盘存储卷概述。

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。



配置完成后，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。

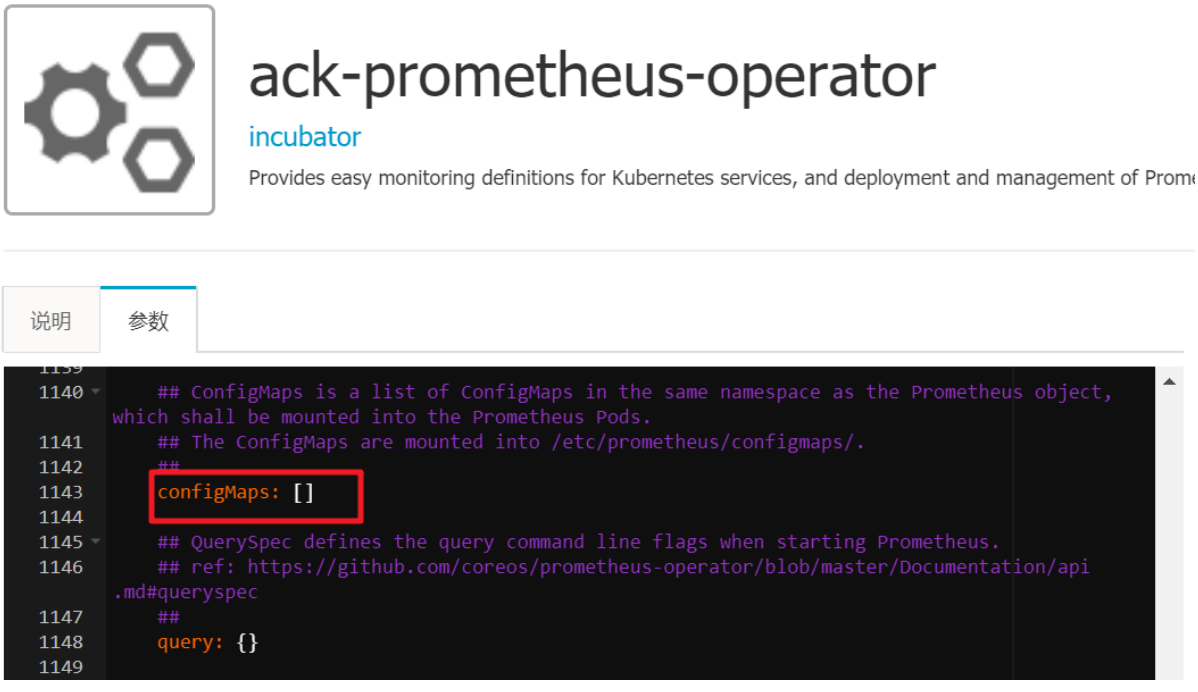
```
kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1"
```

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. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Helm**。
5. 在目标Helm应用右侧单击**更新**，进入**更新发布**页面。
6. 在prometheus和alertmanager的configmaps字段中，填入您自定义configmap名称后，单击**更新**。

```
    ## ConfigMaps is a list of ConfigMaps in the same namespace as the Alertmanager object, which
shall be mounted into the Alertmanager Pods.
    ## The ConfigMaps are mounted into /etc/alertmanager/configmaps/.
    ##
    configMaps:
    - "special-config"
```

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。

```yaml
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。

```yaml
## 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的持久化操作如下：
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
  iv. 在左侧导航栏，选择**应用 > Helm**。
  v. 找到ack-prometheus-operator，然后单击右侧**更新**，弹出**更新发布**对话框。
  vi. 在grafana的字段下找到persistence选项，按照下图配置完成Grafana的持久化操作。

```
428  persistence:
429    #Persistent switch
430    enabled: false
431    #storageClass Name,there are four types of storageclasses: alicloud-disk-available,alicloud-disk
-efficiency,alicloud-disk-essd,alicloud-disk-ssd
432    storageClassName: alicloud-disk-efficiency
433    accessModes:
434      - ReadWriteOnce
435    size: 30Gi
436    # annotations: {}
437    # subPath: ""
438    #existingClaim:
```

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。
  i. 获取钉钉的webhook地址。请参见场景三：使用钉钉实现Kubernetes监控告警。
  ii. 找到dingtalk字段，将enabled设置为true，将Token字段填入钉钉的webhook地址。请参见告警配置中的**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: customresourcedefinit
ions.apiextensions.k8s.io "xxxxxxx.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 `当前集群暂时无法访问，请稍后重试或提交工单反馈` 信息。
  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，详情请参见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？
  当部署好prometheus-operator后，如果有开启部分功能，在集群信息页面，选择**应用 > Helm**，在**ack-prometheus-operator**右侧，单击**更新**，找到对应的开关，做相应的设置，然后单击**更新**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
```

- Grafana dashboard显示有问题
  在集群信息页面选择**应用 > Helm**，在**ack-prometheus-operator**右侧，单击**更新**，查看**clusterVersion**的值是否为正确的集群版本。Kubernetes集群是1.16以前的版本，这里请填写1.14.8-aliyun.1，1.16及以后的版本，请填写1.16.6-aliyun.1。
- 删除ack-prometheus的命名空间后，重新安装ack-prometheus失败。
  只删除ack-prometheus的命名空间，会导致资源删除后有残留配置，影响再次安装。您可以执行以下操作，删除残余配置。
  i. 删除RBAC权限。
  a. 删除ClusterRole。

```
kubectl delete ClusterRole ack-prometheus-operator-grafana-clusterrole
kubectl delete ClusterRole ack-prometheus-operator-kube-state-metrics
kubectl delete ClusterRole psp-ack-prometheus-operator-kube-state-metrics
kubectl delete ClusterRole psp-ack-prometheus-operator-prometheus-node-exporter
kubectl delete ClusterRole ack-prometheus-operator-operator
kubectl delete ClusterRole ack-prometheus-operator-operator-psp
kubectl delete ClusterRole ack-prometheus-operator-prometheus
kubectl delete ClusterRole ack-prometheus-operator-prometheus-psp
```

b. 删除ClusterRoleBinding。

```
kubectl delete ClusterRoleBinding ack-prometheus-operator-grafana-clusterrolebinding
kubectl delete ClusterRoleBinding ack-prometheus-operator-kube-state-metrics
kubectl delete ClusterRoleBinding psp-ack-prometheus-operator-kube-state-metrics
kubectl delete ClusterRoleBinding psp-ack-prometheus-operator-prometheus-node-exporter
kubectl delete ClusterRoleBinding ack-prometheus-operator-operator
kubectl delete ClusterRoleBinding ack-prometheus-operator-operator-psp
kubectl delete ClusterRoleBinding ack-prometheus-operator-prometheus
kubectl delete ClusterRoleBinding ack-prometheus-operator-prometheus-psp
```

ii. 删除CRD。

```
kubectl delete crd alertmanagerconfigs.monitoring.coreos.com
kubectl delete crd alertmanagers.monitoring.coreos.com
kubectl delete crd podmonitors.monitoring.coreos.com
kubectl delete crd probes.monitoring.coreos.com
kubectl delete crd prometheuses.monitoring.coreos.com
kubectl delete crd prometheusrules.monitoring.coreos.com
kubectl delete crd servicemonitors.monitoring.coreos.com
kubectl delete crd thanosrulers.monitoring.coreos.com
```

13.3.6. 阿里云Prometheus监控

您可以通过阿里云Prometheus监控查看容器服务Kubernetes版ACK（Container Service for Kubernetes）预先配置的监控大盘和监控性能指标。本文为您介绍如何在ACK中接入阿里云的Prometheus监控，如何配置Prometheus监控报警及如何自定义Prometheus监控指标并通过Grafana展示。

背景信息

阿里云Prometheus监控全面对接开源Prometheus生态，支持类型丰富的组件监控，提供多种开箱即用的预置监控大盘，且提供全面托管的Prometheus服务。借助阿里云Prometheus监控，您无需自行搭建Prometheus监控系统，因而无需关心底层数据存储、数据展示、系统运维等问题。

有关阿里云Prometheus监控的更多信息，请参见什么是Prometheus监控。

开启阿里云Prometheus监控

您可以通过以下三种方式开启阿里云Prometheus监控：

通过配置集群参数开启Prometheus监控。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**创建集群**。
4. 选择目标集群模板，然后配置创建集群参数。在**组件配置**页面，选中**使用Prometheus监控服务**。



有关创建集群的具体步骤，请参见创建Kubernetes托管版集群。

> **说明** 在创建集群时，系统默认选中**使用Prometheus监控服务**。

集群创建完成后，系统将自动配置阿里云Prometheus监控服务。

通过ACK控制台Prometheus监控开启Prometheus监控。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理左侧导航栏中，选择**运维管理 > Prometheus监控**。
5. 在**Prometheus监控**页面中间，单击**开始安装**。

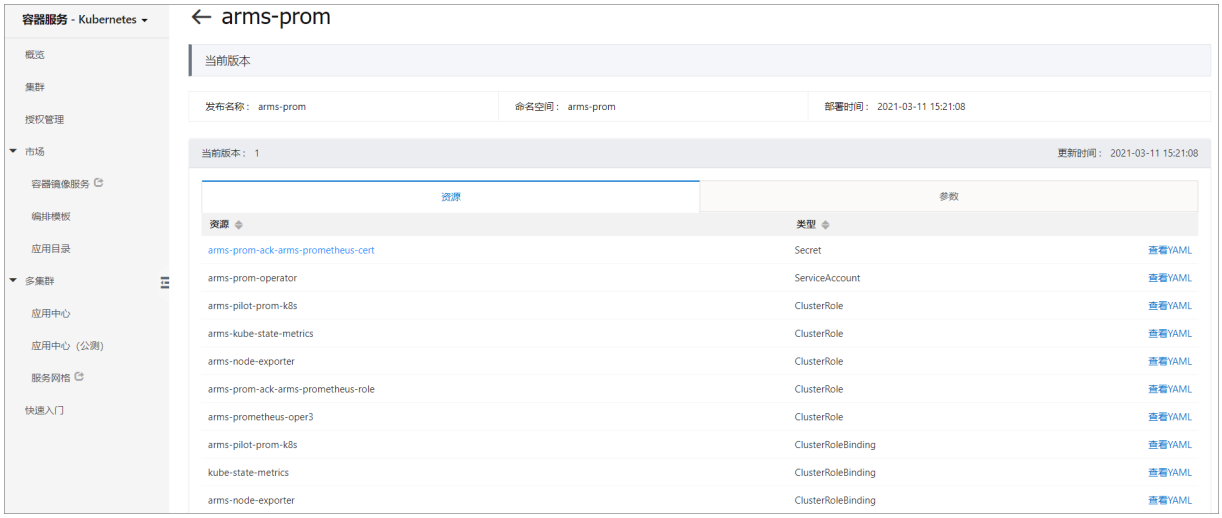
通过ACK控制台应用目录开启阿里云Prometheus监控。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的**阿里云应用**页签中，搜索并单击**ack-arms-prometheus**。
4. 在**应用目录 - ack-arms-prometheus**右侧的**创建**区域中，选择目标集群，然后单击**创建**。

> **说明** 命名空间和发布名称默认为arms-prom。

执行结果

安装完成后会自动跳转到应用**arms-prom**页面，可查看应用信息。



查看阿里云Prometheus Grafana大盘

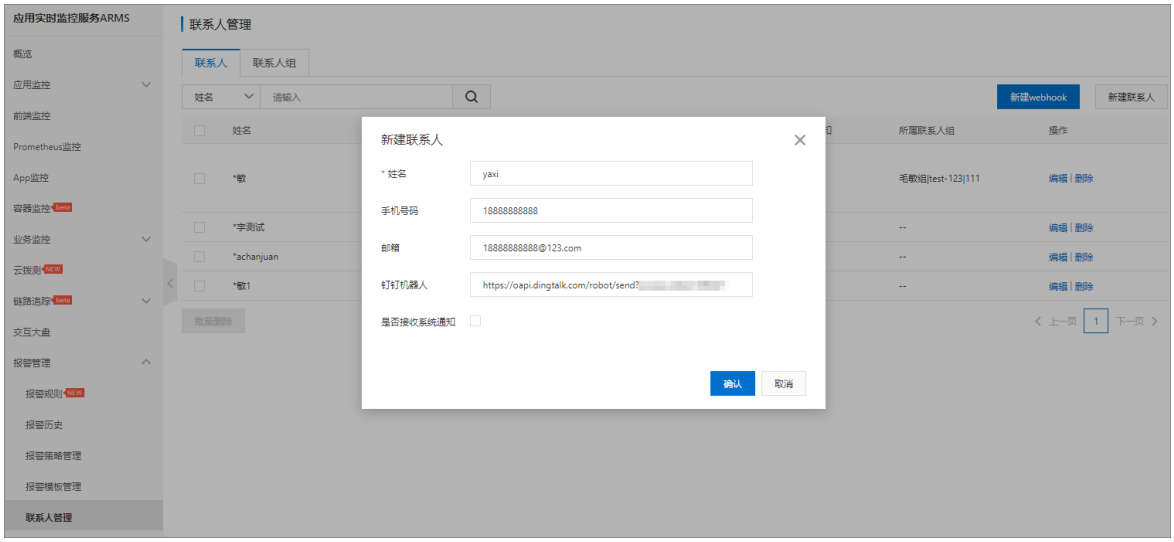
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > Prometheus监控**。
5. 在**Prometheus监控**页面，单击需要查看的Grafana监控大盘名，即可查看相应的监控数据。

配置Prometheus监控报警

为监控任务创建报警可在满足报警条件时通过邮件、短信、钉钉等渠道实时报警，主动帮助您发现异常。报警规则被触发时会向您指定的联系人分组发送通知，而在创建联系人分组之前必须先创建联系人。创建联系人时，您可以指定联系人用于接收通知的手机号码和邮箱地址，也可以提供用于自动发送报警通知的钉钉机器人地址。

> **说明** 如需将钉钉机器人添加为联系人，则需要先获取钉钉机器人的地址。具体操作，请参见设置钉钉机器人报警。

1. 登录ARMS控制台。
2. 在控制台左侧导航栏，选择**报警管理 > 联系人管理**。
3. 在**联系人**页签，单击右上角的**新建联系人**。配置联系人信息，然后单击**确认**。



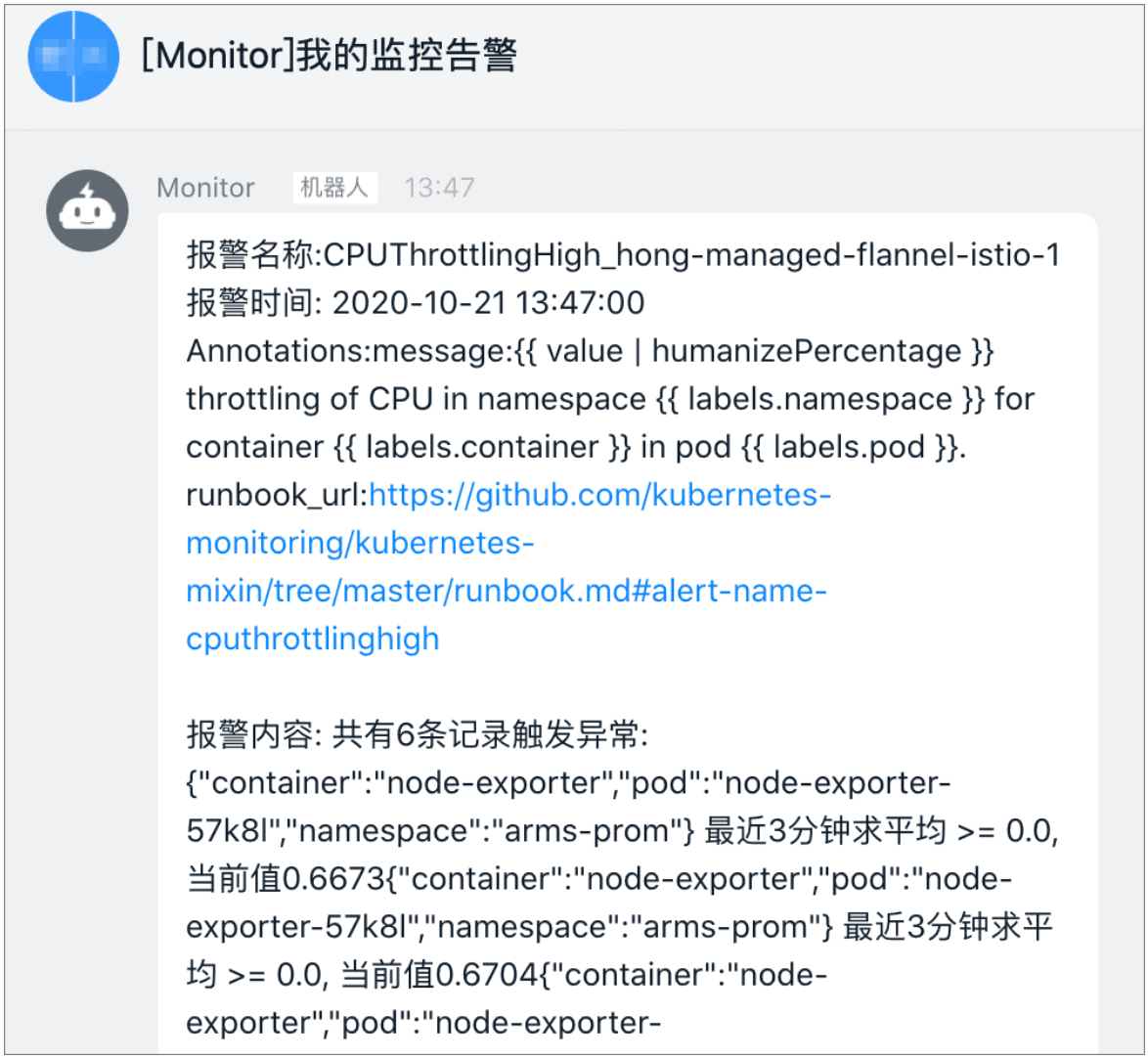
4. 配置报警规则。
   i. 登录ARMS控制台。
   ii. 在控制台左侧导航栏单击**Prometheus监控**。
   iii. 在**Prometheus监控**页面单击**K8s**列目标集群名称。
   iv. 在左侧导航栏单击**报警配置**。
   v. 在报警规则说明列表单击对应报警**操作**列的**编辑**，输入PromQL语句配置报警规则，配置完成后单击**确定**。或者单击**开启**可直接开启预置的报警规则。

      关于PromQL语句的配置，请参见快速创建ARMS报警（旧版）。

> **说明** 报警规则也可在ARMS控制台的**报警管理** > **报警策略管理** 进行管理。

验证结果

进行手动测试，触发钉钉告警。正常触发告警如下：



自定义Prometheus监控指标并通过Grafana展示

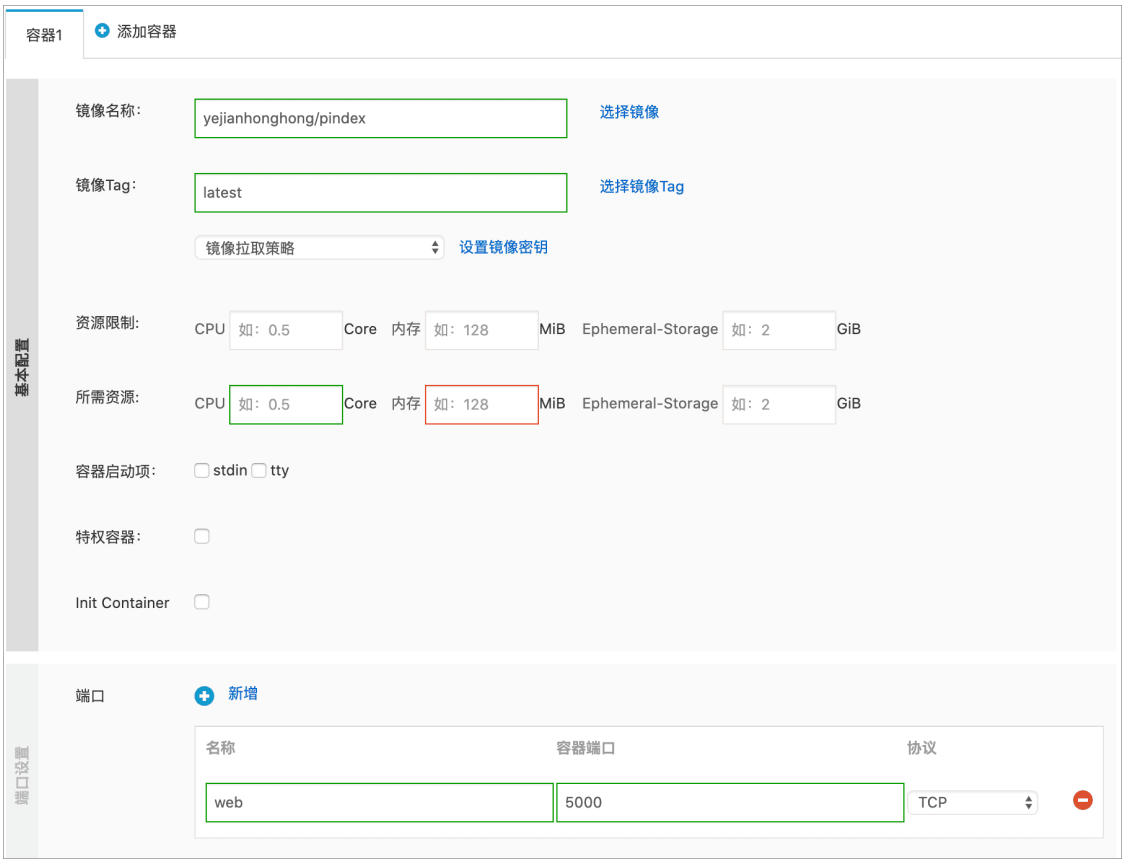
方式一：通过Annotations自定义指标监控

通过给Deployment Pod Template加入Annotations的方式自定义监控，阿里云ARMS应用监控组件将会使ARMS Prometheus监控自动获取Pod自定义指标。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，创建应用。
   i. 单击**使用镜像创建**。
   ii. 在配置向导**应用基本信息**中填写应用的基本信息，单击**下一步**。

iii. 创建一个Web应用，暴露5000端口。

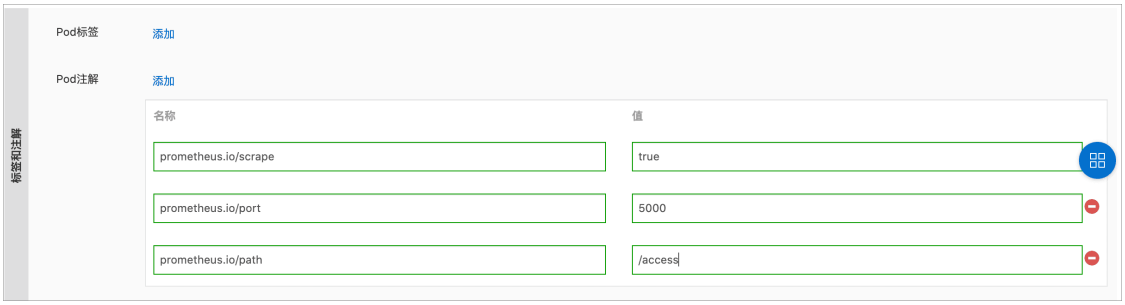
以镜像 `yejianhonghong/pindex` 为示例。



iv. 单击**下一步**。

v. 给Pod添加ARMS相关的Annotations。

*prometheus.io/port*表示Prometheus要scrape的endpoint端口，*prometheus.io/path*表示Prometheus要scrape的endpoint路径。



vi. 单击**创建**，完成应用创建任务。

6. 在**服务**页面创建服务。

i. 在集群管理页左侧导航栏中，选择**网络 > 服务**。

ii. 单击**服务**页面右上角的**创建**。

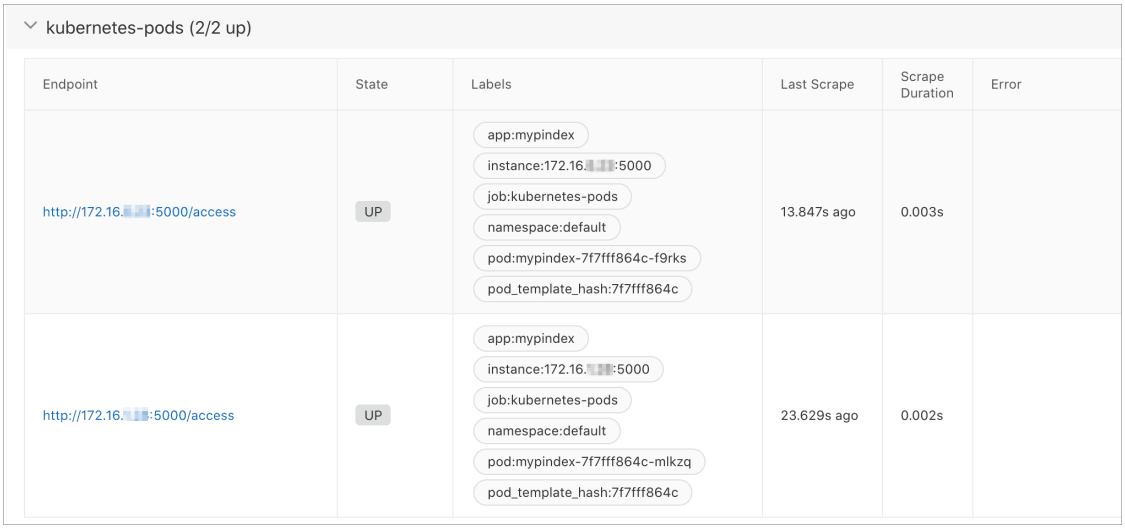
iii. 在**类型**中选择**负载均衡**型服务及**公网访问**。

iv. 在**关联**中选择步骤4创建的应用。

v. 单击**创建**，完成服务创建任务。

具体操作，请参见创建服务。

7. 配置自定义指标。
   i. 登录ARMS控制台。
   ii. 在控制台左侧导航栏单击**Prometheus监控**。
   iii. 在**Prometheus监控**页面，单击K8s列目标集群名称。
   iv. 在左侧导航栏单击**设置**， 然后在右侧区域单击**Targets(beta)**页签，可以看到自定义的指标已经配置成功。

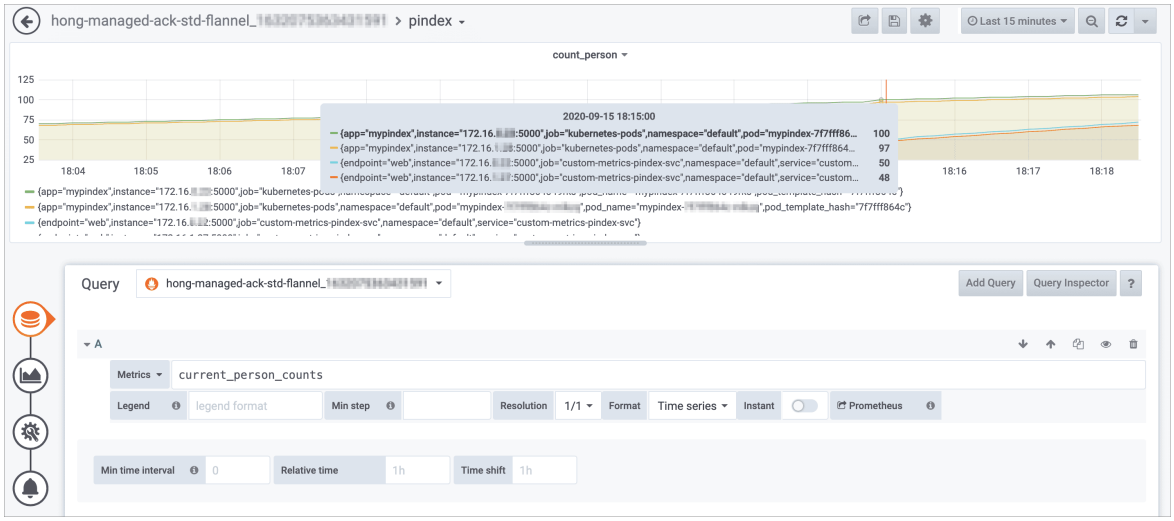


8. 访问步骤5所创建服务暴露出来的公网IP，增加指标值。

   具体指标的配置，请参见DATA MODEL。



9. 单击Prometheus任意一个预置的图表进入Grafana，然后单击**添加Panal**，选择**Graph**类型，将**Metrics**填写为*current_person_counts*。
10. 保存配置即可观察到自定义指标的Grafana图形。

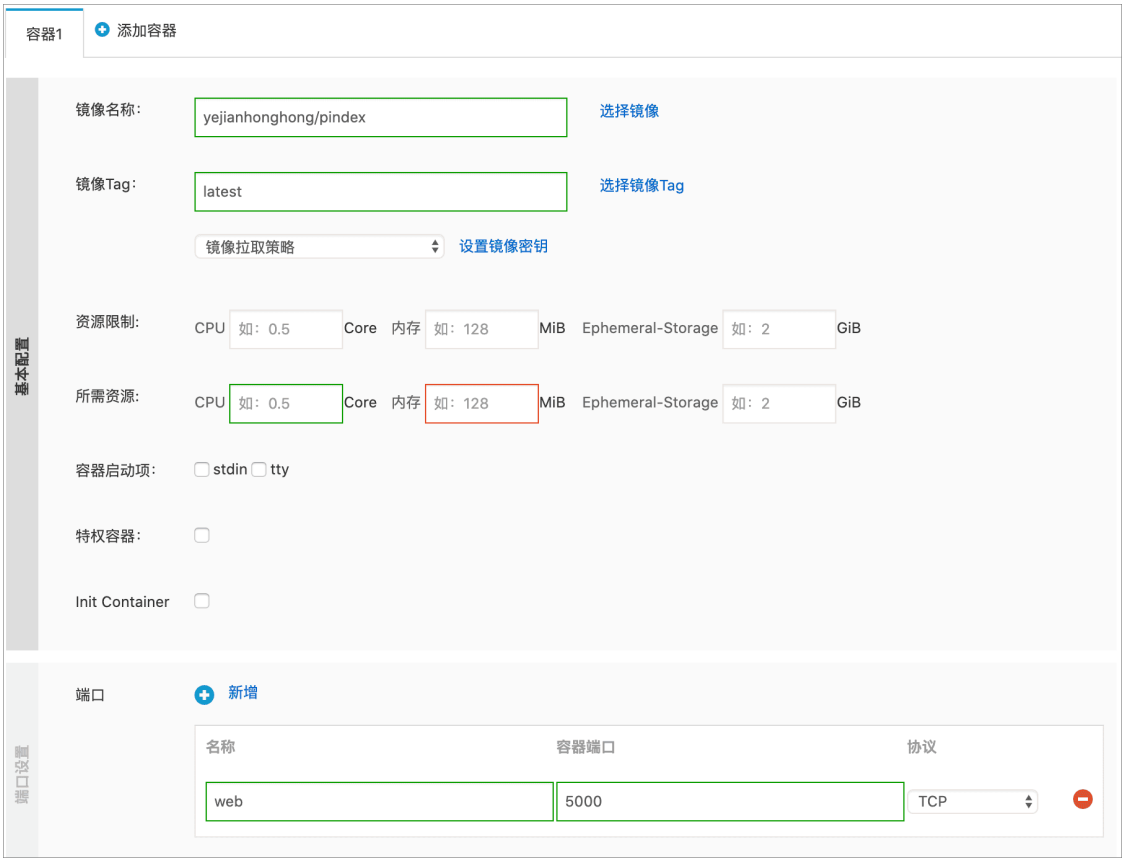


方式二：通过ServiceMonitor自定义指标监控

通过ServiceMonitor方式自定义指标监控在部署的时候不需要配置Annotations，但是要给Service对象添加标签。

1. 登录容器服务管理控制台。

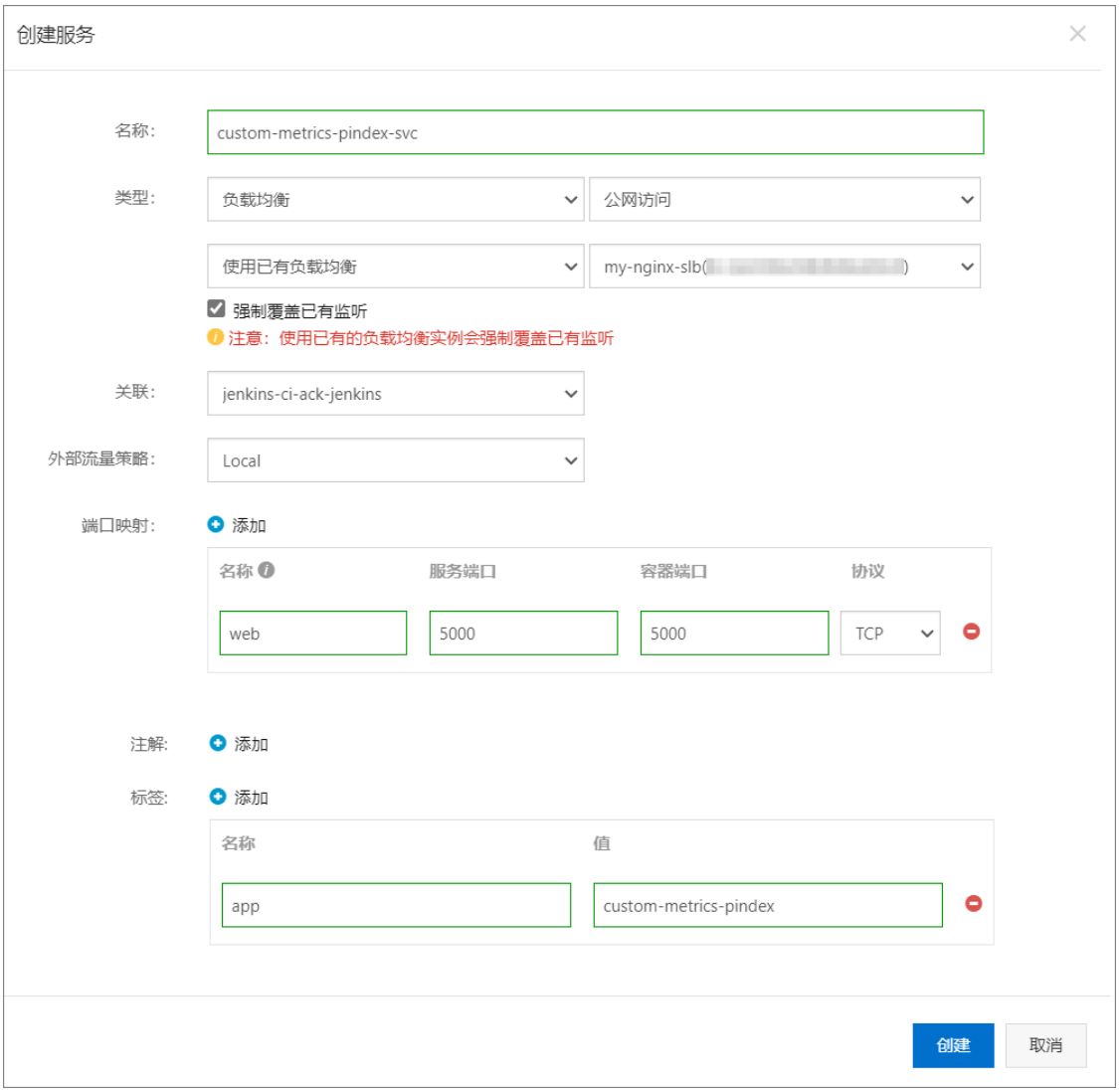
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**无状态**页面中，创建应用。
   i. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
   ii. 单击**使用镜像创建**。
   iii. 在配置向导**应用基本信息**中填写应用的基本信息，单击**下一步**。
   iv. 创建一个Web应用，暴露5000端口。
   以镜像 `yejianhonghong/pindex` 为示例。



   单击**下一步**。
   v. 单击**创建**，完成应用创建任务。
5. 在**服务**页面创建服务。
   i. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
   ii. 单击**服务**页面右上角的**创建**。
   iii. 在**类型**中选择**负载均衡**型服务及**公网访问**。
   iv. 在**关联**中选择步骤4创建的应用。

v. 添加标签。

该标签将被用于serviceMonitor的selector。



vi. 单击**创建**，完成服务创建任务。

具体操作，请参见创建服务。

6. 使Prometheus获得服务Scape Endpioint。

i. 登录Prometheus控制台。

ii. 在控制台左侧导航栏选择**监控列表**，单击**K8s**列目标集群名称。

iii. 在左侧导航栏单击**设置**， 然后在右侧区域单击**服务发现**页签。

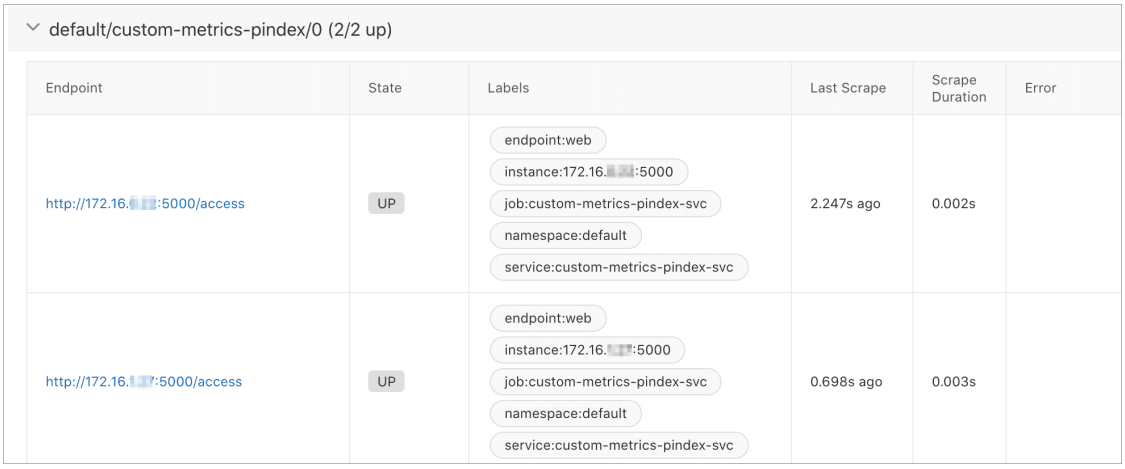
iv. 在**ServiceMonitor**页签右上角单击**添加ServiceMonitor**创建ServiceMonitor。

模板文件如下所示。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
 #填写一个唯一名称。
 name: custom-metrics-pindex
 #填写目标命名空间。
 namespace: default
spec:
 endpoints:
 - interval: 30s
  #填写service.yaml中对应的Port的Name字段的值（参考上图）。
  port: web
  #填写Service对应的Path的值。
  path: /access
 namespaceSelector:
  any: true
  #Nginx Demo的命名空间。
 selector:
  matchLabels:
   #填写service.yaml的Label字段的值以定位目标service.yaml。
   app: custom-metrics-pindex
```

单击**确定**完成ServiceMonitor创建。

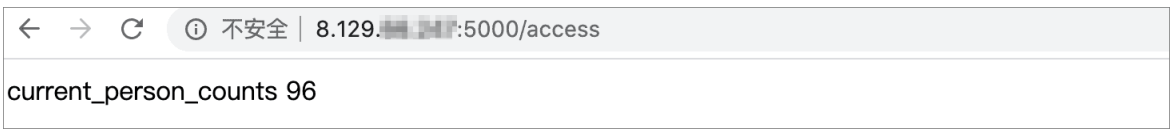
v. 在**Targets(beta)**页签，可见Prometheus已经获得服务Scape Endpioint。



> **说明** 这里ServiceMonitor定义比Annotations方式更直观，指示出了Namespace和Service名称。

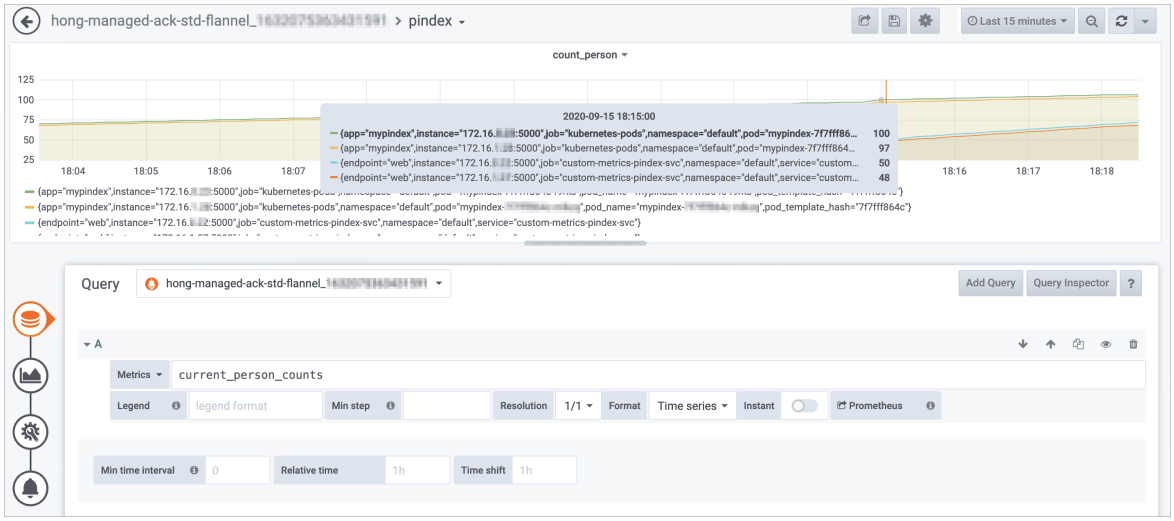
7. 访问步骤5创建的服务暴露出来的公网IP，增加指标值。

关于指标的配置，请参见DATA MODEL。



8. 单击Prometheus任意一个预置的图表进入Grafana，然后单击**添加Panal**，选择**Graph**类型，将**Metrics**填写为*current_person_counts*。

9. 保存配置即可观察到自定义指标的Grafana图形。



常见问题

删除阿里云Prometheus的命名空间后，重新安装阿里云Prometheus失败。

只删除阿里云Prometheus的命名空间，会导致资源删除后有残留配置，影响再次安装。您可以执行以下操作，删除残余配置。

1. 删除ClusterRole。

```
kubectl delete ClusterRole arms-kube-state-metrics
kubectl delete ClusterRole arms-node-exporter
kubectl delete ClusterRole arms-prom-ack-arms-prometheus-role
kubectl delete ClusterRole arms-prometheus-oper3
kubectl delete ClusterRole arms-prometheus-ack-arms-prometheus-role
kubectl delete ClusterRole arms-pilot-prom-k8s
```

2. 删除ClusterRoleBinding。

```
kubectl delete ClusterRoleBinding arms-node-exporter
kubectl delete ClusterRoleBinding arms-prom-ack-arms-prometheus-role-binding
kubectl delete ClusterRoleBinding arms-prometheus-oper-bind2
kubectl delete ClusterRoleBinding kube-state-metrics
kubectl delete ClusterRoleBinding arms-pilot-prom-k8s
kubectl delete ClusterRoleBinding arms-prometheus-ack-arms-prometheus-role-binding
```

3. 删除Role及RoleBinding。

```
kubectl delete Role arms-pilot-prom-spec-ns-k8s
kubectl delete Role arms-pilot-prom-spec-ns-k8s -n kube-system
kubectl delete RoleBinding arms-pilot-prom-spec-ns-k8s
kubectl delete RoleBinding arms-pilot-prom-spec-ns-k8s -n kube-system
```

相关文档

- 配置Prometheus监控采集规则

13.3.7. Ingress Dashboard监控

阿里云Ingress组件支持将您的所有HTTP请求日志记录到标准输出中。同时，阿里云打通Ingress组件访问日志服务与阿里云日志服务，从而您可以使用日志服务快速创建日志分析和监控大盘。本文介绍如何使用Ingress Dashboard与ARMS进行联动排查。

前提条件

实现Ingress Dashboard监控前，您需要先安装日志组件。以下为安装日志组件的两种方法：

- 如果您还没有创建集群，您可以在创建集群时，选中**使用日志服务**、**安装Ingress 组件**以及**创建Ingress Dashboard**。
- 如果您已创建集群，您需要在组件管理页面安装日志组件。更多信息，请参见通过日志服务采集Kubernetes容器日志。

步骤一：查看Ingress日志与报表

1. 登录日志服务控制台。
2. 在**Project列表**区域，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**日志库**页签下单击**nginx-ingress**左侧的 。

   > **说明** 名称为**nginx-ingress**的日志库（logstore）存放着所有的Ingress访问日志。

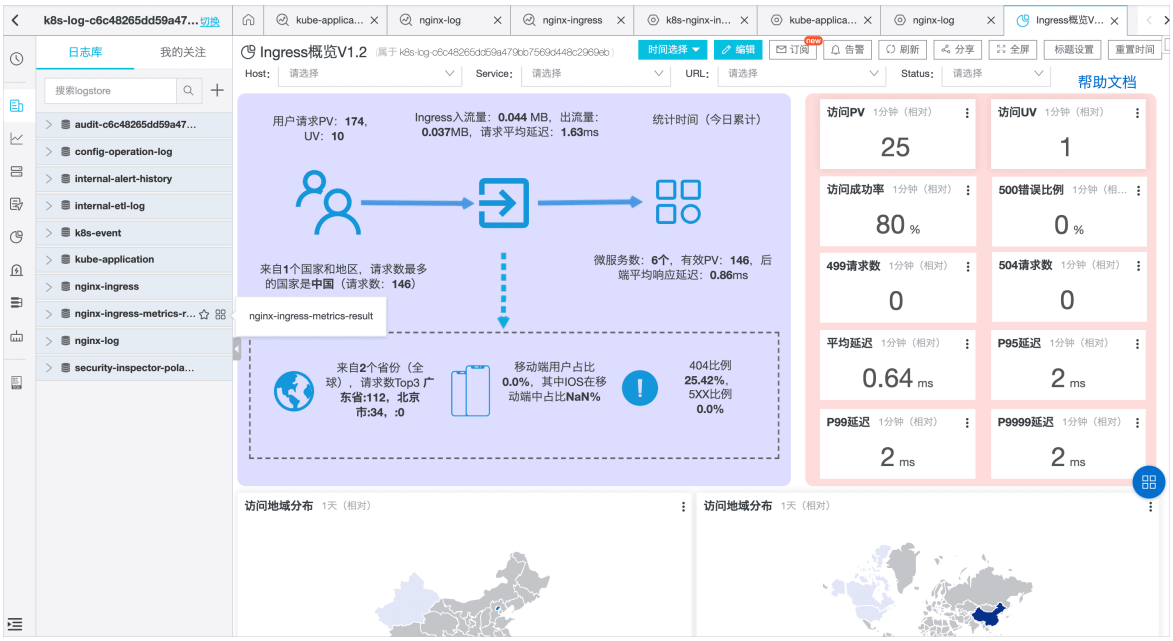
4. 单击**nginx-ingress**下的**可视化仪表盘**，查看所有Ingress的分析报表。

   Ingress Dashboard包含Ingress概览、Ingress异常检测中心、Ingress访问中心、Ingress蓝绿发布中心、Ingress监控中心5个预置图表。以下重点介绍Ingress概览和Ingress蓝绿发布中心。关于其他预置图表的详细信息请参见Ingress访问日志分析与监控。

   Ingress概览

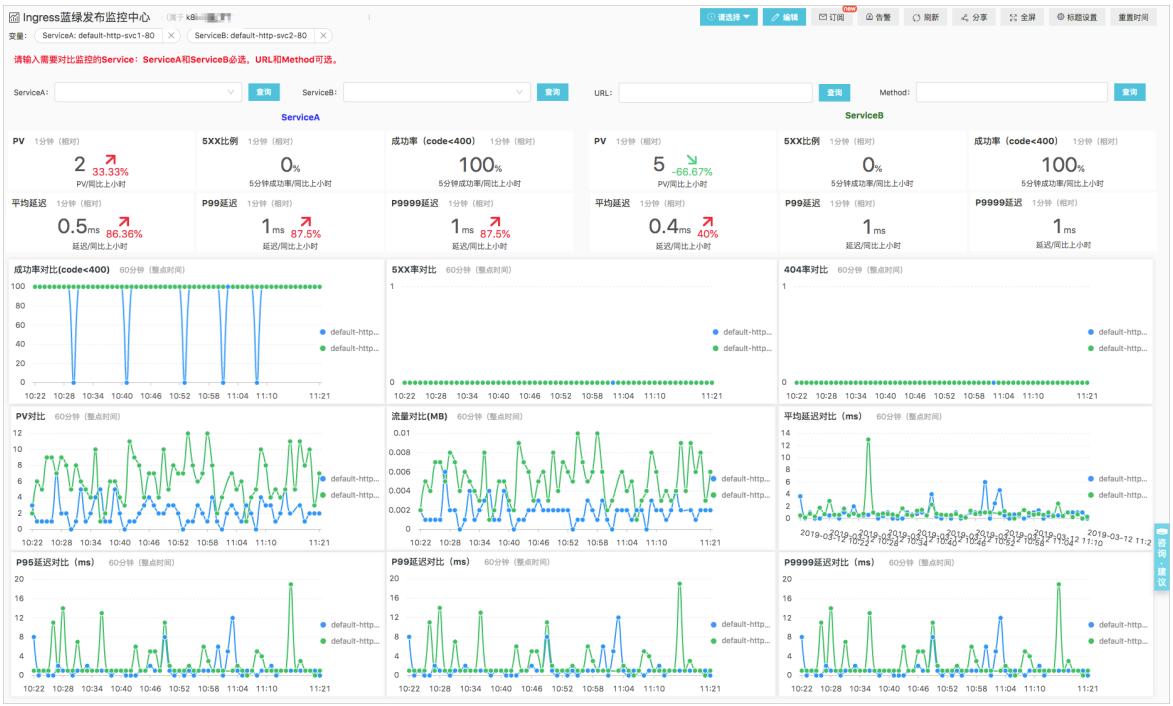
   Ingress概览报表主要展示当前Ingress的整体状态，主要包括以下几类信息：

   - 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。



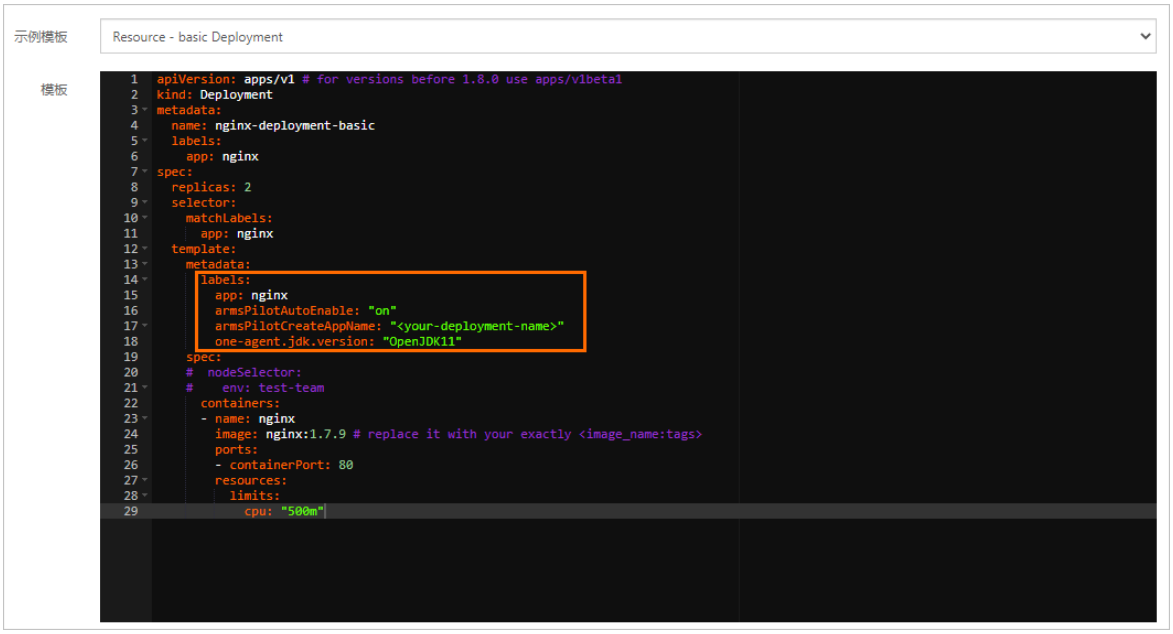
步骤二：为Java应用开启ARMS应用监控

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. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面右上角单击**使用YAML创建资源**。
6. 选择**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：

展开查看完整示例YAML文件（Java）

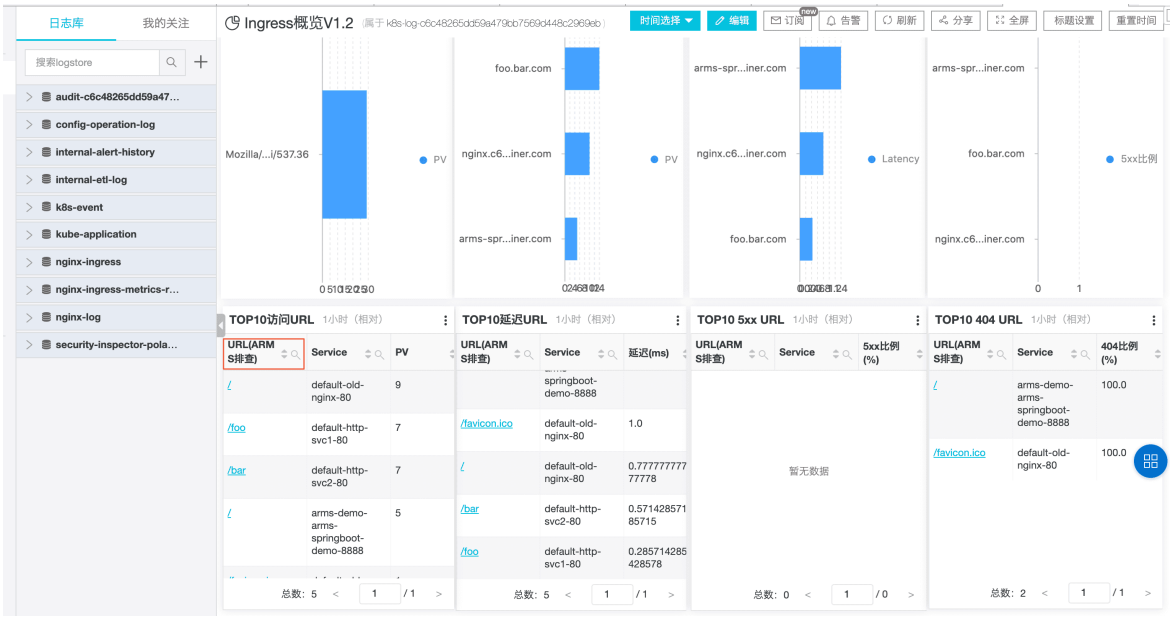
步骤三：为示例服务配置Ingress

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
5. 在**路由**页面单击**创建**。
6. 在**创建**对话框中设置路由，然后单击**创建**。

   关于设置路由的详细操作，请参见创建Ingress路由。

步骤四：使用Ingress Dashboard与ARMS进行联动排查

1. 登录日志服务控制台。
2. 在**Project列表**区域，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**可视化仪表盘**。
4. 在仪表盘列表中单击**Ingress概览V1.2**。
5. 在**Ingress概览V1.2**页面找到**Top10延迟URL**，查看TOP10延迟URL。
6. 在**Ingress概览V1.2**页面单击**URL（ARMS排查）**，跳转到对应服务的ARMS监控链路查询页面。您可以在ARMS控制台查询对应服务的查询调用链路详细情况。



13.3.8. 使用ACK Terway Hubble实现网络可观测性

在ACK集群中部署ACK Terway Hubble可以将容器网络流量、网络策略进行可视化展示，从而实现网络架构、业务拓扑关系的可观测性。本文介绍如何使用ACK Terway Hubble查看容器网络中网络流量的来源、目的地等数据，实现网络可观测性。

前提条件

创建Kubernetes托管版集群

> **说明** ACK Terway Hubble当前仅支持基于IPvlan的Terway ENI多IP网络模式。因此创建集群时必须设置**网络插件**为**Terway**，**Terway模式**为**IPvlan**，不然无法使用ACK Terway Hubble。

步骤一：设置Terway的配置文件eni-config

通过控制台设置Terway的配置文件eni-config

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 编辑Terway的配置文件eni-config。
   i. 在集群管理页左侧导航栏中，选择**配置管理 > 配置项**。
   ii. 在**配置项**页面顶部设置**命名空间**为kube-system，单击eni-config**操作**列下的**YAML编辑**。

iii. 在**查看YAML**面板找到`10-terway.conf`，在`10-terway.conf`下添加以下参数，然后单击**确定**。

```
"cilium_enable_hubble":"true",
"cilium_hubble_listen_address":":4244",
"cilium_hubble_metrics_server":":9091",
"cilium_hubble_metrics":"drop,tcp,flow,port-distribution,icmp",
```

参数	描述	备注
eniip_virtual_type	是否开启IPVLAN模式。	如果配置文件中此项不存在或不为`IPVLAN`，则您的集群不支持ACK Terway Hubble。
cilium_enable_hubble	启用Hubble网络流量分析。	本文设置为`"true"`。
cilium_hubble_listen_address	Hubble网络流量信息暴露地址。	本文设置为`":4244"`。
cilium_hubble_metrics_server	Hubble Metrics暴露的地址。	本文设置为`":9091"`。
cilium_hubble_metrics	Hubble需要采集的Metrics，以半角逗号（,）分割。	目前不支持DNS、HTTP这类L7的能力。Hubble支持采集的Metrics全量列表为`"drop,tcp,flow,port-distribution,icmp"`。 **说明** 如果您启用Metrics数量比较多，则会对Hubble性能产生一定影响。

5. 重启Terway容器组，使其配置文件生效。
   i. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。
   ii. 在**容器组**页面顶部设置**命名空间**为kube-system，在搜索框中搜索terway-eniip，单击terway-eniip-xxx**操作**列下的**删除**。
   iii. 在**删除容器组**对话框单击**确定**。
   在**容器组**页面terway-eniip-xxx**状态**列显示**Running**，说明重启容器组成功。
   iv. 重复执行以上步骤，删除所有Terway容器组。

通过命令行设置Terway的配置文件eni-config

1. 步骤二：选择集群凭证类型。
2. 编辑Terway的配置文件eni-config。
   i. 执行以下命令，编辑Terway的配置文件。

```
kubectl -n kube-system edit configmap eni-config
```

ii. 将以下内容复制到Terway的配置文件，然后保存。

```
"cilium_enable_hubble":"true",
"cilium_hubble_listen_address":":4244",
"cilium_hubble_metrics_server":":9091",
"cilium_hubble_metrics":"drop,tcp,flow,port-distribution,icmp",
```

参数	描述	备注
eniip_virtual_type	是否开启IPVLAN模式。	如果配置文件中此项不存在或不为`IPVLAN`，则您的集群不支持ACK Terway Hubble。
cilium_enable_hubble	启用Hubble网络流量分析。	本文设置为`"true"`。
cilium_hubble_listen_address	Hubble网络流量信息暴露地址。	本文设置为`":4244"`。
cilium_hubble_metrics_server	Hubble Metrics暴露的地址。	本文设置为`":9091"`。
cilium_hubble_metrics	Hubble需要采集的Metrics，以半角逗号（,）分割。	目前不支持DNS、HTTP这类L7的能力。Hubble支持采集的Metrics全量列表为`"drop,tcp,flow,port-distribution,icmp"`。 **说明** 如果您启用Metrics数量比较多，则会对Hubble性能产生一定影响。

3. 重启Terway容器组，使其配置文件生效。

   i. 执行以下命令，查看Terway容器组。

   ```
   kubectl -n kube-system get pod | grep terway-eniip
   ```

   ii. 执行以下命令，删除Terway容器组。

   ```
   kubectl -n kube-system delete pod terway-eniip-xxx
   ```

   其中terway-eniip-xxx需替换为Terway容器组名称，重复执行上述步骤，逐个删除容器组。

步骤二：安装ACK Terway Hubble

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面搜索ack-terway-hubble，找到并单击ack-terway-hubble。
4. 在右侧**创建**面板中，选择目标集群。
5. 在**应用目录-ack-terway-hubble**页面单击**参数**页签，在`ingress`参数下设置`hosts`，用于后续访问Hubble UI的界面，其他参数请根据实际情况设置。

   说明 以下参数均需要在`ingress`参数下配置。

参数	描述
enabled	是否开启Hubble-UI的Ingress访问。

参数	描述
annotations	自定义Ingress Annotations。
path	自定义Ingress根路径。
hosts	自定义Ingress Hosts。
tls	自定义Ingress TLS配置。

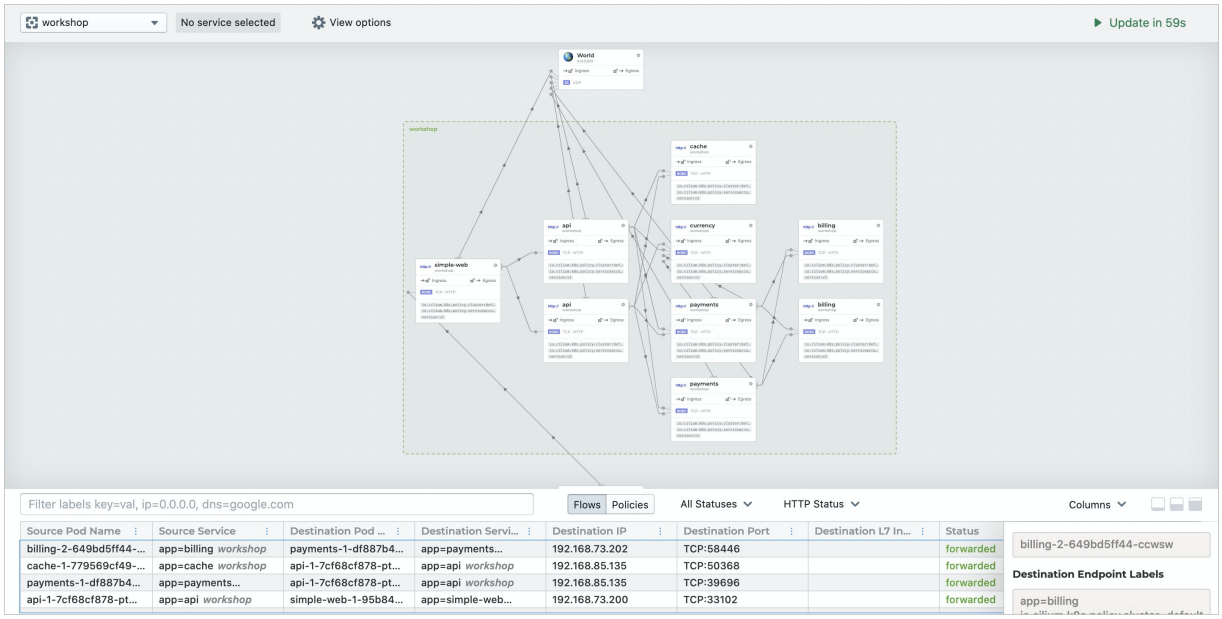
6. 在右侧**创建**面板中单击**创建**。

步骤三：使用ACK Terway Hubble

如果您在ACK Terway Hubble中配置了Ingress域名 **hosts** ，您可以通过该域名的80端口访问到Hubble UI的界面。访问该域名，您可以在界面上看到以下信息：

> **说明** 如果您使用了非权威解析的域名，例如ingress.local，您需要执行 `kubectl -n kube-system get svc nginx-ingress-lb` 命令，查看Hubble UI的IP地址，然后在本地hosts进行绑定，将ingress.local指向Ingress Hubble UI的IP地址。

- 在页面上半部分，您可以查看容器集群中各个命名空间中容器组与服务互访的拓扑结构。
- 在页面下半部分，您可以查看当前容器网络中网络流量的来源、目的地、端口信息以及转发状态。
- 如果您设置了网络策略，您可以查看哪些流量是被网络策略丢弃的。



ACK Terway Hubble可以通过kube-system命名空间下hubble-metrics服务暴露一系列网络流量的指标，具体暴露的指标由eni-config中 **cilium_hubble_metrics** 字段进行控制。您可以使用Prometheus和ARMS Prometheus采集Metrics数据。具体操作，请参见开源Prometheus监控和阿里云Prometheus监控。

> **说明** 更多ACK Terway Hubble暴露的指标，请参见hubble-exported-metrics 。

指标大类	指标名	指标字段	说明
drop	hubble_drop_total	reason，protocol	丢弃报文

指标大类	指标名	指标字段	说明
tcp	hubble_tcp_flags_total	flag，family	TCP标记
flow	hubble_flows_processed_total	type，subtype，verdict	网络流
port-distribution	hubble_port_distribution_total	protocol，port	目的端口报文分布
icmp	hubble_icmp_total	family，type	ICMP报文

13.3.9. 卸载ARMS Prometheus和ACK Prometheus

本文介绍如何卸载ARMS Prometheus和ACK Prometheus。

操作步骤

您可以通过控制台卸载ARMS Prometheus和ACK Prometheus，卸载后不会有数据残留。卸载ARMS Prometheus和ACK Prometheus的操作类似，以下以卸载ARMS Prometheus为例。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Helm**。
5. 在**Helm**页面找到**发布名称**为arms-prom的组件，然后单击arms-prom操作列的**删除**。

   > **说明** 如果您卸载的是ACK Prometheus，您需要在**Helm**页面找到**发布名称**为ack-prometheus-operator的组件，单击ack-prometheus-operator组件**操作**列的**删除**。

6. 在**删除应用**对话框中选中**清除发布记录**，然后单击**确定**。

13.4. 容器服务报警管理

报警配置功能提供统一管理容器报警场景的功能，包括容器服务异常事件报警、集群相关基础资源的关键指标报警、集群核心组件及集群中应用的指标报警。支持在创建集群时默认开启报警功能。容器服务的报警规则支持通过集群内部署CRD的方式配置管理。本文介绍容器服务报警中心功能使用场景、如何接入报警功能，以及专有集群如何授予报警功能资源的访问权限等。

背景信息

容器服务报警管理包含的功能：

- 集群异常事件报警，此类报警的数据链路同步自容器服务事件中心。更多信息，请参见事件监控。
- 集群相关基础资源的关键指标报警，此类报警的数据链路同步自云监控基础资源监控。更多信息，请参见基础资源监控。

功能使用场景

容器服务报警配置功能集合容器场景的监控报警能力，提供报警的统一配置管理，有如下几个典型的使用场景：

- **集群运维**
  可以通过监控报警第一时间了解集群管控、存储、网络、弹性扩缩容等异常事件。例如：
  - 通过**集群资源异常报警规则集**第一时间感知集群基础资源的关键指标是否异常。例如，CPU、Memory、网络等关键指标是否出现高水位情况，避免影响集群稳定性。
  - 通过配置并查看**集群异常事件报警规则集**感知集群节点或容器节点异常。例如，集群节点Docker进程异常、集群节点进程异常及集群容器副本启动失败等异常。

- 通过配置并查看**集群存储异常事件报警规则集**感知集群存储的变更与异常。
- 通过配置并查看**集群网络异常事件报警规则集**感知集群网络的变更与异常。
- 通过配置并查看**集群管控运维异常报警规则集**感知集群管控的变更与异常等。

- **应用开发**
  可以通过监控报警第一时间了解在集群中运行应用的异常事件、指标是否异常。例如，集群容器副本异常或者应用Deployment的CPU、内存水位指标是否超过阈值等。可通过开启报警配置功能中的默认报警规则模板，即可快速接受集群内应用容器副本的异常事件报警通知。例如，通过配置并订阅关注**集群容器副本异常报警规则集**感知所属应用的Pod是否异常。
- **应用管理**
  关注运行在集群上的应用健康、容量规划、集群运行稳定性及异常甚至是错误报警等贯穿应用生命周期的一系列问题。例如，通过配置并订阅关注**集群重要事件报警规则集**感知集群内所有Warning、Error等异常报警；关注**集群资源异常报警规则集**感知集群的资源情况，从而更好地做容量规划等。
- **多集群管理**
  当您有多个集群需要管理，为集群配置报警规则往往会是一个重复繁琐且难以同步的操作。容器服务报警配置功能，支持通过集群内部署**CRD配置的方式管理报警规则**。可通过维护多个集群中同样配置的CRD资源，来方便快捷地实现多集群中报警规则的同步配置。

组件安装与升级

控制台会自动检测报警配置环境是否符合要求，并会引导进行开通或安装、升级组件。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。
5. 在**报警配置**页面控制台会自动检查以下条件。

   若不符合条件，请按以下提示完成操作。

   - 已开通SLS日志服务云产品。当您首次使用日志服务时，需要登录日志服务控制台，根据页面提示开通日志服务。
   - 已安装**事件中心**。具体操作，请参见事件监控。
   - 集群托管组件alicloud-monitor-controller升级到最新版本。更多信息，请参见alicloud-monitor-controller。

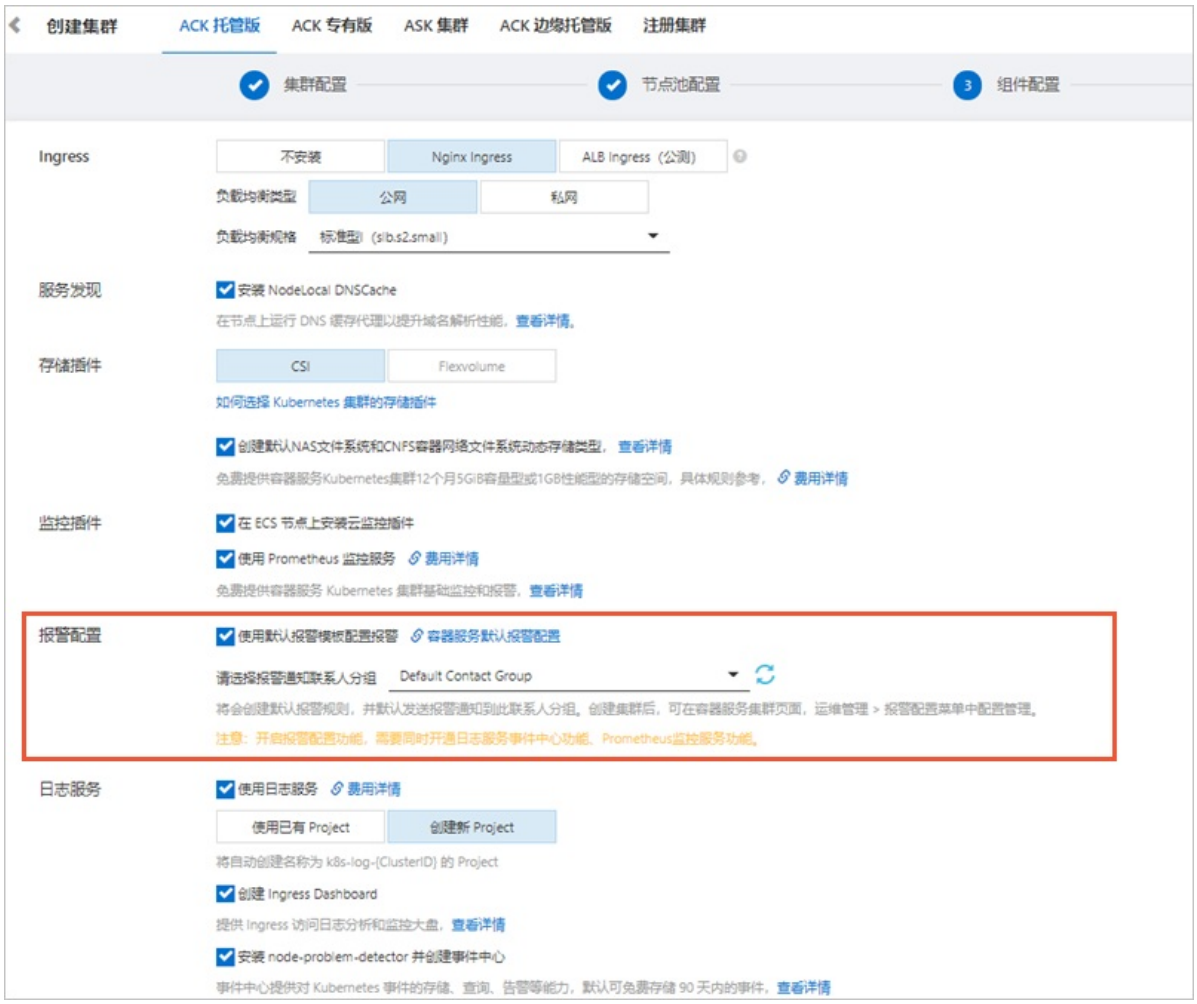


如何接入报警配置功能

目前容器服务报警规则配置功能支持ACK托管版集群、ACK专有版集群。

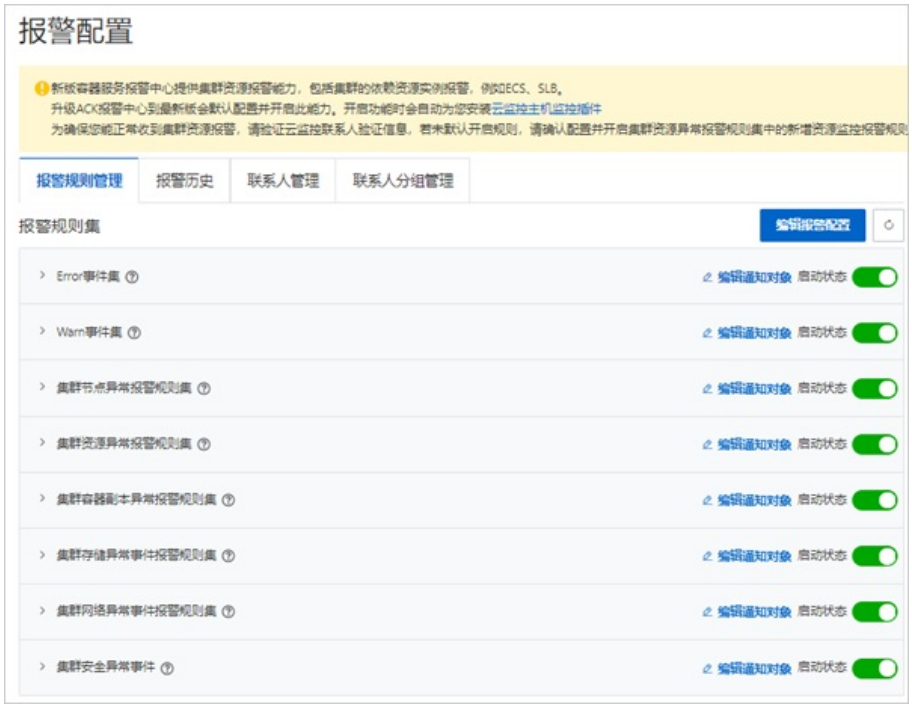
步骤一：开启默认报警规则

- 创建ACK托管版集群时，打开**使用默认报警模板配置报警**开关，并选择报警通知联系人分组。
  开启后将会创建默认报警规则，并默认发送报警通知到此联系人分组。



  具体操作，请参见创建Kubernetes托管版集群。

- 若已创建集群，可在目标集群开启对应报警规则。
  i. 在目标集群左侧导航栏选择**运维管理 > 报警配置**配置管理。
  ii. 在**报警规则管理**页签，打开**启动状态**可开启对应报警规则集。



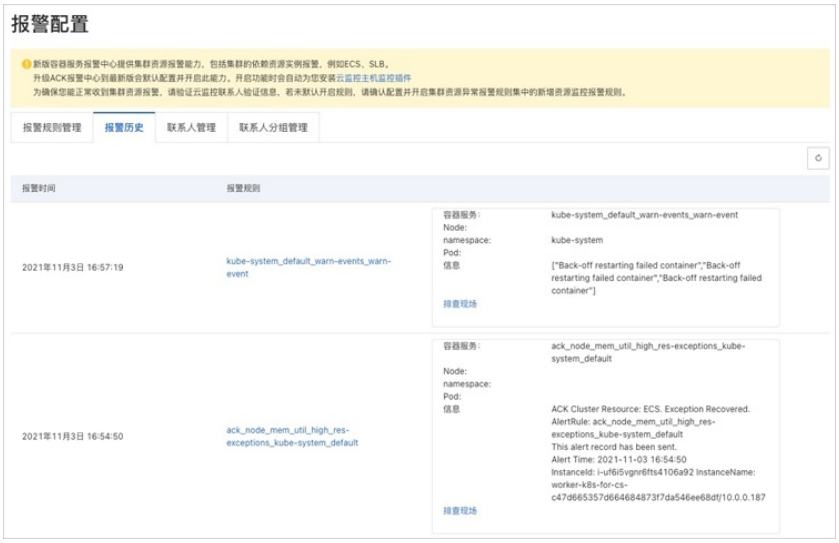
具体操作，请参见步骤二：手动配置报警规则。

步骤二：手动配置报警规则

ACK托管版集群、ACK专有版集群创建后，可进行报警规则、联系人和联系人分组管理。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。

功能特性	说明
报警规则管理	○ 容器服务报警规则功能会默认生成容器场景下的报警模板（包含异常事件报警、异常指标报警）。 ○ 报警规则被分类为若干个报警规则集，可为报警规则集关联多个联系人分组，并启动或关闭报警规则集。 ○ 报警规则集中包含多个报警规则，一个报警规则对应单个异常的检查项。多个报警规则集可以通过一个YAML资源配置到对应集群中，修改YAML会同步生成报警规则。 ○ 关于报警规则YAML配置，请参见如何通过CRD配置报警规则。 ○ 关于默认报警规则模板，请参见默认报警规则模板。

功能特性	说明
报警历史	目前可查看最近发送的近100条历史记录。单击对应报警**报警规则类型**的链接，可跳转到对应监控系统中查看详细规则配置；单击对应报警**排查现场**的链接可快速定位到异常发生的资源页面（异常事件、指标异常的资源）。 
联系人管理	对联系人进行管理，可创建、编辑或删除联系人。 集群资源异常报警规则集中集群节点基础资源报警，联系人短信、邮箱等联系方式需要在云监控中先验证联系方式，才能正常接收集群基础资源报警信息。可在云监控控制台查看同步联系人，若验证信息过期，可删除云监控中对应联系人并重新刷新容器服务报警中心的联系人页面。 
联系人分组管理	对联系人分组进行管理，可创建、编辑或删除联系人分组。当无联系人分组时，控制台会从您的阿里云账号注册信息中同步创建一个默认联系人分组。

5. 在**报警规则管理**页签，单击**编辑通知对象**可设置关联的通知对象；打开**启动状态**可开启对应报警规则集。

如何通过CRD配置报警规则

报警配置功能开启时，会默认在kube-system Namespace下创建一个AckAlertRule类型的资源配置，包含默认报警规则模板。容器服务报警规则集可通过此资源配置在集群中。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。
5. 在**报警规则管理**页签中，单击右上角**编辑报警配置**，可查看当前集群中的AckAlertRule资源配置，并可通过YAML文件修改。

   报警规则配置的YAML文件示例如下：

```
apiVersion: alert.alibabacloud.com/v1beta1
kind: AckAlertRule
metadata:
 name: default
spec:
 groups:
  #以下是一个集群事件报警规则配置样例。
  - name: pod-exceptions                #报警规则分组名，对应报警模板中的Group_Name字段。
    rules:
     - name: pod-oom                    #报警规则名。
       type: event                      #报警规则类型(Rule_Type)，枚举值为event（事件类型）、metric-cms（云监控指标类型）。
       expression: sls.app.ack.pod.oom  #报警规则表达式，当规则类型为event时，表达式的值为本文默认报警规则模板中Rule_Expression_Id值。
       enable: enable                   #报警规则开启状态，枚举值为enable、disable。
     - name: pod-failed
       type: event
       expression: sls.app.ack.pod.failed
       enable: enable
  #以下是一个集群基础资源报警规则配置样例。
  - name: res-exceptions                #报警规则分组名，对应报警模板中的Group_Name字段。
    rules:
     - name: node_cpu_util_high         #报警规则名。
       type: metric-cms                 #报警规则类型(Rule_Type)，枚举值为event（事件类型）、metric-cms（云监控指标类型）。
       expression: cms.host.cpu.utilization  #报警规则表达式，当规则类型为metric-cms时，表达式的值为本文默认报警规则模板中Rule_Expression_Id值。
       contactGroups:                   #报警规则映射的联系人分组配置，由ACK控制台生成，同一个账号下联系人相同，可在多集群中复用。
       enable: enable                   #报警规则开启状态，枚举值为enable、disable。
       thresholds:                      #报警规则阈值，详情见文档如何更改报警规则阈值部分。
        - key: CMS_ESCALATIONS_CRITICAL_Threshold
          unit: percent
          value: '1'
```

默认报警规则模板

在以下情况下ACK会默认创建相应报警规则：

- 开启默认报警规则功能。
- 未开启默认报警规则，首次进入报警规则页面。

默认创建的报警规则如下表所示。

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
	集群Error事件	集群中所有Error Level异常事件触发该报警。	event	error-event	sls.app.ack.error
critical_events集					

critical-events集群重要事件报警规则集类型规则集	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
	集群Warn事件	集群中关键Warn Level异常事件触发该报警，排除部分可忽略事件。	event	warn-event	sls.app.ack.warn
cluster-error集群异常事件报警规则集	集群节点Docker进程异常	集群中节点Dockerd或Containerd 运行时异常。	event	docker-hang	sls.app.ack.docker.hang
	集群驱逐事件	集群中发生驱逐事件。	event	eviction-event	sls.app.ack.eviction
	集群GPU的XID错误事件	集群中GPU XID异常事件。	event	gpu-xid-error	sls.app.ack.gpu.xid_error
	集群节点下线	集群中节点下线。	event	node-down	sls.app.ack.node.down
	集群节点重启	集群中节点重启。	event	node-restart	sls.app.ack.node.restart
	集群节点时间服务异常	集群中节点时间同步系统服务异常。	event	node-ntp-down	sls.app.ack.ntp.down
	集群节点PLEG异常	集群中节点PLEG异常。	event	node-pleg-error	sls.app.ack.node.pleg_error
	集群节点进程异常	集群中节点进程数异常。	event	ps-hang	sls.app.ack.ps.hang
	集群节点-CPU使用率≥85%	集群中节点实例CPU使用率超过水位。默认值85%。 在剩余资源不足15%时，Kubelet会发生强制驱逐行为。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	node_cpu_util_high	cms.host.cpu.utilization

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
	集群节点-内存使用率≥85%	集群中节点实例内存使用率超过水位。默认值85%。 在剩余资源不足15%时，Kubelet会发生强制驱逐行为。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	node_mem_util_high	cms.host.memory.utilization
	集群节点-磁盘使用率≥85%	集群中节点实例磁盘使用率超过水位。默认值85%。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	node_disk_util_high	cms.host.disk.utilization
	集群节点-公网流出带宽使用率≥85%	集群中节点实例公网流出带宽使用率超过水位。默认值85%。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	node_public_net_util_high	cms.host.public.network.utilization
	集群节点-inode使用率≥85%	集群中节点实例inode使用率超过水位。默认值85%。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	node_fs_inode_util_high	cms.host.fs.inode.utilization

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
res-exceptions 集群资源异常报警规则集	集群资源-负载均衡最大连接数使用率≥85%	集群中负载均衡实例最大连接数超过水位。默认值85%。 **说明** 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	slb_qps_util_high	cms.slb.qps.utilization
	集群资源-负载均衡网络流出带宽使用率≥85%	集群中负载均衡实例网络流出带宽使用率超过水位。默认值85%。 **说明** 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	slb_traff_tx_util_high	cms.slb.traffic.tx.utilization

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
	集群资源-负载均衡最大连接数使用率≥85%	集群中负载均衡实例最大连接数使用率超过水位。默认值85%。 说明 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	slb_max_con_util_high	cms.slb.max.connection.utilization
	集群资源-负载均衡监听每秒丢失连接数持续≥1	集群中负载均衡实例每秒丢失连接数持续超过水位。默认值1次。 说明 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见如何修改集群基础资源报警规则的阈值。	metric-cms	slb_drop_con_high	cms.slb.drop.connection
	集群节点文件句柄过多	集群中节点文件句柄数过多异常。	event	node-fd-pressure	sls.app.ack.node.fd_pressure
	集群节点磁盘空间不足	集群中节点磁盘空间不足异常事件。	event	node-disk-pressure	sls.app.ack.node.disk_pressure
	集群节点进程数过多	集群中节点进程数过多异常事件。	event	node-pid-pressure	sls.app.ack.node.pid_pressure
	集群节点调度资源不足	集群中无调度资源异常事件。	event	node-res-insufficient	sls.app.ack.resource.insufficient

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
	集群节点IP资源不足	集群中IP资源不足异常事件。	event	node-ip-pressure	sls.app.ack.ip.not_enough
pod-exceptions 集群容器副本异常报警规则集	集群容器副本OOM	集群容器副本Pod或其中进程出现OOM（Out of Memory）。	event	pod-oom	sls.app.ack.pod.oom
	集群容器副本启动失败	集群容器副本Pod启动失败事件（Pod Start Failed）。	event	pod-failed	sls.app.ack.pod.failed
	集群镜像拉取失败事件	集群容器副本Pod出现镜像拉取失败事件。	event	image-pull-back-off	sls.app.ack.image.pull_back_off
	无可用LoadBalancer	集群无法创建LoadBalancer事件。请提交工单联系容器服务团队。	event	slb-no-ava	sls.app.ack.ccm.no_ava_slb
	同步LoadBalancer失败	集群创建LoadBalancer同步失败事件。请提交工单联系容器服务团队。	event	slb-sync-err	sls.app.ack.ccm.sync_slb_failed
	删除LoadBalancer失败	集群删除LoadBalancer失败事件。请提交工单联系容器服务团队。	event	slb-del-err	sls.app.ack.ccm.del_slb_failed
	删除节点失败	集群删除节点失败事件。请提交工单联系容器服务团队。	event	node-del-err	sls.app.ack.ccm.del_node_failed
	添加节点失败	集群添加节点失败事件。请提交工单联系容器服务团队。	event	node-add-err	sls.app.ack.ccm.add_node_failed
	创建VPC网络路由失败	集群创建VPC网络路由失败事件。请提交工单联系容器服务团队。	event	route-create-err	sls.app.ack.ccm.create_route_failed

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-ops-err 集群管控运维异常报警规则集	同步VPC网络路由失败	集群同步VPC网络路由失败事件。请提交工单联系容器服务团队。	event	route-sync-err	sls.app.ack.ccm.sync_route_failed
	托管节点池命令执行失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-run-cmd-err	sls.app.ack.nlc.run_command_fail
	托管节点池未提供任务的具体命令	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-empty-cmd	sls.app.ack.nlc.empty_task_cmd
	托管节点池出现未实现的任务模式	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-url-m-unimp	sls.app.ack.nlc.url_mode_unimpl
	托管节点池发生未知的修复操作	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-opt-no-found	sls.app.ack.nlc.op_not_found
	托管节点池销毁节点发生错误	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-des-node-err	sls.app.ack.nlc.destroy_node_fail
	托管节点池节点排水失败	集群托管节点池排水异常事件。请提交工单联系容器服务团队。	event	nlc-drain-node-err	sls.app.ack.nlc.drain_node_fail
	托管节点池重启ECS未达到终态	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-restart-ecs-wait	sls.app.ack.nlc.restart_ecs_wait_fail
	托管节点池重启ECS失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-restart-ecs-err	sls.app.ack.nlc.restart_ecs_fail
	托管节点池重置ECS失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-reset-ecs-err	sls.app.ack.nlc.reset_ecs_fail
	托管节点池自愈任务失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-sel-repair-err	sls.app.ack.nlc.repair_fail

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-network-err集群网络异常事件报警规则集	Terway资源无效	集群Terway网络资源无效异常事件。请提交工单联系容器服务团队。	event	terway-invalid-res	sls.app.ack.terway.invalid_resource
	Terway分配IP失败	集群Terway网络资源分配IP失败异常事件。请提交工单联系容器服务团队。	event	terway-alloc-ip-err	sls.app.ack.terway.alloc_ip_fail
	解析Ingress带宽配置失败	集群Ingress网络解析配置异常事件。请提交工单联系容器服务团队。	event	terway-parse-err	sls.app.ack.terway.parse_fail
	Terway分配网络资源失败	集群Terway网络资源分配失败异常事件。请提交工单联系容器服务团队。	event	terway-alloc-res-err	sls.app.ack.terway.allocate_failure
	Terway回收网络资源失败	集群Terway网络资源回收失败异常事件。请提交工单联系容器服务团队。	event	terway-dispose-err	sls.app.ack.terway.dispose_failure
	Terway虚拟模式变更	集群Terway网络虚拟模式变更事件。	event	terway-virt-mod-err	sls.app.ack.terway.virtual_mode_change
	Terway触发PodIP配置检查	集群Terway网络触发PodIP配置检查事件。	event	terway-ip-check	sls.app.ack.terway.config_check
	Ingress重载配置失败	集群Ingress网络配置重载异常事件。请检查Ingress配置是否正确。	event	ingress-reload-err	sls.app.ack.ingress.err_reload_nginx
	云盘容量少于20 GiB限制	集群网盘固定限制，无法挂载小于20 GiB的磁盘。请检查所挂载云盘的容量大小。	event	csi_invalid_size	sls.app.ack.csi.invalid_disk_size

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-storage-err集群存储异常事件报警规则集	容器数据卷暂不支持包年包月类型云盘	集群网盘固定限制，无法挂载包年包月类型的云盘。请检查所挂载云盘的售卖方式。	event	csi_not_portable	sls.app.ack.csi.disk_not_portable
	挂载点正在被进程占用，卸载挂载点失败	集群存储挂载点正在被进程占用，卸载挂载点失败。	event	csi_device_busy	sls.app.ack.csi.deivce_busy
	无可用云盘	集群存储挂载时无可用云盘异常。请提交工单联系容器服务团队。	event	csi_no_ava_disk	sls.app.ack.csi.no_ava_disk
	云盘IOHang	集群出现IOHang异常。请提交工单联系容器服务团队。	event	csi_disk_iohang	sls.app.ack.csi.disk_iohang
	磁盘绑定的PVC发生slowIO	集群磁盘绑定的PVC发生slowIO异常。请提交工单联系容器服务团队。	event	csi_latency_high	sls.app.ack.csi.latency_too_high
	磁盘容量超过水位阈值	集群磁盘使用量超过水位值异常。请检查你的集群磁盘水位情况。	event	disk_space_press	sls.app.ack.csi.no_enough_disk_space
security-err集群安全异常事件	安全巡检发现高危风险配置	集群安全巡检发现高危风险配置事件。请提交工单联系容器服务团队。	event	si-c-a-risk	sls.app.ack.si.config_audit_high_risk

如何为专有版集群授予报警功能访问权限

专有版集群在使用报警规则功能之前，需要手动添加权限。

> **说明** 托管版集群已自动添加SLS报警功能资源的访问权限。

为专有版集群SLS报警功能及ARMS-Prometheus报警功能授予资源访问权限。更多信息，请参见RAM自定义授权场景及访问控制概述。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群信息**页面，单击**集群资源**页签**Worker RAM角色**字段右侧的链接，进入**RAM访问控制**控制台。

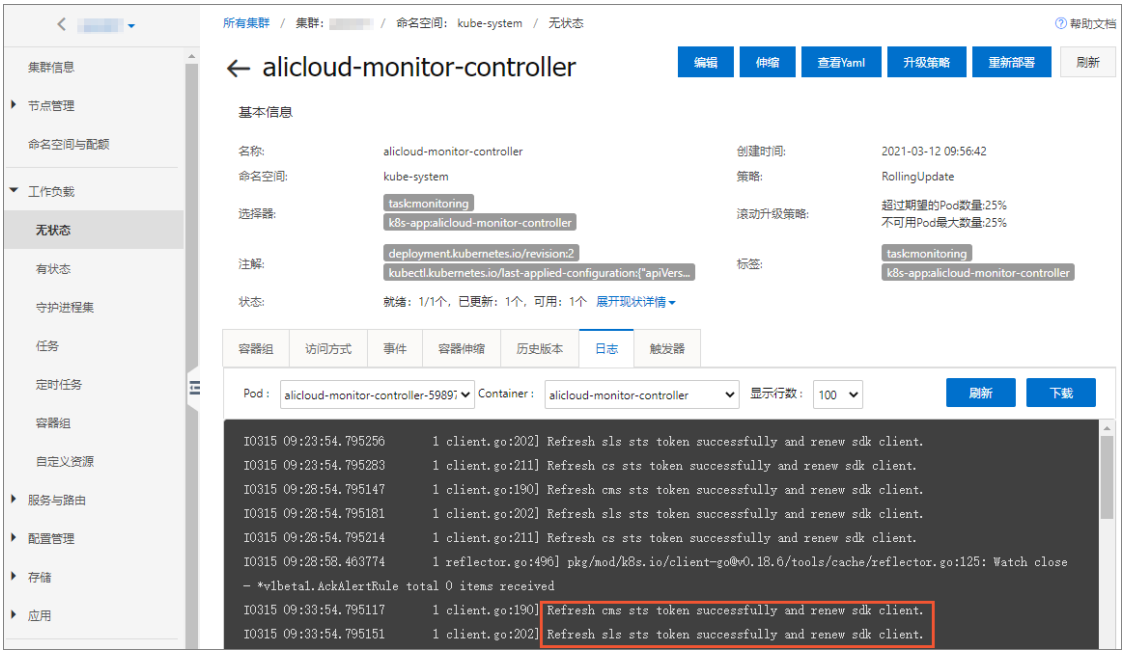


5. 在**RAM角色管理**页面的**权限管理**页签，单击对应**权限策略名称**的链接。
6. 在**策略内容**页签单击**修改策略内容**，并在右侧的**修改策略内容**面板将以下内容添加到策略内容中。

```
{
      "Action": [
        "log:*",
        "arms:*",
        "cms:*",
        "cs:UpdateContactGroup"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
}
```

7. 单击**确定**完成配置。
8. 通过日志查看报警功能访问权限是否已配置。
   i. 在容器服务管理控制台目标集群管理页左侧导航栏，选择**工作负载 > 无状态**。
   ii. 选择**命名空间**kube-system，单击无状态应用列表中alicloud-monitor-controller的**名称**链接。

iii. 单击**日志**页签，可看到授权成功的Pod日志。



如何修改集群基础资源报警规则的阈值

Rule_Type为metric-cms的报警规则，是同步自云监控的基础资源报警规则，此类规则可通过CRD配置报警规则的阈值。具体操作，请参见如何通过CRD配置报警规则。

本示例通过集群节点-CPU使用率规则的报警规则CRD，增加 `thresholds` 参数配置基础监控报警规则的阈值、重试判断次数、静默期配置。

```
apiVersion: alert.alibabacloud.com/v1beta1
kind: AckAlertRule
metadata:
 name: default
spec:
 groups:
  #以下是一个集群基础资源报警规则配置样例。
  - name: res-exceptions                    #报警规则分组名，对应报警模板中的Group_Name字段。
    rules:
     - name: node_cpu_util_high              #报警规则名。
       type: metric-cms                      #报警规则类型(Rule_Type)，枚举值为event（事件类型）、metric-cms（云监控指标类型）。
       expression: cms.host.cpu.utilization  #报警规则表达式，当规则类型为metric-cms时，表达式的值为本文默认报警规则模板中Rule_Expression_Id值。
       contactGroups:                        #报警规则映射的联系人分组配置，由ACK控制台生成，同一个账号下联系人相同，可在多集群中复用。
       enable: enable                        #报警规则开启状态，枚举值为enable、disable。
       thresholds:                           #报警规则阈值，更多信息，请参见如何通过CRD配置报警规则。
        - key: CMS_ESCALATIONS_CRITICAL_Threshold
          unit: percent
          value: '1'
        - key: CMS_ESCALATIONS_CRITICAL_Times
          value: '3'
        - key: CMS_RULE_SILENCE_SEC
          value: '900'
```

参数名	说明	默认值
CMS_ESCALATIONS_CRITICAL_Threshold	报警配置阈值。 • unit：表示单位，可配置为*percent*、*count*或*qps*。 • value：阈值。 此参数必填，未配置时规则将同步失败并关闭。	根据默认报警模板配置而定。
CMS_ESCALATIONS_CRITICAL_Times	云监控规则重试判断次数。 可选参数，未配置时取默认值。	3
CMS_RULE_SILENCE_SEC	云监控持续发送异常触发规则时，初次上报报警后静默期时长（单位：秒），防止过于频繁报警。 可选参数，未配置时取默认值。	900

常见问题

报警规则同步失败且报错信息为The Project does not exist：k8s-log-xxx

问题现象：

当报警中心中报警规则同步状态出现如下情况：



问题原因：

未创建SLS事件中心资源。

解决方案：

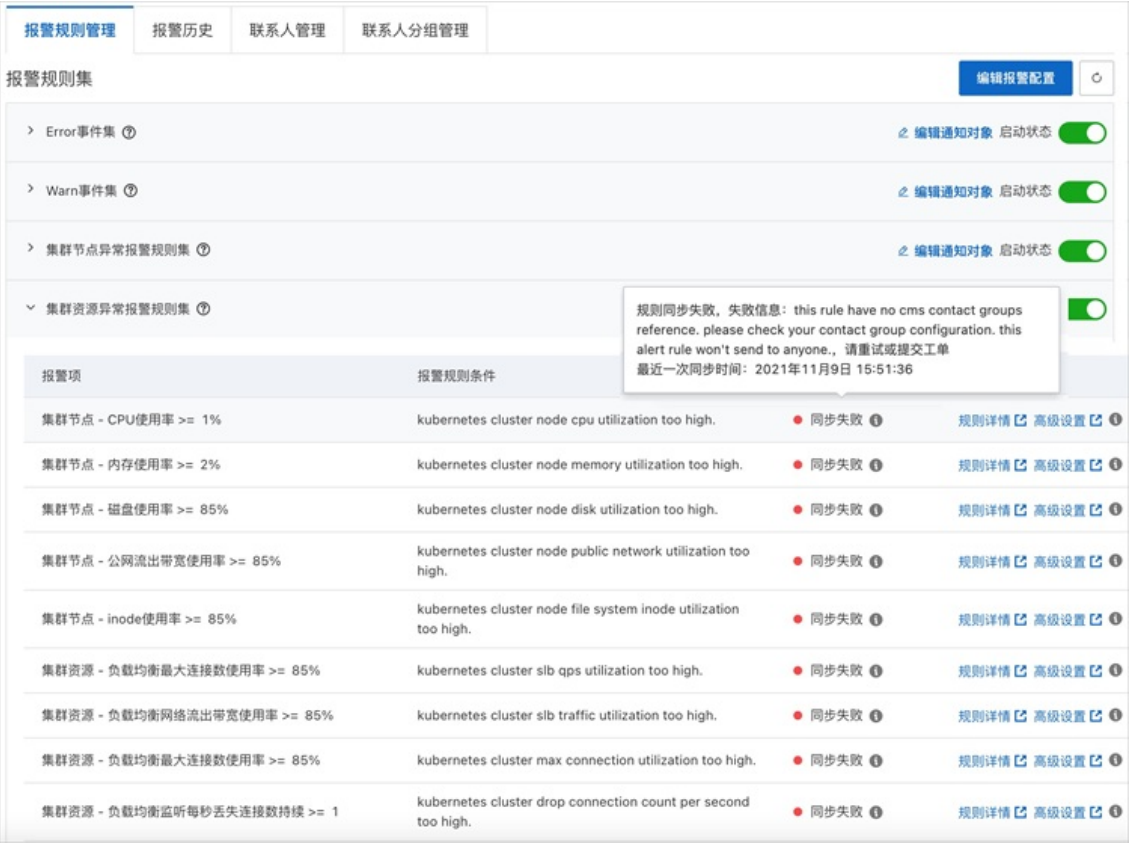
1. 在日志服务管理控制台检查Project是否达到Quota上限，删除多余的Project，或提交工单申请扩大Project资源Quota限制。关于如何删除Project，请参见管理Project。
2. 重新安装ack-node-problem-detector组件。
   i. 在容器服务管理控制台目标集群管理页左侧导航栏中，选择**应用 > Helm**。
   ii. 若您需要通过YAML方式重新安装ack-node-problem-detector组件，请通过以下操作备份ack-node-problem-detector组件资源。
   在**Helm**页面，单击ack-node-problem-detector组件右侧**操作**列的**更新**。更新完成后，单击ack-node-problem-detector组件右侧**操作**列**详情**。在**ack-node-problem-detector**组件详情页，单击所有资源信息的**查看YAML**，并保存各资源的YAML到本地。
   iii. 在**Helm**页面，单击ack-node-problem-detector组件右侧**操作**列的**删除**。
   iv. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
   v. 单击**日志与监控**页签，在ack-node-problem-detector组件的卡片中单击**安装**。
   在**提示**对话框中确认版本信息后单击**确定**。安装成功后，对应组件卡片区域会提示已安装，且可查看组件当前版本。

由于无订阅的联系人组导致报警规则同步失败

问题现象：

当报警中心中报警规则同步状态出现如下情况：

报错信息为类似信息：`this rule have no xxx contact groups reference` 。



问题原因：

报警规则无订阅的联系人组。

解决方案：

1. 已创建联系人，并将联系人加入联系人分组中。
2. 在对应报警规则集右侧单击**编辑通知对象**，为该组报警规则配置订阅的联系人分组。

关于上述操作的详情，请参见如何接入报警配置功能。

13.5. 可观测性FAQ

本文介绍在使用可观测性时遇到的常见问题的解决方法。

Prometheus数据异常无法显示

问题现象：

Prometheus数据异常无法显示。

问题原因：

在同一地域下创建过同名集群，且该同名集群已删除，但Prometheus监控插件没有同步删除，导致同名新集群的Prometheus监控插件安装失败。

解决方法：

1. 在ARMS控制台中卸载旧同名集群的Prometheus监控插件。具体操作，请参见卸载Prometheus监控插件。
2. 在新同名集群中安装Prometheus监控插件。具体操作，请参见开启阿里云Prometheus监控。

14.成本分析

14.1. 集群成本分析

ACK的成本分析功能可以协助企业IT管理员了解集群资源使用量及成本分布，提升集群资源利用率，提供成本节约建议。本文介绍如何配置集群成本分析及集群成本分析功能介绍。

前提条件

- 您已创建1.18.8及以上版本的Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- Kubernetes集群已开启阿里云Prometheus监控。具体操作，请参见开启阿里云Prometheus监控。
- 使用成本分析功能前，请提交工单申请添加白名单。

功能介绍

容器服务成本分析主要包含以下几个重要的功能：

- 云资源费用分析
  将Kubernetes集群下所有的云资源费用进行聚合，支持查看集群整体费用的同比与环比。
- 集群费用趋势分析
  查看Kubernetes集群下所有云资源的消费趋势，分析热点云产品消费，协助成本节约决策。
- 集群成本节约推荐
  将Kubernetes集群的费用趋势进行分析，提供资源用量和成本消费的相关性分析，提供成本节约建议。
- 实时价格模拟预测
  将集群中计算资源的实时价格进行模拟，提供小时维度的成本预估，支持包年包月、按量付费、竞价付费等多种付费方式。
- 命名空间资源分账
  根据命名空间拆分集群成本比例，提供实时成本预估、成本趋势变化、集群成本占比等能力。
- 应用成本优化
  在命名空间下查看Top 10的成本消耗最多的应用以及应用成本消费的趋势，提供应用维度的成本优化分析能力。

配置成本分析

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 成本分析**。
5. 在**成本分析**页面单击集群的**KubernetesWorkerRole-*****，在**RAM控制台**完成RAM授权策略。



i. 在**RAM角色管理**页面单击**K8sWorkerRolePolicy-*****权限策略名称。

ii. 进入**权限策略管理**页面，单击**修改策略内容**。

iii. 在**修改策略内容**面板增加以下内容后，单击**确定**。

```
{
  "Action": [
    "bssapi:QueryInstanceBill"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
{
  "Action": [
    "ecs:DescribeSpotPriceHistory",
    "ecs:DescribeInstances",
    "ecs:DescribePrice"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

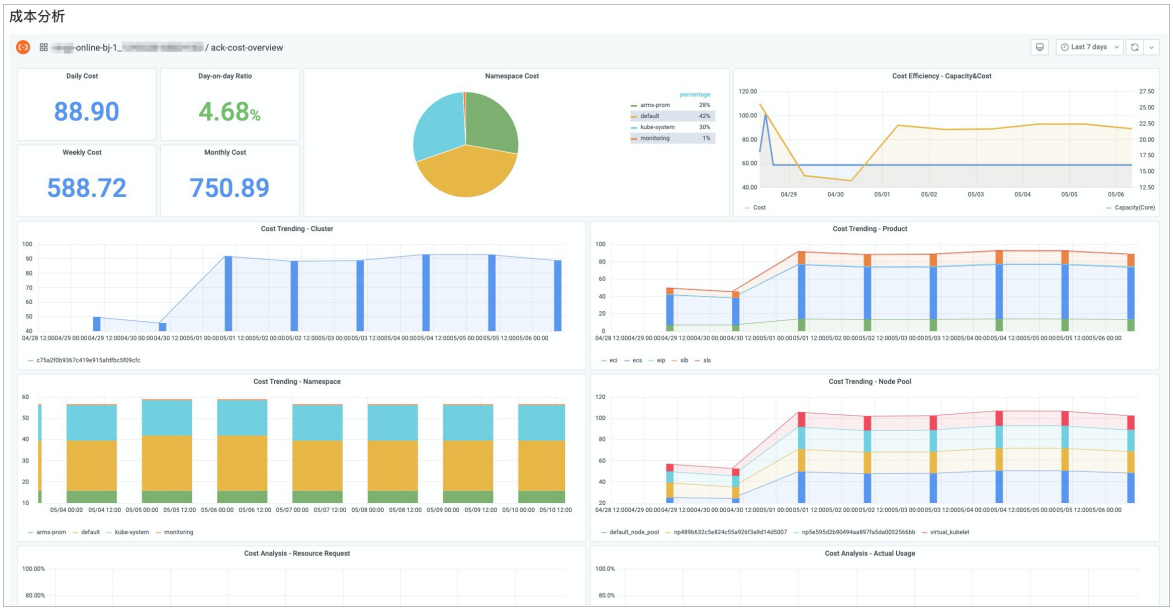
> **说明** 多个策略内容需要用半角逗号（,）分隔。

6. 在**成本分析**页面单击**开始安装**。

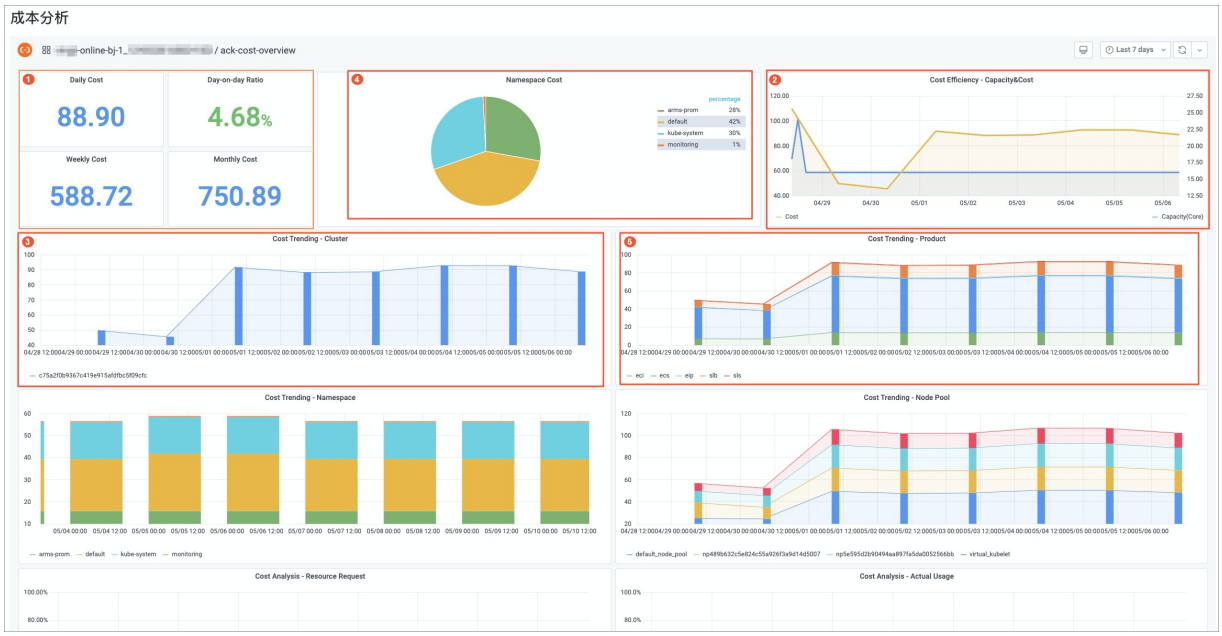


7. 在**成本分析**页面查看成本可视化大盘。

> **说明** 在开启成本分析功能后第二天08: 00自动显示账单数据。



使用成本分析能力



成本可视化大盘的相关功能介绍如下表：

序号	功能	描述
①	通过成本趋势判断集群浪费情况	成本的费用，Daily Cost为昨日费用，Day-on-day Ratio为相对昨日费用的变化比率。 当相对昨日费用的变化比率的字体颜色为绿色时，表示相比前一日的成本是降低的；当相对昨日费用的变化比率的字体颜色为红色时，表示相比前一日的成本是增涨的。
②		黄色表示成本消费，蓝色表示实际的集群容量。通常情况下，两条曲线会存在一定的相关性。 当对两者进行相关性对比后发现两者呈现不一致的趋势，表示集群的单位核成本异常，需要检查是否使用了更贵的资源。

序号	功能	描述
③		一段时间内的成本变化趋势。
④	通过命名空间进行部门成本分账	在一个集群中，可能存在各种不同的规格、付费方式的节点类型，当您通过命名空间进行分账时，不能够单一的通过每个命名空间的资源申请值进行判断，同时需要考虑命名空间下Pod所在节点情况。 成本分析通过alibaba-cloud-price-exporter将每个节点的实时费用进行了转换，在统计命名空间维度的费用时，可转换为： **Σ(Pod资源申请/节点容量)*节点单位价格** 这种方式可以精确的估算出命名空间的成本费用，不过由于用户折扣、代金券抵扣、包年包月等各种延迟计费策略，可能会导致命名空间的成本费用和集群的实际账单费用不一致。但是，可以通过命名空间的占比乘以集群总费用的方式获得命名空间级别的分账。
⑤	通过云产品费用趋势与比例分析云产品成本	在集群中会包含多种不同的云产品，不同的使用方式、计费模型会导致云产品的费用上存在比较多的差异。可以通过成本趋势和组合看到不同的云产品消费，从而进行成本决策。

常见问题

问题一：为什么开启成本可视化后没有数据显示？ ❯

问题二：为什么命名空间的费用相加与实际的费用账单不相等？ ❯

15.弹性伸缩

15.1. 弹性伸缩概述

弹性伸缩是根据业务需求和策略，经济地自动调整弹性计算资源的管理服务。本文介绍弹性伸缩的背景信息和弹性伸缩涉及的组件。

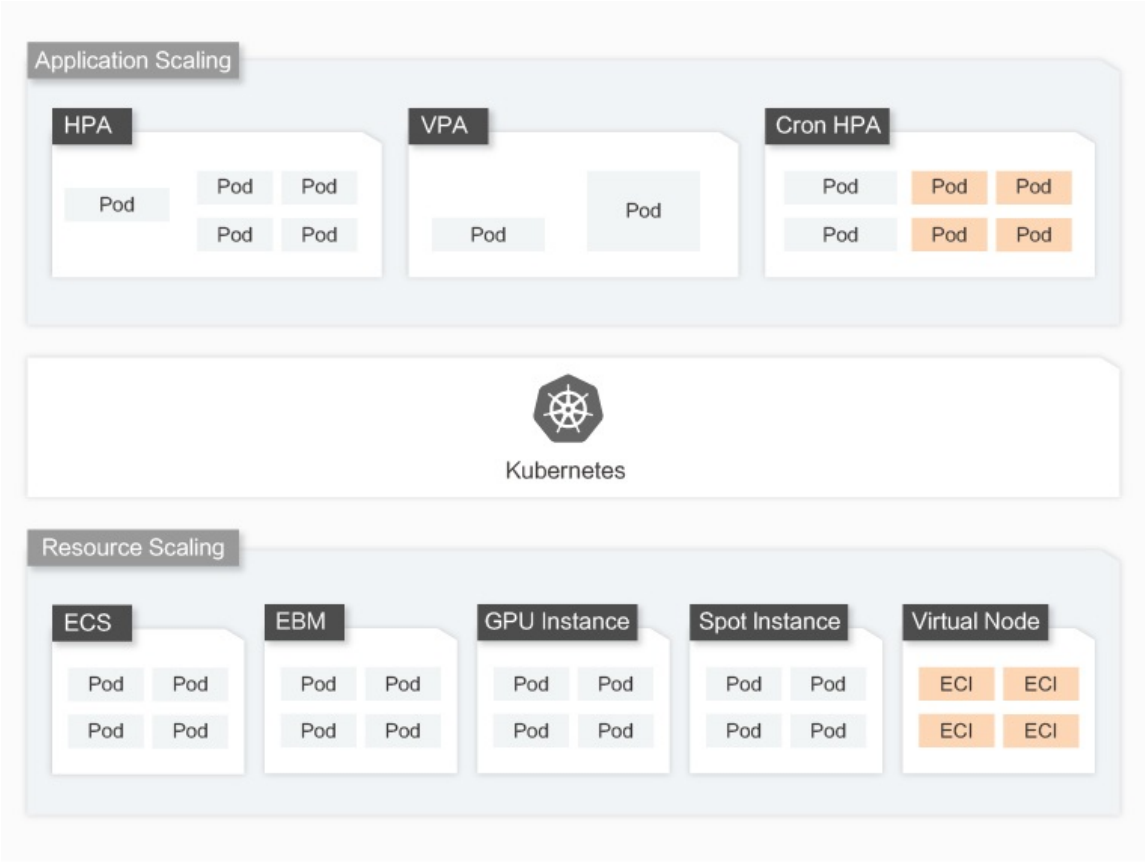
背景介绍

弹性伸缩是ACK上被广泛采用的功能，典型的场景包含在线业务弹性、大规模计算训练、深度学习GPU或共享GPU的训练与推理、定时周期性负载变化等。弹性伸缩分为两个维度：

- 调度层弹性，主要是负责修改负载的调度容量变化。例如，HPA是典型的调度层弹性组件，通过HPA可以调整应用的副本数，调整的副本数会改变当前负载占用的调度容量，从而实现调度层的伸缩。
- 资源层弹性，主要是集群的容量规划不能满足集群调度容量时，会通过弹出ECS或ECI等资源的方式进行调度容量的补充。

两层的弹性组件与能力可以分开使用，也可以结合在一起使用，并且两者之间是通过调度层面的容量状态进行解耦。

ACK弹性伸缩组件介绍



调度层弹性组件介绍

组件名称	组件介绍	适用场景	使用限制	参考文档
HPA	Kubernetes内置组件，主要面向在线业务。	在线业务	适用于Deployment、StatefulSet等实现scale接口的对象。	容器水平伸缩（HPA）

组件名称	组件介绍	适用场景	使用限制	参考文档
VPA【alpha】	开源社区组件，主要面向大型单体应用。	大型单体应用	适用于无法水平扩展的应用，通常是在Pod出现异常恢复时生效。	容器垂直伸缩（VPA）
CronHPA	ACK开源的组件，主要面向应用资源使用率存在周期性变化的场景。	周期性负载业务	适用于Deployment、StatefulSet等，实现了scale接口的对象。此外CronHPA提供了HPA对象的兼容能力，您可以同时使用CronHPA与HPA。	容器定时伸缩（CronHPA）
Elastic-Workload	ACK组件，主要面向精细化角度场景，例如希望在不同的可用区进行负载分布。	精细调度场景	适用于在线业务精细化控制的场景。例如，一个Deployment部分的应用运行在ECS上，而超出的部分运行在ECI上。	安装Elastic Workload

资源层弹性组件介绍

组件名称	组件介绍	适用场景	资源交付速度	参考文档
cluster-autoscaler	Kubernetes社区开源组件，节点水平伸缩组件，阿里云提供了独有的调度、弹性优化、成本优化的功能。	全场景支持，适合在线业务、深度学习、大规模成本算力交付等。	以100节点为一个交付批次为例： • 标准模式：120s • 极速模式：60s • 标准模式-Qboot镜像：90s • 极速模式-Qboot镜像：45s 关于Qboot镜像的更多信息，请参见Alibaba Cloud Linux 2快速启动版。	节点自动伸缩
virtual-node	ACK开源组件，提供无服务器运行时环境。开发者无需关心节点资源，只需针对Pod按量付费即可。	部分场景支持，主要包括在线突增流量、CI/CD、大数据作业。	以1000个Pod为一个交付批次为例： • 未开启镜像缓存：30s • 已开启镜像缓存：15s	通过部署ACK虚拟节点组件创建ECI Pod
virtual-kubelet-autoscaler	ACK服务组件，提供无服务弹性伸缩能力。	部分场景支持，主要包括在线突增流量、CI/CD、大数据作业。	以1000个Pod为一个交付批次为例： • 未开启镜像缓存：30s • 已开启镜像缓存：15s	安装virtual-kubelet-autoscaler

15.2. 节点自动伸缩

ACK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

在Kubernetes中，节点自动伸缩的工作原理与传统意义上基于使用率阈值的模型有所差别，这也是很多开发者在从传统的IDC或者其他编排系统迁移到Kubernetes后最难理解的地方。例如，将Swarm集群迁移至ACK集群。

传统的弹性伸缩模型是基于使用率的，例如：一个集群中有3个节点，当集群中的节点CPU、内存使用率超过特定的阈值时，此时弹出新的节点。但当深入思考时会发现以下几个问题：

1. 阈值是怎么选择与判断的？ ❯
2. 弹出实例后是怎么缓解压力的？ ❯
3. 怎么判断以及执行实例的缩容？ ❯

Kubernetes节点伸缩是怎么解决以上问题的呢？Kubernetes是通过调度与资源解耦的两层弹性模型来解决的。

基于资源的使用率来触发应用副本的变化，也就是调度单元的变化。而当集群的调度水位达到100%的时候会触发资源层的弹性扩容，当资源弹出后，无法调度的单元会自动调度到新弹出的节点上，从而降低整个应用的负载状况。以下介绍Kubernetes弹性伸缩的技术细节：

1. 如何判断节点的弹出？ ❯
2. 如何判断节点的缩容？ ❯
3. 多个分组之间如何选择？ ❯
4. 如何提高弹性伸缩的成功率？ ❯
5. 如何提高弹性伸缩的速度？ ❯

注意事项

- 默认单个用户每个区域按量付费实例的配额是50 vCPU，在专有网络下创建的单个路由表可创建的自定义路由数限额是48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。
- 选择绑定EIP时，通过ECS控制台直接删除自动伸缩组或者扩容出的ECS节点会导致弹性公网EIP（Elastic IP Address）无法自动释放。

步骤一：执行自动伸缩

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，执行以下操作进入**集群自动弹性伸缩配置**页面。

   您可以通过以下两种路径配置自动弹性伸缩。

   - 路径一：在目标集群右侧的**操作**列下，选择**更多 > 自动伸缩**。
   - 路径二：
     a. 选择目标集群然后单击操作列的**详情**。
     b. 在集群应用配置左侧导航栏，选择**节点管理 > 节点池**。
     c. 在节点池页面右上方，单击**集群自动弹性伸缩配置**。

步骤二：授权

自动弹性伸缩的授权操作分以下三种场景：

对于节点权限被收敛的集群，您需要授权RAM角色**AliyunCSManagedAutoScalerRole**。

> **说明** 此授权每个阿里云账号只需要进行一次。

1. 开通ESS服务。

i. 单击提示对话框中的第一个链接，进入**弹性伸缩服务ESS**页面。



ii. 单击**开通ESS服务**，进入**云产品开通页**。

iii. 选中**我已阅读并同意**复选框，单击**立即开通**。

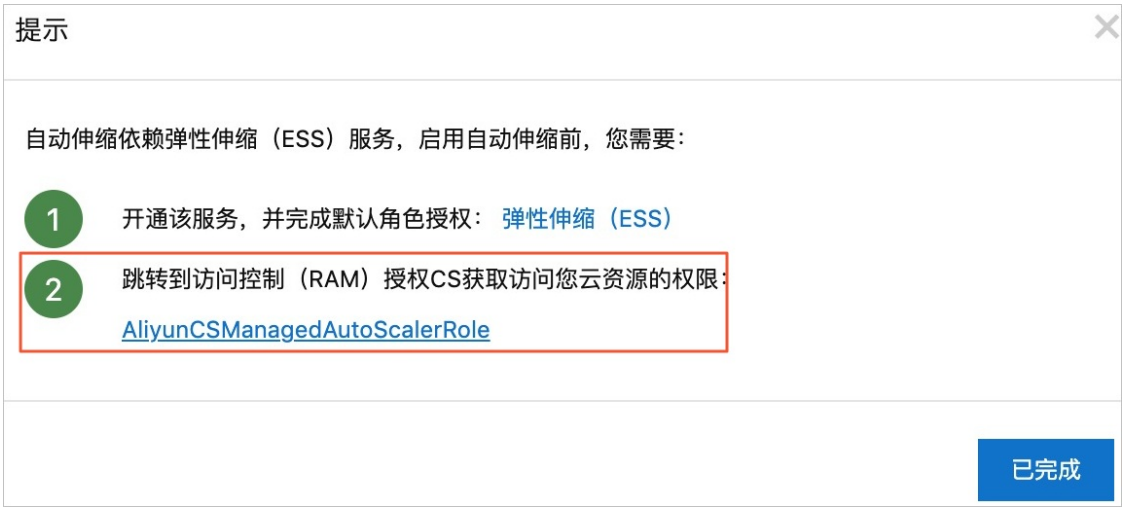
iv. 开通成功后，在**开通完成**页签，单击**管理控制台**，进入**弹性伸缩服务ESS**页面。

v. 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。

vi. 单击**同意授权**。

2. 授权角色。

i. 单击弹出对话框中的第二个链接。



> **说明** 此处需要以阿里云账号登录控制台。

ii. 在**云资源访问授权**页面，单击**同意授权**。



1. 开通ESS服务。

i. 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。



ii. 单击**开通ESS服务**，进入**云产品开通页**。

iii. 选中**我已阅读并同意**复选框，单击**立即开通**。

iv. 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。

v. 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。

vi. 单击**同意授权**。

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。

i. 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。

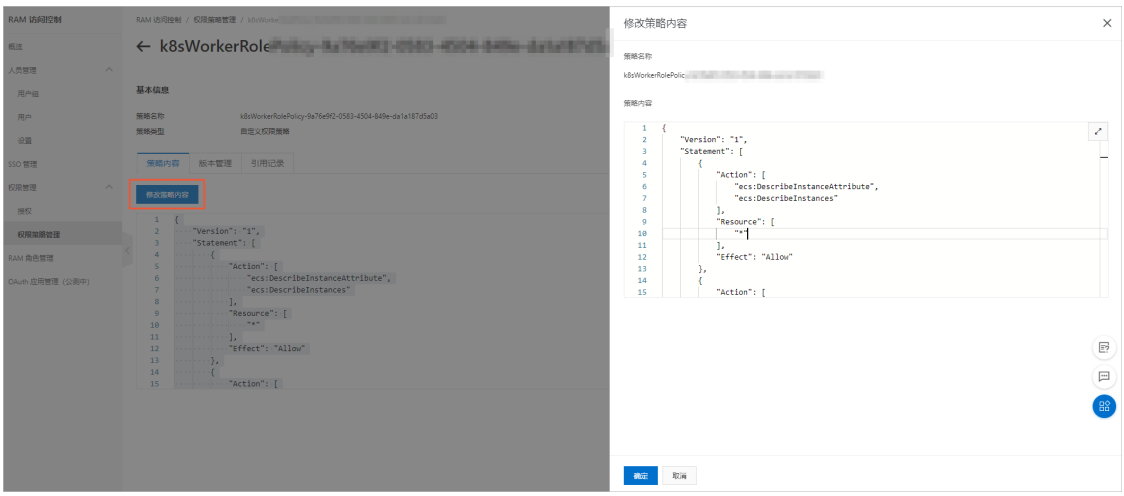


> **说明** 此处需要以阿里云账号登录控制台。

ii. 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。



iii. 单击**修改策略内容**，从右侧滑出侧边栏**修改策略内容**面板。



iv. 在**策略内容**的 Action 字段中补充以下策略，然后单击**确定**。

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

> **说明** 在策略内容的任意一个 Action 字段的最后一行补充英文逗号（，）。

如果自动伸缩组需要绑定EIP，授权步骤如下。

1. 开通ESS服务。
   i. 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。



   ii. 单击**开通ESS服务**，进入**云产品开通页**。
   iii. 选中**我已阅读并同意**复选框，单击**立即开通**。
   iv. 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。
   v. 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。
   vi. 单击**同意授权**。
   vii.

   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。

i. 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。

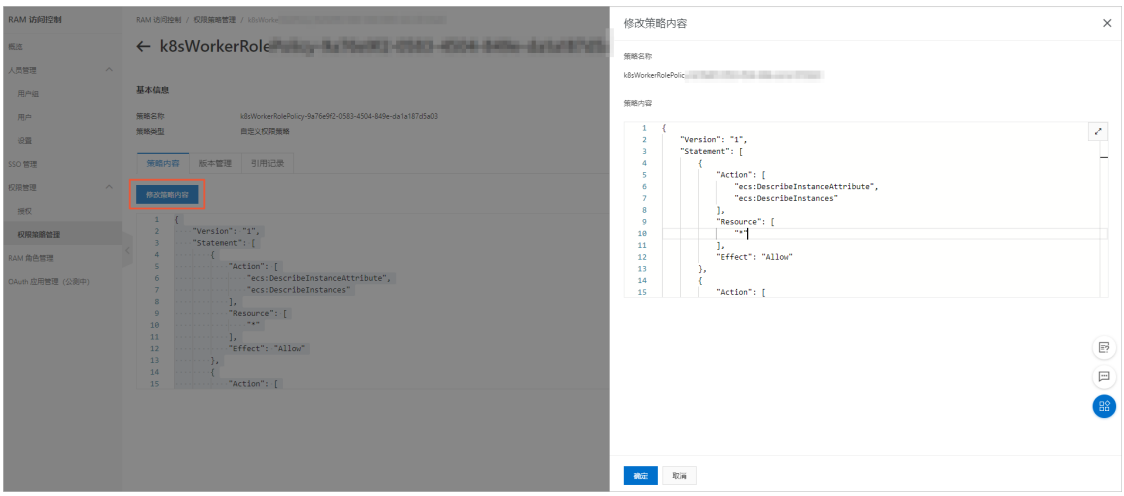


> **说明** 此处需要以阿里云账号登录控制台。

ii. 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。



iii. 单击**修改策略内容**，从右侧滑出侧边栏**修改策略内容**面板。



iv. 在**策略内容**的 Action 字段中补充以下策略，然后单击**确定**。

```
"ecs:AllocateEipAddress",
"ecs:AssociateEipAddress",
"ecs:DescribeEipAddresses",
"ecs:DescribeInstanceTypes",
"ecs:DescribeInvocationResults",
"ecs:DescribeInvocations",
"ecs:ReleaseEipAddress",
"ecs:RunCommand",
"ecs:UnassociateEipAddress",
"ess:CompleteLifecycleAction",
"ess:CreateScalingRule",
"ess:DeleteScalingRule",
"ess:Describe*",
"ess:DetachInstances",
"ess:ExecuteScalingRule",
"ess:ModifyScalingGroup",
"ess:ModifyScalingRule",
"ess:RemoveInstances",
"vpc:AllocateEipAddress",
"vpc:AssociateEipAddress",
"vpc:DescribeEipAddresses",
"vpc:DescribeVSwitches",
"vpc:ReleaseEipAddress",
"vpc:UnassociateEipAddress",
"vpc:TagResources"
```

> **说明** 在策略内容的任意一个 Action 字段的最后一行补充英文逗号（，）。

v. 在**RAM角色管理**详情页，单击**信任策略管理**页签，在**修改信任策略**中添加oos.aliyuncs.com，然后单击**确定**。



节点权限被收敛的集群

节点权限未被收敛的集群

自动伸缩节点池需要绑定EIP的集群

步骤三：配置自动伸缩

1. 在**集群自动弹性伸缩配置**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。
允许缩容	是否允许进行节点缩容。若设置为不允许缩容，缩容相关配置将不生效，请谨慎设置。
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。
GPU缩容阈值	GPU类型实例的缩容阈值，当低于配置的阈值时，GPU类型节点会进行缩容。
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	扩容出的节点，在静默时间过后，方可进入缩容判断。
弹性灵敏度	集群自动弹性伸缩配置支持设置弹性灵敏度，以调整系统判断伸缩的间隔时间。目前支持15s、30s、60s的选项，默认值是30s。
节点池扩容顺序策略	
least-waste	默认策略。如果可扩容节点池有多个，从中选择一个资源浪费最少的节点池进行扩容。
random	随机策略。如果可扩容节点池有多个，从中任意选择一个节点池进行扩容。

配置	说明

priority	优先级策略。如果可扩容节点池有多个，会按照您自定义的伸缩组顺序，选择优先级高的节点池进行扩容。
	优先级的配置存储在kube-system命名空间下ConfigMap的 cluster-autoscaler-priority-expander 中。当自动伸缩进行扩容时，实时读取此配置，把可扩容的伸缩组ID与配置中伸缩组ID进行匹配，在匹配上的伸缩组中选取优先级最大值的伸缩组作为扩容对象。
	说明
	◦ 优先级取值1~100，必须为正整数。
	◦ 如果选择优先级策略，开启自动伸缩节点池的伸缩组ID必须配置优先级，一个伸缩组ID属于且只属于一个优先级。
	◦ 如果伸缩组ID没有配置在ConfigMap中，即使满足扩容需求，也会由于优先级未配置而不扩容。因此，如果选择使用priority策略，所有开启自动伸缩的节点池对应伸缩ID需要配置在ConfigMap的 cluster-autoscaler-priority-expander 中。
	配置的优先级策略示例如下：
	`kind: ConfigMap`
	`metadata:`
	`name: cluster-autoscaler-priority-expander`
	`data:`
	`priorities: |-`
	`10:`
	`- asg-1largeid #asg-1largeid优先级为10。`
	`- asg-2largeid #asg-2largeid优先级为10。`
	`50:`
	`- asg-3largeid #asg-3largeid优先级为50。`
	假设某场景下扩容asg-2largeid和asg-3largeid都可以满足集群扩容需求，但由于asg-3largeid优先级较高，自动伸缩会按照优先级对asg-3largeid扩容。

2. 单击页面右侧的**创建节点池**。
3. 在**创建节点池**对话框，设置伸缩组的配置项。

   有关配置项的详细说明，请参见创建Kubernetes托管版集群。部分配置项说明如下：

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。
自动伸缩	根据需要选择弹性伸缩的节点资源类型（包含*普通实例*、*GPU实例*、*共享GPU实例*、*抢占式实例*），与创建集群时所选择的节点类型一致。

配置	说明
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。
操作系统	在开启自动伸缩时，支持选择Alibaba Cloud Linux、CentOS、Windows镜像、Windows Core镜像。 **说明** 当所选镜像是Windows镜像或Windows Core镜像时，系统将自动配置污点（Taints）`{effect: 'NoSchedule', key: 'os', value: 'windows'}`。
密钥对	登录伸缩后的节点时所使用的密钥对。可以在ECS控制台新建密钥对。 **说明** 目前只支持密钥对方式登录。
RDS白名单	弹性伸缩后的节点可以访问的RDS实例。
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。
伸缩模式	支持*标准模式*和*极速模式*。 ○ *标准模式*：根据资源申请值的使用量，通过创建、释放ECS的方式进行伸缩。 ○ *极速模式*：通过创建、停机、启动的方式进行伸缩，提高再次伸缩的速度。
污点（Taints）	添加污点后，集群将不会将Pod调度到该节点上。

4. 单击**确认配置**，创建伸缩组。

预期结果

1. 在**集群自动弹性伸缩配置**页面，可以在普通实例下看到新创建的一个伸缩组。



2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 在**无状态**页面，选择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。
- 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*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。
  - 节点上的Pod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。
- 多个伸缩组在弹性伸缩的时候是如何被选择的？
  在Pod处在无法调度时，会触发弹性伸缩组件的模拟调度逻辑，会根据伸缩组配置的标签和污点以及实例规格等信息进行判断。当配置的伸缩组可以模拟调度Pod的时候，就会被选择进行节点弹出。当同时有多个伸缩组满足模拟调度条件的时候，默认采用的是最少浪费原则，即根据模拟弹出后节点上剩余的资源最小为原则进行抉择。

相关文档

- 控制集群API Server的公网访问能力
- 容器水平伸缩（HPA）
- 基于阿里云组件指标的容器水平伸缩
- 容器定时伸缩（CronHPA）
- 容器垂直伸缩（VPA）

15.3. 容器水平伸缩（HPA）

ACK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来进行配置。

体验实验室

为了帮助您快速上手本教程，ACK为您提供了可以动手操作的交互式实验环境。如您需要体验，请访问通过HPA进行Pod水平弹性伸缩。

通过容器服务控制台创建HPA应用

阿里云容器服务已经集成了HPA，您可以简便地通过容器服务控制台进行创建。您可以在创建应用的时候创建HPA，也可以在已有应用的基础上开启HPA。

方式一：在创建应用过程中，开启HPA

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面单击**使用镜像创建**。
6. 在**应用基本信息**页填写应用的名称等信息，然后单击**下一步**。

配置项	描述
应用名称	设置应用的名称。
副本数量	即应用包含的Pod数量，默认数量为2。
类型	定义资源对象的类型，可选择**无状态**、**有状态**、**任务**、**定时任务**、**守护进程集**。
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）。
时区同步	容器与节点是否使用相同的时区。

7. 在**容器配置**页进行容器设置，选择镜像，并设置所需的资源。然后单击**下一步**。详情请参见配置容器。

   > **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。

8. 在**高级设置**页的**访问设置**区域，单击服务的**创建**，设置服务的相关信息，详情请参见高级设置。
9. 在**高级设置**页选中**指标伸缩**的**开启**，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。有关Pod水平自动扩缩的算法，请参见算法细节。
   - **最大副本数**：该Deployment可扩容的容器数量上限。
   - **最小副本数**：该Deployment可缩容的容器数量下限。
10. 单击右下角的**创建**，一个支持HPA的Deployment就已经创建完毕。

结果验证

1. 单击**查看应用详情**或者在**工作负载 > 无状态**中单击**应用名称**或**操作**列的**详情**，单击**容器伸缩**，您可在部署的详情中查看伸缩组信息。
2. 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。

方式二：为已有应用开启HPA

以下以无状态应用为例，演示为已有应用开启HPA。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中单击目标应用名称。
6. 单击**容器伸缩**页签，然后单击**创建**。
7. 在**创建**对话框中，设置伸缩的配置。有关伸缩的参数配置，请参见上文步骤9的水平伸缩配置。
8. 单击**确定**。

通过kubectl命令创建HPA应用

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。

1. 创建并复制以下内容到*nginx.yml*中。

   Deployment的编排模板如下。

```
apiVersion: apps/v1
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
```

2. 执行以下命令，创建Nginx应用。

```
kubectl create -f nginx.yml
```

3. 创建HPA。

   通过scaleTargetRef设置当前HPA绑定的对象，在本例中绑定名叫**nginx**的Deployment。

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
 name: nginx-hpa
spec:
 scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: nginx
 minReplicas: 1
 maxReplicas: 10
 metrics:
 - type: Resource
  resource:
   name: cpu
   target:
    type: Utilization
    averageUtilization: 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。

4. 执行 `kubectl describe hpa name` 会发现有类似以下的警告信息。

```
Warning  FailedGetResourceMetric     2m (x6 over 4m)  horizontal-pod-autoscaler  missing request for cpu on c
ontainer nginx in pod default/nginx-deployment-basic-75675f5897-mqzs7
Warning  FailedComputeMetricsReplicas  2m (x6 over 4m)  horizontal-pod-autoscaler  failed to get cpu utilizati
on: missing request for cpu on container nginx in pod default/nginx-deployment-basic-75675f5
```

5. 创建好HPA后，再次执行 `kubectl describe hpa name` 命令。

   可以看到以下信息，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。

15.4. 容器定时伸缩（CronHPA）

应对资源浪费的场景，阿里云服务容器提供kubernetes-cronhpa-controller组件，实现资源定时扩容。本文介绍如何实现容器的定时伸缩，及介绍定时伸缩CronHPA兼容HPA的原理。

前提条件

- 您已创建一个容器服务Kubernetes集群，请参见创建Kubernetes托管版集群。
- 在本地计算机上安装Helm，请参见按照Install Helm，并确保Helm版本v2.11.0及以上。
- 在本地使用命令前，您需要先设置通过kubectl工具连接集群。

背景信息

kubernetes-cronhpa-controller是一个Kubernetes HPA controller，按照类似Crontab的策略定时地对容器服务Kubernetes集群进行扩缩容。您可以把CronHorizontalPodAutoscaler（以下简称CronHPA）用在任何Kubernetes中定义的对象上，只要该对象支持伸缩子资源（如Deployment和StatefulSet）的项目已在GitHub开源。

以下举例对CronHPA的各字段进行解释说明。

```
apiVersion: autoscaling.alibabacloud.com/v1beta1
kind: CronHorizontalPodAutoscaler
metadata:
 labels:
  controller-tools.k8s.io: "1.0"
 name: cronhpa-sample
 namespace: default
spec:
 scaleTargetRef:
   apiVersion: apps/v1
   kind: Deployment
   name: nginx-deployment-basic
 excludeDates:
 # exclude November 15th
 - "* * * 15 11 *"
 # exclude every Friday
 - "* * * * * 5"
 jobs:
 - name: "scale-down"
  schedule: "30 */1 * * * *"
  targetSize: 1
 - name: "scale-up"
  schedule: "0 */1 * * * *"
  targetSize: 3
  runOnce: true
```

字段	说明
scaleTargetRef	scaleTargetRef指定去扩缩容对象。如果对象支持scale子资源，CronHPA即可支持。

excludeDates	excludeDates是日期数组。当遇到符合excludeDates描述的日期时任务将会被跳过。
	说明 最小单位为天。
	`- "* * * * * *"` 表示-"<seconds> <minutes> <hours> <days of month> <months>"。
	如您想在11月15日不运行任务，可像以下示例一样指定excludeDates。
	`excludeDates:`
	`- "* * * 15 11 *"`

字段	说明
jobs	支持在一个spec中设定多个CronHPA任务。每个CronHPA任务可以配置以下字段： • name：它在一个CronHPA中应是唯一的，这样就可以通过name来区分不同的任务。 • schedule：它的策略和crontab类似。kubernetes-cronhpa-controller使用了Golang库（go-cron）功能更强大的cron，它支持更多表示规则。创建时请严格按照以下格式说明进行设置，否则将导致创建失败。 cron表达式的格式如下： Field name | Mandatory? | Allowed values | Allowed special characters ---------- | ---------- | -------------- | -------------------------- Seconds | Yes | 0-59 | * / , - Minutes | Yes | 0-59 | * / , - Hours | Yes | 0-23 | * / , - Day of month | Yes | 1-31 | * / , - ? Month | Yes | 1-12 or JAN-DEC | * / , - Day of week | Yes | 0-6 or SUN-SAT | * / , - ? • targetSize：到调度时间时，您想扩缩容到的Pod的数目。 • runOnce：如果runOnce设置为true，那么任务将只执行一次，第一次执行完则exit。

安装CronHPA组件

您可以通过以下三种方式安装CronHPA组件ack-kubernetes-cronhpa-controller。

方式一：通过组件管理入口安装CronHPA组件。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**组件管理**页面单击**其他**页签，找到**ack-kubernetes-cronhpa-controller**，然后单击**安装**。

方式二：通过系统组件管理入口安装CronHPA组件。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 系统组件管理**。
5. 在**组件管理**页面单击**其他**页签，找到**ack-kubernetes-cronhpa-controller**，然后单击**安装**。

方式三：通过应用目录入口安装CronHPA组件。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面**名称**文本框中输入*kubernetes-cronhpa-controller*，然后单击**ack-kubernetes-cronhpa-controller**组件。
4. 在**应用目录-ack-kubernetes-cronhpa-controller**页面右侧的**创建**区域，选择目标集群，然后单击**创建**。

如果您无需使用容器定时伸缩功能，可以删除CronHPA组件。关于删除CronHPA组件ack-kubernetes-cronhpa-controller的具体步骤，请参见管理组件或删除发布。

创建定时伸缩CronHPA任务

为应用创建和运行CronHPA前，请确保集群中的CronHPA组件已正常运行，且当前应用只有一个HPA对象。关于CronHPA和HPA的兼容策略，请参见下文的定时伸缩CronHPA兼容HPA。您可以在以下两种场景中创建CronHPA任务。

场景一：在创建应用时创建CronHPA任务。

在创建应用的**高级配置**页面的**伸缩配置**区域，选中**定时伸缩**右侧**开启**为应用创建定时伸缩任务。关于创建应用的详细步骤，请参见创建无状态工作负载Deployment或者创建有状态工作负载StatefulSet。



容器服务管理控制台会自动检查是否已安装CronHPA组件。如果CronHPA组件未被安装，页面提示**点击安装**。安装CronHPA组件后，页面显示定时伸缩任务的创建配置。配置参数的详细描述如下：

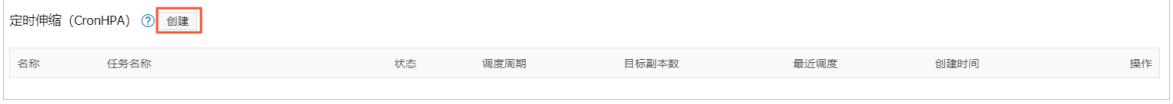
定时伸缩配置说明

参数	描述
定时任务名称	为定时任务定义名称。每个任务的名称是唯一的，不能与其他任务名称重复。
目标副本数	当到达设定计划时间时，应用副本数自动伸缩至该值。
调度周期	设置调度周期。 关于为定时伸缩任务设置调度周期的详细描述，请参见AliyunContainerService/kubernetes-cronhpa-controller。

场景二：为已有应用创建CronHPA任务。

本文以为无状态应用创建定时伸缩任务为例，说明如何为已有应用创建CronHPA任务。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面，单击应用右侧**操作**列下的**详情**。
6. 单击**容器伸缩**页签，配置定时伸缩任务。
   - 如果CronHPA组件未被安装，页面提示**点击安装**。单击**点击安装**后，然后进行以下步骤。
   - 如果CronHPA组件已被安装，直接进行以下步骤。
7. 单击**定时伸缩（CronHPA）**右侧**创建**，然后在**创建**对话框中，设置定时伸缩任务的创建参数。

   关于创建定时伸缩任务的配置信息，请参见上文的定时伸缩配置说明。

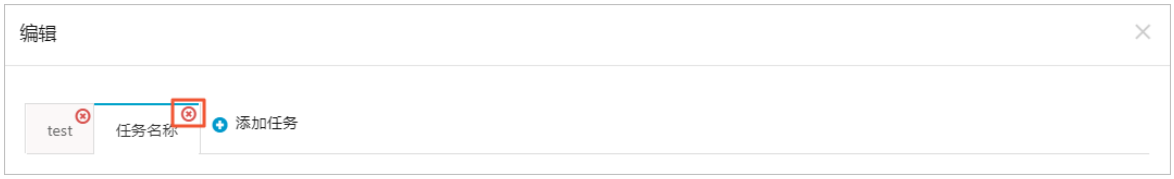
添加或修改定时伸缩任务

1. 请参见上文的为已有应用创建CronHPA任务的步骤进入**容器伸缩**页面。
2. 在**容器伸缩**页签的**定时伸缩（CronHPA）**区域，单击目标任务右侧**操作**下的**任务添加或编辑**。
3. 在**编辑**对话框中，单击**添加任务**进行任务创建或者点击已有CronHPA任务中进行修改，然后单击**确定**。



您可以进行以下操作删除CronHPA任务。

4. 在**编辑**对话框中，单击任务名称右上角的删除图标，然后单击**确定**。



定时伸缩CronHPA兼容HPA

定时伸缩CronHPA通过设置定时的方式触发容器的水平副本伸缩。为了防止突发的流量冲击等状况，您可能已经或者打算配置HPA保障应用的正常运行。如果同时检测到了HPA和CronHPA的存在，阿里云容器服务ACK会将HPA作为定时伸缩CronHPA的扩缩容对象，从而实现对该HPA定义的Deployment对象的定时扩缩容。

CronHPA和HPA的定义模板如下。

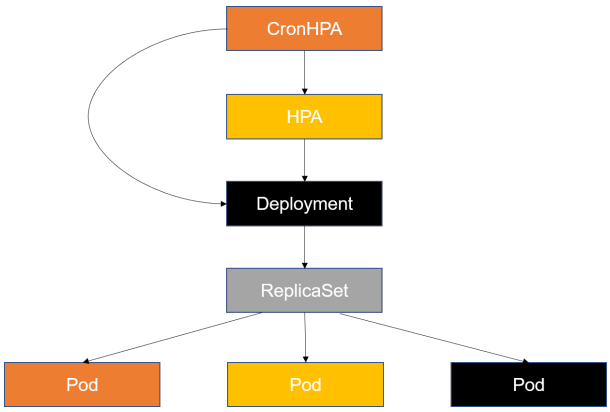
对比上述CronHPA和HPA资源的定义模板，可以看出：

- CronHPA和HPA都是通过scaleTargetRef字段来获取伸缩对象。
- CronHPA通过jobs的**crontab**规则定时伸缩副本数。
- HPA通过资源利用率判断伸缩的情况。

设想如果同时设置CronHPA和HPA，那么会出现CronHPA和HPA同时操作一个scaleTargetRef的场景。因为CronHPA和HPA相互独立无法感知，所以就会出现两个Controller各自工作，后执行的操作会覆盖先执行的操作。

CronHPA和HPA的兼容解决方案

从上文可知，CronHPA和HPA同时使用会导致后执行的操作覆盖先执行操作的本质原因是两个controller无法相互感知，那么只需要让CronHPA感知HPA的当前状态就能解决冲突问题。HPA的定义将Deployment配置在scaleTargetRef字段下，然后Deployment通过自身定义查找ReplicaSet，最后通过ReplicaSet调整真实的副本数目。阿里云容器服务ACK将CronHPA中的scaleTargetRef设置为HPA对象，然后通过HPA对象来寻找真实的scaleTargetRef，从而让CronHPA感知HPA的当前状态。



CronHPA兼容HPA的定义模板如下。

```
apiVersion: autoscaling.alibabacloud.com/v1beta1
kind: CronHorizontalPodAutoscaler
metadata:
 labels:
  controller-tools.k8s.io: "1.0"
 name: cronhpa-sample
spec:
  scaleTargetRef:
    apiVersion: autoscaling/v1
    kind: HorizontalPodAutoscaler
    name:  nginx-deployment-basic-hpa
  jobs:
  - name: "scale-down"
    schedule: "30 */1 * * * *"
    targetSize: 1
    runOnce: true
  - name: "scale-up"
    schedule: "0 */1 * * * *"
    targetSize: 3
    runOnce: true
```

通过上述CronHPA的定义，CronHPA可以明确知晓HPA中的minReplicas、maxReplicas、desiredReplicas的数值，但是同时也知晓HPA中scaleTargetRef所对应的当前Relicas值。CronHPA会通过调整HPA的方式感知HPA。CronHPA通过识别要达到的副本数与当前副本数两者间的较大值，判断是否需要扩缩容及修改HPA的上限；CronHPA通过识别CronHPA要达到的副本数与HPA的配置间的较小值，判断是否需要修改HPA的下限。

以下根据不同的场景，解释CronHPA兼容HPA的规则。

HPA（min/max）	Cronhpa	Deployment	扩缩结果	兼容规则说明
1/10	5	5	• HPA（min/max）：1/10 • Deployment：5	当CronHPA中的目标副本数和当前副本数一致时，HPA中的最大和最小副本数，还有应用当前的副本数无需变更。
1/10	4	5	• HPA（min/max）：1/10 • Deployment：5	当CronHPA中的目标副本数低于当前副本数时，保留当前副本数。

HPA（min/max）	Cronhpa	Deployment	扩缩结果	兼容规则说明
1/10	6	5	• HPA（min/max）：6/10 • Deployment：6	• 当CronHPA中的目标副本数高于当前副本数时，保留CronHPA的目标副本数。 • CronHPA目标副本数高于HPA副本数下限（minReplicas）时，修改HPA的副本数下限。
5/10	4	5	• HPA（min/max）：4/10 • Deployment：5	• 当CronHPA中的目标副本数低于当前副本数时，保留当前应用副本数。 • CronHPA目标副本数低于HPA副本数下限（minReplicas）时，修改HPA的副本数下限。
5/10	11	5	• HPA（min/max）：11/11 • Deployment：11	• 当CronHPA中的目标副本数高于当前副本数时，保留CronHPA的目标副本数。 • CronHPA目标副本数高于HPA副本数上限（maxReplicas）时，修改HPA的副本数上限。

表格中参数说明如下：

- HPA（min/max）：表示HPA定义中最小和最大的副本数值。
- CronHPA：表示预期扩缩容的副本数。
- Deployment：表示当前应用的副本数。

从上述看出，CronHPA不会直接调整Deployment的副本数目，而是通过HPA来操作Deployment，这样可以避免HPA和CronHPA的冲突问题。

CronHPA HPA

相关文档

- 容器水平伸缩（HPA）

15.5. 容器垂直伸缩（VPA）

通过在ACK集群上部署安装VPA（vertical-pod-autoscaler），ACK可以提供垂直的容器伸缩的功能。VPA会基于Pod的资源使用情况自动为集群设置资源占用的限制，从而让集群将Pod调度到有足够资源的最佳节点上。VPA也会保持最初容器定义中资源request和limit的占比。本文介绍如何配置YAML实现容器的垂直伸缩。

前提条件

请确保您已完成以下操作：

- 已创建一个Kubernetes集群，且ACK Kubernetes版本高于1.12。具体操作请参见创建Kubernetes托管版集群。
- 已使用命令行工具连接集群。具体操作请参见通过kubectl工具连接集群。
- 已卸载集群中已经部署的VPA，以避免新安装的VPA与旧版VPA冲突而导致VPA不可用。

背景信息

> **注意** 容器垂直伸缩功能目前处于试验阶段，请谨慎使用。

- 更新正在运行的Pod资源配置是VPA的一项试验性功能，会导致Pod的重建和重启，而且有可能被调度到其他的节点上。
- VPA不会驱逐没有在副本控制器管理下的Pod。目前对于这类Pod，Auto模式等同于Initial模式。
- 目前VPA不能和监控CPU和内存度量的Horizontal Pod Autoscaler（HPA）同时运行，除非HPA只监控其他定制化的或者外部的资源度量。
- VPA使用admission webhook作为其准入控制器。如果集群中有其他的admission webhook，需要确保它们不会与VPA发生冲突。准入控制器的执行顺序定义在API Server的配置参数中。
- VPA会处理出现的绝大多数OOM（Out Of Memory）的事件，但不保证所有的场景下都有效。
- VPA的性能还没有在大型集群中测试过。
- VPA对Pod资源requests的修改值可能超过实际的资源上限，例如节点资源上限、空闲资源或资源配额，从而造成Pod处于Pending状态无法被调度。同时使用集群自动伸缩（ClusterAutoscaler）可以一定程度上解决这个问题。
- 多个VPA同时匹配同一个Pod会造成未定义的行为。

安装vertical-pod-autoscaler

1. 通过以下命令创建vertical-pod-autoscaler的CRD。

   CRD可以提高Kubernetes的扩展能力，详情请参见Extend the Kubernetes API with CustomResourceDefinitions。

   ```
   kubectl apply -f crd.yaml
   ```

   以下为*crd.yaml*的定义。

   ```
   apiVersion: apiextensions.k8s.io/v1beta1
   kind: CustomResourceDefinition
   metadata:
    name: verticalpodautoscalers.autoscaling.k8s.io
    annotations:
     "api-approved.kubernetes.io": "https://github.com/kubernetes/kubernetes/pull/63797"
   spec:
    group: autoscaling.k8s.io
    scope: Namespaced
    names:
     plural: verticalpodautoscalers
     singular: verticalpodautoscaler
     kind: VerticalPodAutoscaler
     shortNames:
      - vpa
    version: v1beta1
    versions:
     - name: v1beta1
       served: false
       storage: false
     - name: v1beta2
       served: true
       storage: true
     - name: v1
       served: true
       storage: false
    validation:
     # openAPIV3Schema is the schema for validating custom objects.
     openAPIV3Schema:
   ```

```
    openAPIV3Schema:
      type: object
      properties:
        spec:
          type: object
          required: []
          properties:
            targetRef:
              type: object
            updatePolicy:
              type: object
              properties:
                updateMode:
                  type: string
            resourcePolicy:
              type: object
              properties:
                containerPolicies:
                  type: array
                  items:
                    type: object
                    properties:
                      containerName:
                        type: string
                      controlledValues:
                        type: string
                        enum: ["RequestsAndLimits", "RequestsOnly"]
                      mode:
                        type: string
                        enum: ["Auto", "Off"]
                      minAllowed:
                        type: object
                      maxAllowed:
                        type: object
                      controlledResources:
                        type: array
                        items:
                          type: string
                          enum: ["cpu", "memory"]
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: verticalpodautoscalercheckpoints.autoscaling.k8s.io
  annotations:
    "api-approved.kubernetes.io": "https://github.com/kubernetes/kubernetes/pull/63797"
spec:
  group: autoscaling.k8s.io
  scope: Namespaced
  names:
    plural: verticalpodautoscalercheckpoints
    singular: verticalpodautoscalercheckpoint
    kind: VerticalPodAutoscalerCheckpoint
    shortNames:
      - vpacheckpoint
  version: v1beta1
  versions:
    - name: v1beta1
```

```
    served: false
    storage: false
  - name: v1beta2
    served: true
    storage: true
  - name: v1
    served: true
    storage: false
```

2. 安装vertical-pod-autoscaler的组件。

   vertical-pod-autoscaler的组件包括：admission-controller、recommender、updater。

   > **说明** 安装admission-controller组件前，您需要用此脚本为Webhook生成证书。

   安装admission-controller组件 安装recommender组件 安装updater组件

验证安装VPA

1. 使用以下YAML定义创建名为nginx-deployment-basic的Deployment和名为nginx-deployment-basic-vpa的VPA资源。

```
apiVersion: apps/v1
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
    ports:
    - containerPort: 80
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
 name: nginx-deployment-basic-vpa
spec:
 targetRef:
  apiVersion: "apps/v1"
  kind:    Deployment
  name:    nginx-deployment-basic
 updatePolicy:
  updateMode: "Off"
```

> **说明** 设置updateMode为Off，并且将Deployment中的资源requests和limits留空。

2. 执行以下命令可以查询VPA为Deployment推荐的CPU和内存资源的requests值。

> **说明** 执行命令查询VPA为Deployment推荐的CPU和内存资源的requests值时，需要等待两分钟，才能返回结果。

```
kubectl describe vpa nginx-deployment-basic-vpa
```

执行命令后可以看到以下VPA为Deployment推荐的值。

```
recommendation:
  containerRecommendations:
  - containerName: nginx
    lowerBound:
      cpu: 50m
      memory: 300144k
    target:
      cpu: 50m
      memory: 300144k
    upperBound:
      cpu: 8031m
      memory: 800000k
```

您可以根据推荐值来实际配置Deployment中资源的requests。VPA会持续的监控应用资源的使用情况，并提供优化建议。

相关文档

- 容器水平伸缩（HPA）

15.6. 基于阿里云组件指标的容器水平伸缩

本文介绍如何实现阿里云指标自动伸缩，帮助您完成集群内应用的扩缩容。

前提条件

已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

背景信息

很多场景下，往往需要更多的监测指标来指导集群的扩缩容，如HTTP请求率、Ingress QPS等，但是默认HPA并不支持这些自定义的指标或外部指标，但是Kubernetes提供了External Metrics的机制，为应用提供更加便捷的扩缩机制。

部署alibaba-cloud-metrics-adapter组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面**阿里云应用**页签下单击**弹性伸缩**，单击ack-alibaba-cloud-metrics-adapter。
4. 在**创建**区域选择目标**集群**，单击**创建**。
5. 查看已创建的ack-alibaba-cloud-metrics-adapter组件信息。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**工作负载 > 有状态**。

iv. 在集群管理页左侧导航栏中，选择**应用 > Helm**。

在Helm页面可看到**ack-alibaba-cloud-metrics-adapter**组件已经成功部署到集群中。

配置HPA自动伸缩示例

本文以创建名为Nginx的Deployment和Service为例配置HPA自动伸缩。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用YAML创建资源**。
6. 设置**示例模板**为**自定义**，并使用以下YAML文件创建一个应用，并暴露一个ClusterIP的Service，然后单击**创建**。

```
apiVersion: apps/v1
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
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
 name: nginx
 namespace: default
spec:
 ports:
  - port: 80
   protocol: TCP
   targetPort: 80
 selector:
  app: nginx
 type: ClusterIP
```

7. 在集群管理页左侧导航栏中，选择**网络 > 路由**，并在**路由**页面右上方，单击**创建**。
8. 在**创建**面板中填写相关信息，然后单击**创建**。完成创建后，系统自动跳转至**路由**页面。
9. 在生成的Ingress地址右侧，单击**详情**查看路由信息。
10. 配置HPA。

i. 单击左上角的 < 符号，返回**集群列表**页面。

ii. 在控制台左侧导航栏选择**市场 > 编排模板**。

iii. 在**模板列表**页面，选择hpa并单击**创建应用**，然后部署以下文件。

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: ingress-hpa
spec:
 scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: nginx-deployment-basic
 minReplicas: 2
 maxReplicas: 10
 metrics:
  - type: External
   external:
    metric:
     name: sls_ingress_qps
     selector:
      matchLabels:
       sls.project: "***" # 替换sls.project的值为真实值。
       sls.logstore: "nginx-ingress"
       sls.ingress.route: "default-nginx-80"
    target:
     type: AverageValue
     averageValue: 10
  - type: External
   external:
    metric:
     name: sls_ingress_latency_p9999
     selector:
      matchLabels:
       # default ingress log project is k8s-log-clusterId
       sls.project: "****"
       # default ingress logstre is nginx-ingress
       sls.logstore: "nginx-ingress"
       # namespace-svc-port
       sls.ingress.route: "default-nginx-80"
       # sls vpc endpoint, default true
       # sls.internal.endpoint:ture
    target:
     type: Value
     # sls_ingress_latency_p9999 > 10ms
     value: 10
```

HPA的配置涉及的参数如下：

参数名	描述

参数名	描述
sls.ingress.route	必填，参数的格式为：<namespace>-<svc>-<port>，例如default-nginx-80。 说明 <namespace>为Ingress所在的命名空间，<svc>是Ingress对应的Service名称，<port>是Ingress对应Service的Port名称。
sls.logstore	日志服务的日志库名称，必填。
sls.project	日志服务的Project名称，必填。
sls.internal.endpoint	设置通过内网还是外网访问日志服务，默认为true。选择true说明通过内网访问日志服务；选择false说明通过外网访问日志服务。

> 说明
> sls_ingress_qps与sls_ingress_latency_p9999是本文实现伸缩使用的两个指标。在target的type中，
> - 为sls_ingress_qps设置的是AverageValue，表示QPS要除以Pod的数目进行判断。
> - 为sls_ingress_latency_p9999设置的是Value，表示无需除以Pod的数目。
>
> 以上两种不同的type在HPA的设置中会经常用到。

iv. 在**模板列表 - hpa**页面右侧，单击**创建**。

11. 设置好HPA后，执行以下脚本进行压测。

```
#!/bin/bash
##使用Apache Benchmark对Ingress暴露的服务设置并发为10，时间为300秒的压测。
ab -t 300 -c 10 <ingress配置的域名>
```

12. 验证指标伸缩状态。
    i. 在控制台左侧导航栏单击**集群**，并在**集群列表**页面，选择目标集群**操作**列的**更多 > 通过CloudShell管理集群**。
    ii. 执行以下命令检查伸缩状态。

```
kubectl get hpa ingress-hpa
```

预期输出：

```
NAME          REFERENCE                          TARGETS      MINPODS   MAXPODS   REPLICAPS   AGE
ingress-hpa   Depolyment/nginx-deployment-basic   21/10 (avg)   2         10        10          7m49s
```

当REPLICAS的值和MAXPODS的值相同时，说明伸缩成功。

常见问题

- 问：如果执行 kubectl get hpa 后发现target一栏为<unknow>怎么办？
  答：请按照以下操作解决。
  i. 执行 kubectl describe hpa <hpa_name> ，确认HPA失效的原因。
  - 如果Conditions字段提示AbleToScale为False，请确认Deployment是否正常部署。
  - 如果Conditions字段提示ScalingActive为False，请继续下一步。

ii. 执行 `kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1/"` 。如果返回为 `Error from server (NotFound): the server could not find the requested resource` ，请确认alibaba-cloud-metrics-adapter的启动状态。如果确认alibaba-cloud-metrics-adapter状态正常，请确认HPA指标是否是Ingress相关指标。如果是Ingress相关指标，您需要提前部署日志服务组件。更多信息，请参见Ingress访问日志分析与监控。

iii. 确认HPA指标填写正确。sls.ingress.route的值格式为 `<namespace>-<svc>-<port>` 。

- *namespace*为Ingress所在的命名空间。
- *svc*是Ingress对应的Service名称。
- *port*是Ingress对应Service的端口名称。

- 问：如何查找HPA支持的指标名称？
  答：请参见阿里云HPA指标， 以下列举为常用指标。

指标名称	描述	附加参数
sls_ingress_qps	指定的IngressRoute每秒查询率	sls.ingress.route
sls_ingress_latency_avg	所有请求的延迟	sls.ingress.route
sls_ingress_latency_p50	50%请求的延迟	sls.ingress.route
sls_ingress_latency_p95	95%请求的延迟	sls.ingress.route
sls_ingress_latency_p99	99%请求的延迟	sls.ingress.route
sls_ingress_latency_p9999	99.99%请求的延迟	sls.ingress.route
sls_ingress_inflow	Ingress的流入带宽	sls.ingress.route

15.7. 基于阿里云Prometheus指标的容器水平伸缩

默认HPA只支持基于CPU和内存的自动伸缩，并不能满足日常的运维需求。阿里云Prometheus监控全面对接开源Prometheus生态，支持类型丰富的组件监控，提供多种开箱即用的预置监控大盘，且提供全面托管的Prometheus服务。本文介绍如何将阿里云Prometheus指标转换成HPA可用的指标，从而为应用提供更加便捷的扩缩机制。

前提条件

在将阿里云Prometheus指标转换成HPA可用的指标前，您需要部署相关组件。

1. 部署阿里云Prometheus监控组件。

   有关部署阿里云Prometheus的详细步骤，请参见开启阿里云Prometheus监控。

2. 部署alibaba-cloud-metrics-adapter组件。

   有关部署alibaba-cloud-metrics-adapter组件详细步骤，请参见部署alibaba-cloud-metrics-adapter组件。

示例

本文举例如何配置alibaba-cloud-metrics-adapter，实现将阿里云Prometheus指标转换为HPA可用指标，并实现该指标自动伸缩。

1. 部署工作负载。

   i. 登录容器服务管理控制台。

   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

   iii. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

   iv. 在**无状态**页面，单击**使用YAML创建资源**。

v. 在**创建**页面，部署以下YAML文件创建一个名为sample-app的应用及对应的Service，然后单击**创建**。

> **说明** 此容器暴露出http_requests_total的指标用来标识访问次数。

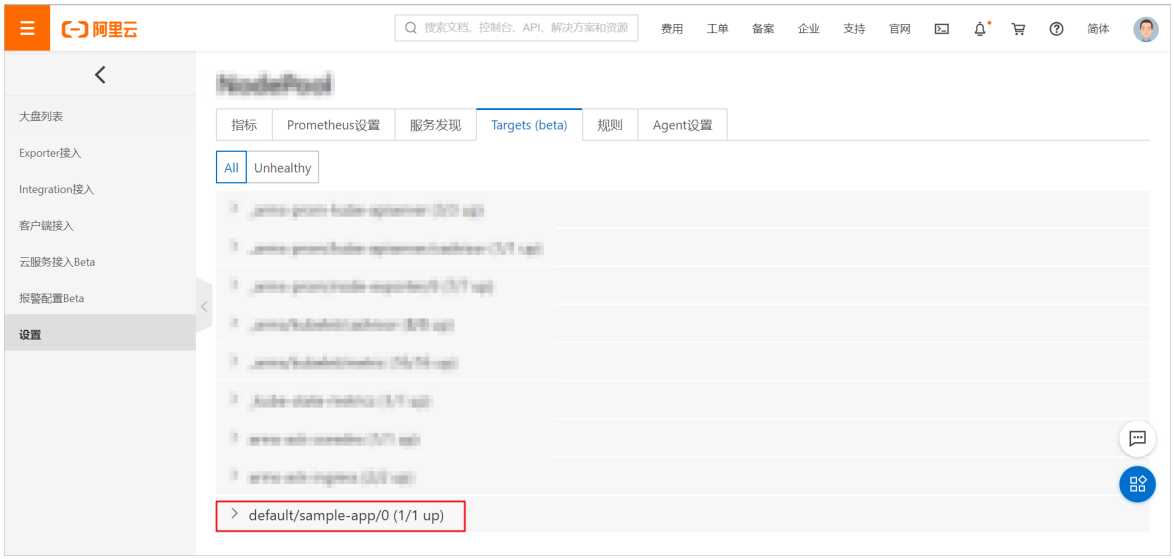
```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: sample-app
 labels:
  app: sample-app
spec:
 replicas: 1
 selector:
  matchLabels:
   app: sample-app
 template:
  metadata:
   labels:
    app: sample-app
  spec:
   containers:
   - image: luxas/autoscale-demo:v0.1.2
     name: metrics-provider
     ports:
     - name: http
       containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
 name: sample-app
 namespace: default
 labels:
  app: sample-app
spec:
 ports:
  - port: 8080
    name: http
    protocol: TCP
    targetPort: 8080
 selector:
  app: sample-app
 type: ClusterIP
```

2. 添加ServiceMonitor。
   i. 登录ARMS控制台。
   ii. 在左侧导航栏中单击**Prometheus监控**。
   iii. 在目标集群右侧的**操作**列单击**设置**。
   iv. 单击**服务发现**页签，然后单击ServiceMonitor页签，单击**添加ServiceMonitor**。

v. 复制以下YAML文件，然后单击**确定**。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
 name: sample-app
 namespace: default
spec:
 endpoints:
 - interval: 30s
  port: http
  path: /metrics
 namespaceSelector:
  any: true
 selector:
  matchLabels:
   app: sample-app
```

3. 确认监控状态。
   单击**Targets(beta)**，如果看到**default/sample-app/0(1/1 up)**，则说明您成功在阿里云Prometheus监控到了部署的应用。



4. 修改alibaba-cloud-metrics-adapter配置。
   i. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   ii. 在集群管理页左侧导航栏中，选择**应用 > Helm**。
   iii. 在**Helm**页面的**发布名称**列，找到alibaba-cloud-metrics-adapter，并单击右侧的**更新**。
   iv. 将alibaba-cloud-metrics-adapter中完整的YAML文件内容复制粘贴至模板中，然后单击**更新**。
   以下为部分YAML文件内容。

```
AlibabaCloudMetricsAdapter:
 affinity: {}
 commonLabels: ""
 env:
 - AccessKeyId: ""
 - AccessKeySecret: ""
 - Region: ""
 fullnameOverride: ""
 image:
  pullPolicy: IfNotPresent
  repository: registry.aliyuncs.com/acs/alibaba-cloud-metrics-adapter-amd64
  tag: v0.2.0-alpha-0c00ec2
 listenPort: 443
 nameOverride: ""
 nodeSelector: {}
 podAnnotations: {}
 prometheus:
  adapter:
   rules:
    custom:
    # 添加新的转换规则，请确保阿里云Prometheus中指标标签和此处一致，如果不一致，请照阿里云Prometheu
s中指标标签修改。
     - seriesQuery: 'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
       resources:
        overrides:
         kubernetes_namespace: {resource: "namespace"}
         kubernetes_pod_name: {resource: "pod"}
       name:
        matches: "^(.*)_total"
        as: "${1}_per_second"
       metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)'
    default: true
  #1. 这里设置为true，打开Prometheus adapter。
  enabled: true
  logLevel: 2
  metricsRelistInterval: 1m
  tls:
   ca: ""
   certificate: ""
   enable: false
   key: ""
  #2. 这里填入阿里云Prometheus控制的地址。
  url: http://arms-prometheus-proxy.aliyun.com:9090/api/v1/prometheus/8cba801fff65546a3012e9a6843
afd/1240538168824185/ce63742a509f948dda8ef18e75e703356/cn-shenzhen
 ramRoleType: __ACK_RAM_ROLE_TYPE__
 replicas: 1
 service:
  type: ClusterIP
 serviceAccountName: admin
 tolerations: []
```

有关ack-alibaba-cloud-adapter配置文件详细说明，请参见ack-alibaba-cloud-adapter配置文件详解。

5. （可选）检查HPA可用指标。

   执行以下命令查看HPA可用指标详情。

```
kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/*/http_requests_per_second" | jq .
```

```
{
 "kind": "MetricValueList",
 "apiVersion": "custom.metrics.k8s.io/v1beta1",
 "metadata": {
  "selfLink": "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/%2A/http_requests_per_second
"
 },
 "items": [
  {
   "describedObject": {
    "kind": "Pod",
    "namespace": "default",
    "name": "sample-app-579bc6774c-rmjfd",
    "apiVersion": "/v1"
   },
   "metricName": "http_requests_per_second",
   "timestamp": "2020-09-22T08:42:58Z",
   "value": "33m",
   "selector": null
  }
 ]
}
```

6. 使用以下YAML文件部署HPA，然后执行 `kubectl apply -f hpa.yaml` 创建HPA应用。

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
 name: sample-app
spec:
 scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: sample-app
 minReplicas: 1
 maxReplicas: 10
 metrics:
 - type: Pods
  pods:
   metricName: http_requests_per_second
   targetAverageValue: 500m
```

7. 执行压测验证。

```
ab -c 50 -n 2000 ClusterIP(sample-app):8080
```

执行以下命令查看HPA详情。

```
kubectl get hpa sample-app
```

预期输出：

```
NAME     REFERENCE         TARGETS   MINPODS  MAXPODS  REPLICAS  AGE
sample-app  Deployment/sample-app  33m/500m  1     10     1     72m
```

ack-alibaba-cloud-adapter配置文件详解

ack-alibaba-cloud-adapter通过以下步骤将Prometheus中的指标转换成HPA可用的指标：

1. Discovery：ack-alibaba-cloud-adapter会从Prometheus发现可用的指标。
2. Association：将指标与Kubernetes资源（Pod、Node、Namespace）相关联。
3. Naming：定义转换后的HPA可用指标名称。
4. Querying：定义查询Prometheus语句。

以上文中sample-app容器中暴露出来的 `http_requests_total` 指标转换成HPA中的 `http_requests_per_second` 为例，完整的ack-alibaba-cloud-adapter配置文件如下。

```
- seriesQuery: 'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
  resources:
    overrides:
      kubernetes_namespace: {resource: "namespace"}
      kubernetes_pod_name: {resource: "pod"}
  name:
    matches: "^(.*)_total"
    as: "${1}_per_second"
  metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)'
```

- Discovery
  指定待转换的Prometheus指标，您可以通过seriesFilters精确过滤指标。seriesQuery可以根据标签进行查找（示例代码如下）。

  ```
  seriesQuery: '{kubernetes_namespace!="",kubernetes_pod_name!=""}'
  seriesFilters:
    - isNot: "^container_.*_seconds_total"
  或者
  seriesQuery: 'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
  ```

  > **说明** seriesFilters为非必填项，用来过滤指标：
  > - is：<regex>，匹配包含该正则表达式的指标。
  > - isNot：<regex>，匹配不包含该正则表达式的指标。

- Association
  设置Prometheus指标标签与Kubernetes中的资源映射关系。`http_requests_total` 指标的标签包括 `kubernetes_namespace!=""` 和 `kubernetes_pod_name!=""` 。

  ```
  - seriesQuery: 'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
    resources:
      overrides:
        kubernetes_namespace: {resource: "namespace"}
        kubernetes_pod_name: {resource: "pod"}
  ```

- Naming
  用于将Prometheus指标名称转换成HPA的指标名称，但不会改变Prometheus本身的指标名称。如果使用Prometheus原来的指标，可以不设置。

> **说明** 您可以通过执行命令 `kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1"` 查看HPA可用的所有指标。

```
- seriesQuery: 'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
  resources:
    overrides:
      kubernetes_namespace: {resource: "namespace"}
      kubernetes_pod_name: {resource: "pod"}
  name:
    matches: "^(.*)_total"
    as: "${1}_per_second"
```

- Querying
  查询Prometheus API的模板。ack-alibaba-cloud-adapter会根据HPA中的参数，填充参数到此模板中，然后发送给Prometheus API请求，并将获得的值最终提供给HPA进行弹性扩缩。

```
- seriesQuery: 'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
  resources:
    overrides:
      kubernetes_namespace: {resource: "namespace"}
      kubernetes_pod_name: {resource: "pod"}
  name:
    matches: "^(.*)_total"
    as: "${1}_per_second"
  metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)'
```

相关文档

- 阿里云Prometheus监控
- 容器水平伸缩（HPA）

15.8. 使用ECI弹性调度

ECI弹性调度是阿里云提供的弹性调度策略。您可以在部署服务时通过添加Annotations来申明只使用ECS或ECI弹性资源，或者是在ECS资源不足时自动申请ECI资源。通过ECI弹性调度可以满足您在不同工作负载的场景下对弹性资源的不同需求。

更多信息，请参见使用ECI弹性调度。

15.9. 事件驱动弹性

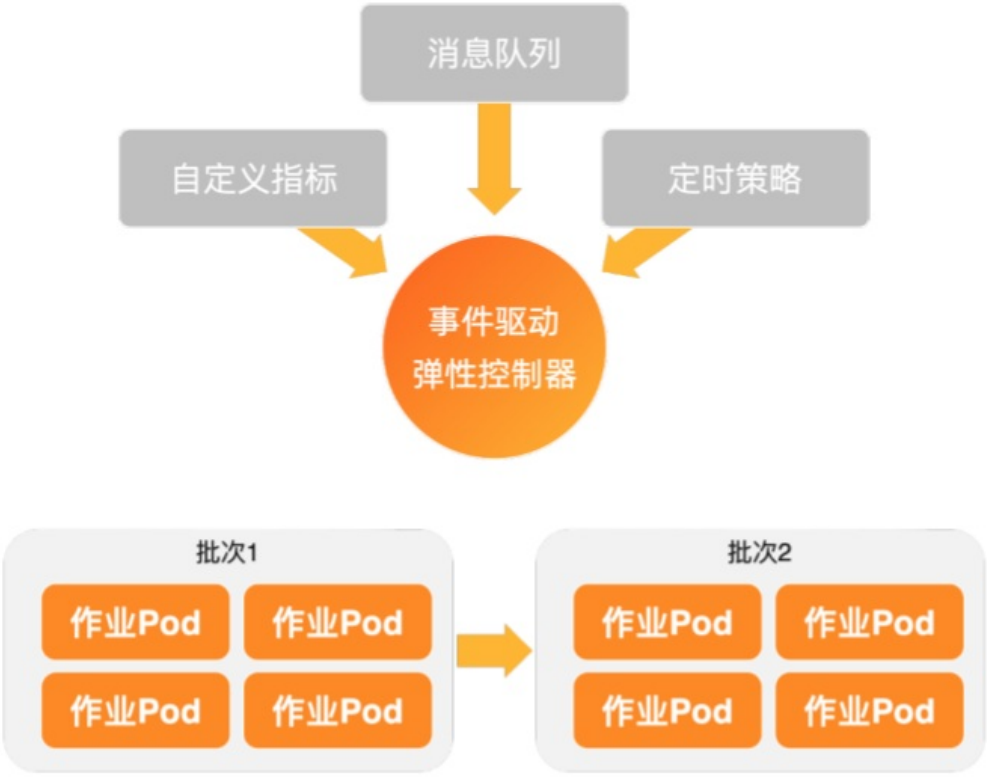
ACK支持在控制台上安装ack-keda，提供事件驱动弹性能力。本文介绍事件驱动弹性的概念、原理及如何使用事件驱动弹性。

事件驱动弹性概述

Kubernetes中，容器水平伸缩器HPA（Horizontal Pod Autoscaler）是最常用的应用弹性方案。容器水平伸缩的核心是基于资源利用率与预设的阈值水位之间的关系，来确认伸缩的计划。容器水平伸缩的方式具有使用简单、资源指标丰富等特点，但是它对于需要即时弹性的场景，尤其是对基于事件源进行离线作业支撑不足。ACK提供了ack-keda来提供事件驱动弹性能力，事件驱动弹性适用于音视频离线转码、事件驱动作业、流式数据处理等场景。

事件驱动弹性的原理

ACK通过增强版本的ack-keda来提供事件驱动弹性能力，下图是ack-keda的基本原理。



ack-keda会从事件源中进行数据的周期性消费。当消息出现堆积，即可秒级触发一个批次的离线任务伸缩。下一个周期到来时，会异步进行下一个批次的作业伸缩。ack-keda具有以下特性：

- 丰富的事件源支持
  ack-keda支持例如Kafka、MySQL、PostgreSQL、Rabbitmq、MongoDB等数据源。更多数据源，请参见RabbitMQ Queue。
- 离线任务的并发控制
  对于大规模的离线作业而言，底层管控的稳定性会面临比较大的挑战，需要提供资源、额度、API请求的整体控制。ack-keda提供了单批次、总批次的任务并发控制，保障系统的稳定性。
- 结束任务后自动清理元数据
  大规模离线作业执行完毕后，会留存大量的元数据信息。元数据信息的堆积会造成APIServer的稳定性下降，造成集群的性能下降、稳定性不足，甚至可能影响其他的业务。ack-keda会在任务执行结束后自动清理元数据，降低元数据的量级。

案例说明

在本案例中，准备了一个简单的转码作业。当有一个新任务到来的时候会向MongoDB插入一条类似下面的数据 {"type":"mp4","state":"waiting","createTimeStamp":"1610332940","fileName":"World and peace","endTimeStamp":"","uuid":"1fae72ff-3239-42f5-af97-04711d8007e8"} 。此时，容器服务的事件驱动控制器会从数据库中查询到状态为 "state":"waiting" 的任务，弹出与任务数目匹配的Job Pod来承载转码任务，完成转码业务并将数据中的 state 字段从之前的 waiting 修改成 finished 。同时Job完成后，将自动清理元数据，降低元数据对APIServer带来的压力，减轻开发者的负担。

步骤一：部署ack-keda

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面右上角搜索框中搜索ack-keda，找到然后单击ack-keda。
4. 在**应用目录 - ack-keda**页面**创建**面板选择目标集群，单击**创建**。

在控制台左侧导航栏单击**集群**，在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。在集群管理页面选择**工作负载 > 无状态**。在**无状态**页签左上角选择**命名空间**为kube-system，可以看到创建的ack-keda。

步骤二：部署基于mongoDB事件源驱动弹性示例

1. 部署Mongo DB。

   如果您已有MongoDB服务，可跳过此步骤。

   > **注意** 此数据库只用于测试，请勿用于生产环境。

   i. 创建*mongoDB.yaml*。

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
    name: mongodb
   spec:
    replicas: 1
    selector:
     matchLabels:
      name: mongodb
    template:
     metadata:
      labels:
       name: mongodb
     spec:
      containers:
      - name: mongodb
       image: mongo:4.2.1
       imagePullPolicy: IfNotPresent
       ports:
       - containerPort: 27017
        name: mongodb
        protocol: TCP
   ---
   kind: Service
   apiVersion: v1
   metadata:
    name: mongodb-svc
   spec:
    type: ClusterIP
    ports:
    - name: mongodb
     port: 27017
     targetPort: 27017
     protocol: TCP
    selector:
     name: mongodb
   ```

   ii. 部署Mongo DB到集群的mongodb命名空间中。

   ```
   kubectl apply -f mongoDB.yaml -n mongodb
   ```

2. 登录数据库，注册用户。

i. 新建用户。

```
//新建用户
kubectl exec -n mongodb mongodb-xxxxx -- mongo --eval 'db.createUser({ user:"test_user",pwd:"test_password",roles:[{ role:"readWrite", db: "test"}]})'
```

ii. 登录数据库。

```
//登录认证
kubectl exec -n mongodb mongodb-xxxxx -- mongo --eval  'db.auth("test_user","test_password")'
```

iii. 创建Collection。

```
//创建Collection
kubectl exec -n mongodb mongodb-xxxxx -- mongo --eval  'db.createCollection("test_collection")'
```

3. 部署TriggerAuthentication和ScaledJob。

在ack-keda中对于事件源的登录认证需要使用TriggerAuthentication。例如对于MongoDB事件源，TriggerAuthentication中的 `secretTargetRef` 字段会将指定Secret中的连接方式读取到ack-keda中，完成对MongoDB的登录认证。

i. 创建*auth.yaml*。

```
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
 name: mongodb-trigger
spec:
 secretTargetRef:
  - parameter: connectionString
    name: mongodb-secret
    key: connect
---
apiVersion: v1
kind: Secret
metadata:
 name: mongodb-secret
type: Opaque
data:
 connect: bW9uZ29kYjovL3Rlc3RfdXNlcjp0ZXN0X3Bhc3N3b3JkQG1vbmdvZGItc3ZjLm1vbmdvZGIuc3ZjLmNsdXN0ZXIubG9jYWw6MjcwMTcvdGVzdA==
```

ii. 部署TriggerAuthentication到集群的mongodb-test命名空间中。

```
kubectl apply -f auth.yaml -n mongodb-test
```

4. 部署ScaledJob。

ScaledJob主要用于配置Job模板以及指定查询的数据库及查询表达式等。以下示例配置的是从test数据库中的test_collection中，查询满足 `{"type":"mp4","state":"waiting"}` 的待转码数据。

i. 创建*job.yaml*。

```
apiVersion: keda.sh/v1alpha1
kind: ScaledJob
metadata:
 name: mongodb-job
spec:
 jobTargetRef:
  // Job模板配置
  template:
   spec:
    containers:
     - name: mongo-update
       image: registry.cn-hangzhou.aliyuncs.com/carsnow/mongo-update:v6
       args:
        - --connectStr=mongodb://test_user:test_password@mongodb-svc.mongodb.svc.cluster.local:270
17/test
        - --dataBase=test
        - --collection=test_collection
       imagePullPolicy: IfNotPresent
    restartPolicy: Never
  backoffLimit: 1
 pollingInterval: 15
 maxReplicaCount: 5
 successfulJobsHistoryLimit: 0
 failedJobsHistoryLimit: 10
 triggers:
  - type: mongo
    metadata:
     dbName: test                    //要查询的数据库
     collection: test_collection          //要查询的collection
     query: '{"type":"mp4","state":"waiting"}' //会对查询转码类型为mp4且状态是waiting的数据拉起job进行处
理
     queryValue: "1"
    authenticationRef:
     name: mongodb-trigger
```

`query`：配置数据条目。当ack-keda查询到Mongo数据库中有满足该条件的数据条目时，将启动Job资源。

ii. 部署ScaledJob到集群的mongodb-test命名空间中。

```
kubectl apply -f scaledJob.yaml -n mongodb-test
```

5. 插入5条待转码数据。

```
//插入5条待转码数据
kubectl exec -n mongodb mongodb-xxxxx -- mongo --eval 'db.test_collection.insert([
 {"type":"mp4","state":"waiting","createTimeStamp":"1610352740","fileName":"My Love","endTimeStamp":"
","uuid":"1gae72ff-3239-42f5-af97-04711d8007e8"},
 {"type":"mp4","state":"waiting","createTimeStamp":"1610350740","fileName":"Harker","endTimeStamp":""
,"uuid":"1gae72ff-3239-42f5-af97-04711d8007e8"},
 {"type":"mp4","state":"waiting","createTimeStamp":"1610152940","fileName":"The World","endTimeStamp
":"","uuid":"1gae72ff-3239-42f5-af97-04711d87767e8"},
 {"type":"mp4","state":"waiting","createTimeStamp":"1610390740","fileName":"Mother","endTimeStamp":"
","uuid":"1gae72ff-3239-42f5-af97-04799d8007e8"},
 {"type":"mp4","state":"waiting","createTimeStamp":"1610344740","fileName":"Jagger","endTimeStamp":""
,"uuid":"1gae72ff-3239-42f5-af97-04711d80099e8"},
 ])'
```

步骤三：验证事件驱动弹性

执行以下命令，查看Job动态。

```
//watch job
watch -n 1 kubectl get job -n mongodb-test
```

```
Every 1.0s: kubectl get job -n mongodb-test                    iZt4nbpmuvh2hxnjuqdmr4Z: Sun Jan 10 22:09:45 2021

NAME               COMPLETIONS   DURATION   AGE
mongodb-job-2qv7m   1/1           1s         3s
mongodb-job-f2ccp   0/1           3s         3s
mongodb-job-fbfs9   1/1           1s         3s
mongodb-job-kjrmd   0/1           3s         3s
mongodb-job-p48w7   0/1           3s         3s
```

可以看到成功扩展出5个Job。此时再登录数据库，观察转码业务状态，可以看到数据状态已经从 `waiting` 变成了 `finished` 。

```
  db.test_collection.find()
{ "_id" : ObjectId("5ffd37a95217f9cfe[redacted]") }
{ "_id" : ObjectId("5ffd38105217f9cfe[redacted]"), "type" : "mp4", "state" : "finished", "createTimeStamp" : "1610352740", "fileName" : "M
y Love", "endTimeStamp" : "", "uuid" : "1gae72ff-3239-42f5-af97-04711[redacted]" }
{ "_id" : ObjectId("5ffd38335217f9cfe[redacted]"), "type" : "mp4", "state" : "finished", "createTimeStamp" : "1610350740", "fileName" : "H
arker", "endTimeStamp" : "", "uuid" : "1gae72ff-3239-42f5-af97-04711[redacted]" }
{ "_id" : ObjectId("5ffd38585217f9cfe[redacted]"), "type" : "mp4", "state" : "finished", "createTimeStamp" : "1610152940", "fileName" : "T
he World", "endTimeStamp" : "", "uuid" : "1gae72ff-3239-42f5-af97-04711d8[redacted]" }
{ "_id" : ObjectId("5ffd38735217f9cfe[redacted]"), "type" : "mp4", "state" : "finished", "createTimeStamp" : "1610390740", "fileName" : "M
other", "endTimeStamp" : "", "uuid" : "1gae72ff-3239-42f5-af97-04799d8[redacted]" }
{ "_id" : ObjectId("5ffd388a5217f9cfe7[redacted]"), "type" : "mp4", "state" : "finished", "createTimeStamp" : "1610344740", "fileName" : "J
agger", "endTimeStamp" : "", "uuid" : "1gae72ff-3239-42f5-af97-04711d[redacted]" }
>
```

15.10. 弹性伸缩FAQ

本文介绍容器服务ACK弹性伸缩的常见问题及解决办法。

类型	问题
水平伸缩（HPA）常见问题	• HPA的监控数据current字段为何显示为unknown? • 为何HPA在滚动发布时出现扩容多余Pod? • HPA到达阈值为何不进行扩缩容? • HPA采集周期如何配置? • CronHPA是否兼容HPA? • 如何解决HPA启动时CPU或内存飙高造成扩容出多余Pod的多弹现象?

类型	问题
基于组件指标的水平伸缩常见问题	• 如果执行kubectl get hpa后发现target一栏为<unknown>怎么办？ • 如何查找HPA支持的指标名称？
节点自动伸缩常见问题	• 为什么节点自动伸缩组件无法弹出节点？ • 为什么节点自动伸缩组件无法缩容节点？ • 多个伸缩组在弹性伸缩的时候是如何被选择的？

HPA的监控数据current字段为何显示为*unknown*?

当HPA的监控数据的current字段显示为*unknown*时，表示kube-controller-manager无法访问监控数据源获取对应的监控数据。

```
Name:                           kubernetes-tutorial-deployment
Namespace:                      default
Labels:                         <none>
Annotations:                    <none>
CreationTimestamp:              Mon, 10 Jun 2019 11:46:48 0530
Reference:                      Deployment/kubernetes-tutorial-deployment
Metrics:                        ( current / target )
 resource cpu on pods  (as a percentage of request):  <unknown> / 2%
Min replicas:                   1
Max replicas:                   4
Deployment pods:                1 current / 0 desired
Conditions:
 Type       Status  Reason            Message
 ----       ------ ------            -------
 AbleToScale   True   SucceededGetScale      the HPA controller was able to get the target's current scale
 ScalingActive  False  FailedGetResourceMetric  the HPA was unable to compute the replica count: unable to get me
trics for resource cpu: unable to fetch metrics from resource metrics API: the server is currently unable to handle th
e request (get pods.metrics.k8s.io)
Events:
 Type    Reason            Age             From              Message
 ----    ------            ----            ----              -------
 Warning  FailedGetResourceMetric  3m3s (x1009 over 4h18m)  horizontal-pod-autoscaler  unable to get metrics for
resource cpu: unable to fetch metrics from resource metrics API: the server is currently unable to handle the reques
t (get pods.metrics.k8s.io)
```

原因如下：

- 原因一：resource metrics数据源无法使用。先执行命令 `kubectl top pod` 检查是否返回数据。如果所有的Pod都无数据，请执行 `kubectl get apiservice` 检查当前提供resource metrics的数据源的情况。返回的示例数据如下。

```
NAME                     SERVICE           AVAILABLE  AGE
v1.                      Local             True     29h
v1.admissionregistration.k8s.io     Local             True     29h
v1.apiextensions.k8s.io          Local             True     29h
v1.apps                  Local             True     29h
v1.authentication.k8s.io         Local             True     29h
v1.authorization.k8s.io          Local             True     29h
v1.autoscaling           Local             True     29h
v1.batch                 Local             True     29h
v1.coordination.k8s.io           Local             True     29h
v1.monitoring.coreos.com         Local             True     29h
v1.networking.k8s.io             Local             True     29h
v1.rbac.authorization.k8s.io     Local             True     29h
v1.scheduling.k8s.io             Local             True     29h
v1.storage.k8s.io        Local             True     29h
v1alpha1.argoproj.io             Local             True     29h
v1alpha1.fedlearner.k8s.io       Local             True     5h11m
v1beta1.admissionregistration.k8s.io  Local             True     29h
v1beta1.alicloud.com             Local             True     29h
v1beta1.apiextensions.k8s.io     Local             True     29h
v1beta1.apps             Local             True     29h
v1beta1.authentication.k8s.io    Local             True     29h
v1beta1.authorization.k8s.io     Local             True     29h
v1beta1.batch            Local             True     29h
v1beta1.certificates.k8s.io      Local             True     29h
v1beta1.coordination.k8s.io      Local             True     29h
v1beta1.events.k8s.io            Local             True     29h
v1beta1.extensions       Local             True     29h
...
[v1beta1.metrics.k8s.io          kube-system/metrics-server  True     29h]
...
v1beta1.networking.k8s.io        Local             True     29h
v1beta1.node.k8s.io              Local             True     29h
v1beta1.policy           Local             True     29h
v1beta1.rbac.authorization.k8s.io    Local             True     29h
v1beta1.scheduling.k8s.io        Local             True     29h
v1beta1.storage.k8s.io           Local             True     29h
v1beta2.apps             Local             True     29h
v2beta1.autoscaling              Local             True     29h
v2beta2.autoscaling              Local             True     29h
```

如果v1beta1.metrics.k8s.io所对应的apiservice不是*kube-system/metrics-server*，检查是否由于安装Prometheus Operator覆盖导致。如果是覆盖导致的问题，可以通过部署以下的YAML模板进行恢复。

```
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
```

如果非上述问题，请参见metrics-server。

- 原因二：滚动发布或者扩容时无法获取数据。
  默认metrics-server的采集周期是1秒。刚扩容或更新完成后，metric-server会有一段时间无法获取监控数据。请于扩容或更新后2秒左右进行查看。
- 原因三：缺少request字段设置。
  HPA默认是通过**实际的利用率/request**作为利用率的数值，因此可以检查Pod的Resource字段中是否包含Request字段。

为何HPA在滚动发布时出现扩容多余Pod?

社区的Controller Manager会在滚动发布的时候，对于没有监控数据的Pod，进行监控数据的补零操作，从而有一定的概率出现扩容出多余的Pod现象（多弹现象）。您可以通过升级ACK提供的最新版metrics-server，并在metrics-server的启动参数上开启开关防止多弹。

```
##在metrics-server的启动参数中加入以下选项。
--enable-hpa-rolling-update-skipped=true
```

HPA到达阈值为何不进行扩缩容?

HPA的扩缩容的触发条件不仅是CPU利用率或内存使用率超过阈值与低于阈值，需要额外考虑的是如果扩容或者缩容后，是否会再次触发缩容或者扩容，减少震荡的场景。

HPA采集周期如何配置?

对版本号大于v0.2.1-b46d98c-aliyun的metric-server，在metric-server的启动参数中设置`--metric_resolution`，例如`--metric_resolution=15s`即可。

CronHPA是否兼容HPA?

CronHPA可以兼容HPA。阿里云容器服务ACK将CronHPA中的scaleTargetRef设置为HPA对象，然后通过HPA对象来寻找真实的scaleTargetRef，从而让CronHPA感知HPA的当前状态。CronHPA不会直接调整Deployment的副本数目，而是通过HPA来操作Deployment，这样可以避免HPA和CronHPA的冲突问题。关于CronHPA兼容HPA的更多信息，请参见容器定时伸缩（CronHPA）。

如何解决HPA启动时CPU或内存飙高造成扩容出多余Pod的多弹现象?

对于Java等需要预热的语言与框架而言，在容器刚启动的时候，有可能会出现分钟级别的CPU、内存飙高，其可能会造成HPA的误触发。您可以通过升级ACK提供的最新版metrics-server组件，并在Pod的Annotation上增加开关防止误触发。关于如何升级metrics-server组件，请参见升级Kubernetes集群的metrics-server组件。

增加开关防止误触发部署示例YAML如下。

```
## 以Deployment为例
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment-basic
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    HPAScaleUpDelay: 3m # m代表分钟，表示HPA在Pod创建3分钟后开始生效，可选单位为[s(秒),m(分)]。
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9 # Replace it with your exactly <image_name:tags>.
    ports:
    - containerPort: 80
```

节点自动伸缩常见问题

- 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。
- 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*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。
  - 节点上的Pod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。
- 多个伸缩组在弹性伸缩的时候是如何被选择的？
  在Pod处在无法调度时，会触发弹性伸缩组件的模拟调度逻辑，会根据伸缩组配置的标签和污点以及实例规格等信息进行判断。当配置的伸缩组可以模拟调度Pod的时候，就会被选择进行节点弹出。当同时有多个伸缩组满足模拟调度条件的时候，默认采用的是最少浪费原则，即根据模拟弹出后节点上剩余的资源最小为原则进行抉择。

基于阿里云组件指标的容器水平伸缩常见问题

- 问：如果执行 `kubectl get hpa` 后发现target一栏为<unknow>怎么办？
  答：请按照以下操作解决。
  i. 执行 `kubectl describe hpa <hpa_name>` ，确认HPA失效的原因。
     - 如果Conditions字段提示AbleToScale为False，请确认Deployment是否正常部署。
     - 如果Conditions字段提示ScalingActive为False，请继续下一步。

ii. 执行 `kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1/"` 。如果返回为 `Error from server (NotFound): the server could not find the requested resource` ，请确认alibaba-cloud-metrics-adapter的启动状态。如果确认alibaba-cloud-metrics-adapter状态正常，请确认HPA指标是否是Ingress相关指标。如果是Ingress相关指标，您需要提前部署日志服务组件。更多信息，请参见Ingress访问日志分析与监控。

iii. 确认HPA指标填写正确。sls.ingress.route的值格式为 `<namespace>-<svc>-<port>` 。

- *namespace*为Ingress所在的命名空间。
- *svc*是Ingress对应的Service名称。
- *port*是Ingress对应Service的端口名称。

- 问：如何查找HPA支持的指标名称？
  答：请参见阿里云HPA指标， 以下列举为常用指标。

指标名称	描述	附加参数
sls_ingress_qps	指定的IngressRoute每秒查询率	sls.ingress.route
sls_ingress_latency_avg	所有请求的延迟	sls.ingress.route
sls_ingress_latency_p50	50%请求的延迟	sls.ingress.route
sls_ingress_latency_p95	95%请求的延迟	sls.ingress.route
sls_ingress_latency_p99	99%请求的延迟	sls.ingress.route
sls_ingress_latency_p9999	99.99%请求的延迟	sls.ingress.route
sls_ingress_inflow	Ingress的流入带宽	sls.ingress.route

相关文档

- metrics-server
- 容器水平伸缩（HPA）
- 容器定时伸缩（CronHPA）
- 升级Kubernetes集群的metrics-server组件

16.发布

16.1. 灰度发布

16.1.1. 灰度发布概述

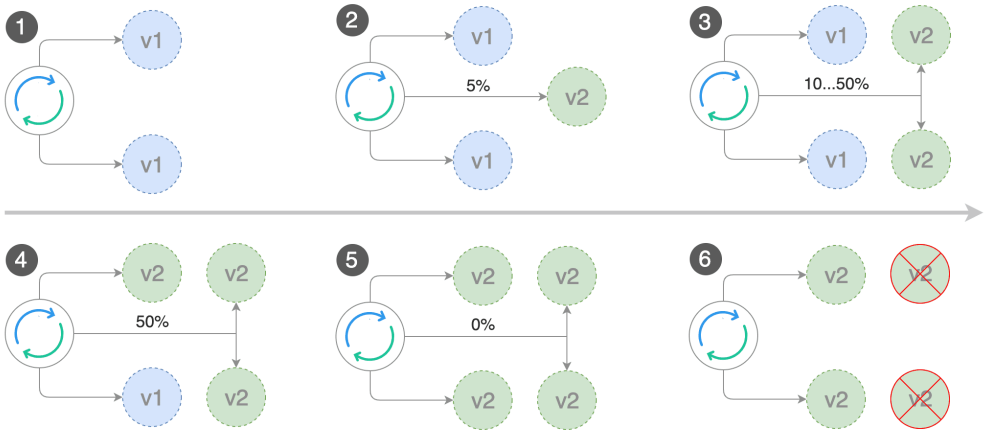
灰度发布是可以帮助您渐进式更新Deployment的工具。它能结合Ingress Controller实现多版本共存，发布暂停，流量百分比切换，健康检查等功能，极大解放灰度发布过程中的手动操作，全自动化实现线上灰度流量切换。本文介绍灰度发布的概念和如何安装灰度发布组件。

什么是灰度发布？

灰度发布，又称金丝雀发布，是将应用的旧版本A与新版本B同时部署在环境中，业务请求可能会被路由到版本A的后端上，也可能会被路由到版本B的后端上。您可以自定义灰度发布策略，快速调整版本A和B的流量占比。

灰度发布可以在发布新版本应用时，自定义控制新版本应用流量比重，渐进式完成新版本应用的全量上线，最大限度地控制新版本发布带来的业务风险，降低故障带来的影响面，同时支持快速回滚。

灰度发布流程图



安装灰度发布组件

您需要在ACK中安装对应的组件后才可以使用灰度发布功能。在**灰度发布**页面，直接通过单击**开始安装**即可实现对应的组件安装。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > 灰度发布（公测）**。
5. 单击**开始安装**。
   完成灰度发布组件安装后，控制台自动跳转到灰度发布页面。



16.1.2. 创建灰度发布

灰度发布是可以帮助您渐进式更新Deployment的工具。它能结合Ingress Controller实现多版本共存，发布暂停，流量百分比切换，健康检查等功能，极大解放灰度发布过程中的手动操作，全自动化实现线上灰度流量切换。本文介绍如何创建灰度发布。

前提条件

您已完成以下操作：

- 创建一个类型为Deployment的工作负载。有关具体操作，请参见创建无状态工作负载Deployment。
- 创建一个服务（Service）。有关具体操作，请参见管理服务。
- 创建一个路由（Ingress）。有关具体操作，请参见创建Ingress路由。

创建灰度发布

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > 灰度发布（公测）**。
5. 在**灰度发布**页面右侧，单击**创建**。
6. 在**创建**面板，设置目标命名空间、Deployment、Service、Ingress和发布方式。

   有关发布方式特性的描述详情，请参见灰度发布的高级选项。
7. 单击**确定**。

相关文档

- 灰度发布概述
- 使用手动发布模式创建灰度发布
- 使用自动发布模式创建灰度发布

16.1.3. 灰度发布的高级选项

创建灰度发布时，您可以选择自动发布和手动发布两种灰度发布方式。文本介绍两种发布方式的高级选项特性。

背景信息

您在创建灰度发布过程中，可以设置灰度发布的方式。有关创建灰度发布的步骤，请参见创建灰度发布。

手动发布

当发布方式被设置为**手动发布**时，您可以设置**钉钉通知地址**。



钉钉通知的主要功能是通过钉钉机器人的方式通知发布过程中的一些关键事件。有关钉钉机器人的具体创建方式，请参见机器人开发。在设置钉钉机器人的安全设置时，您可以添加**自定义关键词**为*Canary*。设置完成后，将生成的钉钉机器人Webhook填入对应的钉钉通知地址。



自动发布

当发布方式被设置为**自动发布**时，您可以设置健康检查路径、允许失败次数、超时时间、钉钉通知地址等。

参数	描述
权重调整步长	切换新的流量百分比。例如设置此参数为20，则说明新流量每次增加20%。
健康检查路径	在自动发布的过程中，填入一个URL地址。允许灰度发布控制器可以通过这个URL检查网络的连通性。此健康检查必须是一个HTTP模式的URL，且返回的HTTP Code必须是2xx才会认为是通过，否则就认为失败。
允许失败次数	需要配合健康检查路径来使用。如果健康检查失败次数超过了允许失败的次数，就会自动回滚。
超时时间	如果在自动发布过程中，新建Pod出现无法启动，或者健康检查卡住无法返回时超过设定的超时时间，就会认为发布失败。如果超过了允许失败的次数，系统就会触发自动回滚。
钉钉通知地址	生成的钉钉机器人Webhook。将Webhook填入到对应的钉钉通知地址。有关详情，请参见上述手动发布。

相关文档

- 创建灰度发布

16.1.4. 使用手动发布模式创建灰度发布

在使用灰度发布的手动发布方式的时候，您需要创建一个Ingress、Service和Deployment结构的应用。目前灰度发布只支持基于Deployment和Ingress的灰度发布。本文举例介绍使用手动发布方式的灰度发布流程。

步骤一：创建测试应用

本文使用以下YAML文件模板作为测试示例。该示例使用Deployment部署一个Web应用，并通过路由（Ingress）向外暴露服务（Service）。有关如何使用YAML模板创建应用的具体步骤，请参见通过编排模板创建Linux应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: podinfo
 labels:
  app: podinfo
spec:
 replicas: 4
 minReadySeconds: 5
 revisionHistoryLimit: 5
 progressDeadlineSeconds: 60
 strategy:
  rollingUpdate:
   maxUnavailable: 1
  type: RollingUpdate
 selector:
  matchLabels:
   app: podinfo
 template:
  metadata:
   labels:
    app: podinfo
  spec:
   containers:
   - name: podinfod
     image: registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green
     imagePullPolicy: IfNotPresent
     ports:
      - name: http
        containerPort: 8080
        protocol: TCP
     readinessProbe:
      tcpSocket:
       port: 8080
      initialDelaySeconds: 5
      timeoutSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
 name: my-podinfo-svc
spec:
 selector:
  app: podinfo
 ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: podinfo
 labels:
  app: podinfo
spec:
 rules:
  - host: app.example.com
```

```
  - host: app.example.com
    http:
      paths:
        - backend:
            serviceName: my-podinfo-svc
            servicePort: 80
```

步骤二：创建灰度发布

创建灰度发布时，设置上述创建的Deployment、Service、Ingress，然后设置发布方式为**手动发布**。有关创建灰度发布的具体操作，请参见创建灰度发布。

完成创建灰度发布后，单击创建的灰度发布名称，发布状态显示**初始化完毕**。如果显示**初始化中**的话，可以多次单击**刷新**。



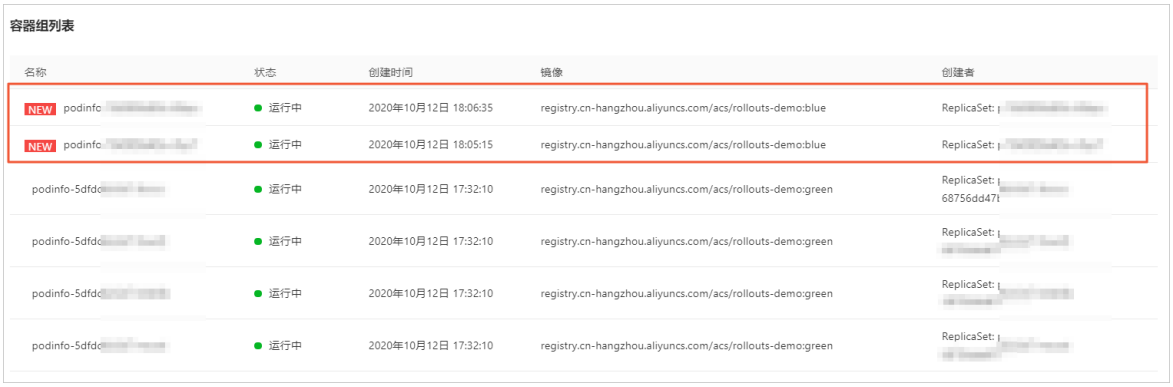
步骤三：执行灰度发布

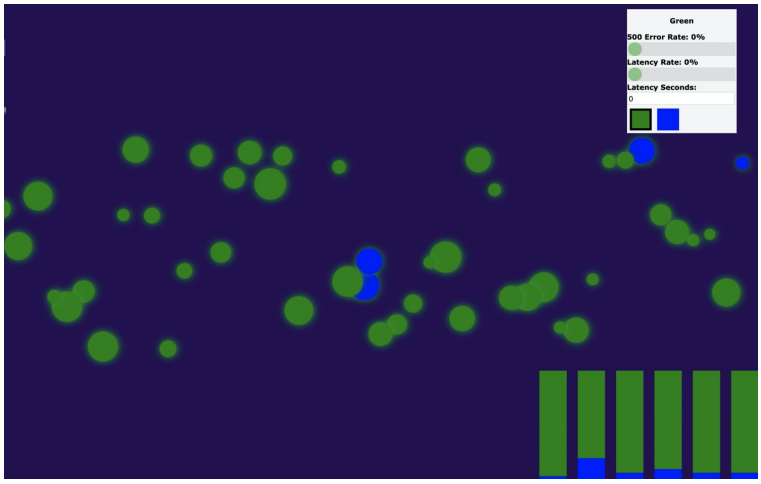
初始化完毕后，就可以进行灰度发布。开始灰度发布前，您需要先更新一下应用。本文示例以更新镜像tag为例，修改镜像tag中的green为blue。先发布两个Pod为新版本，同时将新版本的灰度流量设置为10%。

1. 创建灰度发布。
2. 单击目标灰度发布名称，然后在详情页面，单击**发布对象**名称后面的编辑图标。
3. 在**查看YAML**面板中，将green修改为blue，然后单击**确定**。



4. 在灰度发布**基本信息**区域，编辑**当前权重[0-100]**为*10*，**新版容器组副本数[0-4]**为*2*。
   可以看到，新版本的Pod已经创建。这时候访问应用观察流量情况，可以看到流量中绿色占比90%，蓝色占比10%。





5. 根据业务的实际情况，不断调整新旧版容器数量，将旧版Pod缩容，新版Pod扩容，然后将更多的流量导入到新版本的Pod中。当测试完毕且没有问题时，单击页面右上方的**确认发布**，完成此次发布。

   发布成功后，可以查看流量情况。当业务的所有流量都转向了新的Pod时，就完成了一次应用的发布。您可以单击页面右上方的**完成**，结束此次发布动作。

查看历史记录

每次发布完成后，您可以查看发布的记录：

- 在**灰度发布**页面的**历史记录**区域，您可以查看所有创建的灰度发布的发布记录。
- 在每个创建的灰度发布详情页面下方的**发布日志**区域，查看发布的过程。

相关文档

- 使用自动发布模式创建灰度发布

16.1.5. 使用自动发布模式创建灰度发布

ACK的灰度发布功能除了手动发布方式外，还提供了自动发布方式。自动发布模式可以让您全自动化地完成灰度发布流程。

步骤一：创建测试应用

本文使用以下YAML文件模板作为测试示例。该示例使用Deployment部署一个Web应用，并通过路由（Ingress）向外暴露服务（Service）。有关如何使用YAML模板创建应用的具体步骤，请参见通过编排模板创建Linux应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: podinfo
  labels:
    app: podinfo
```

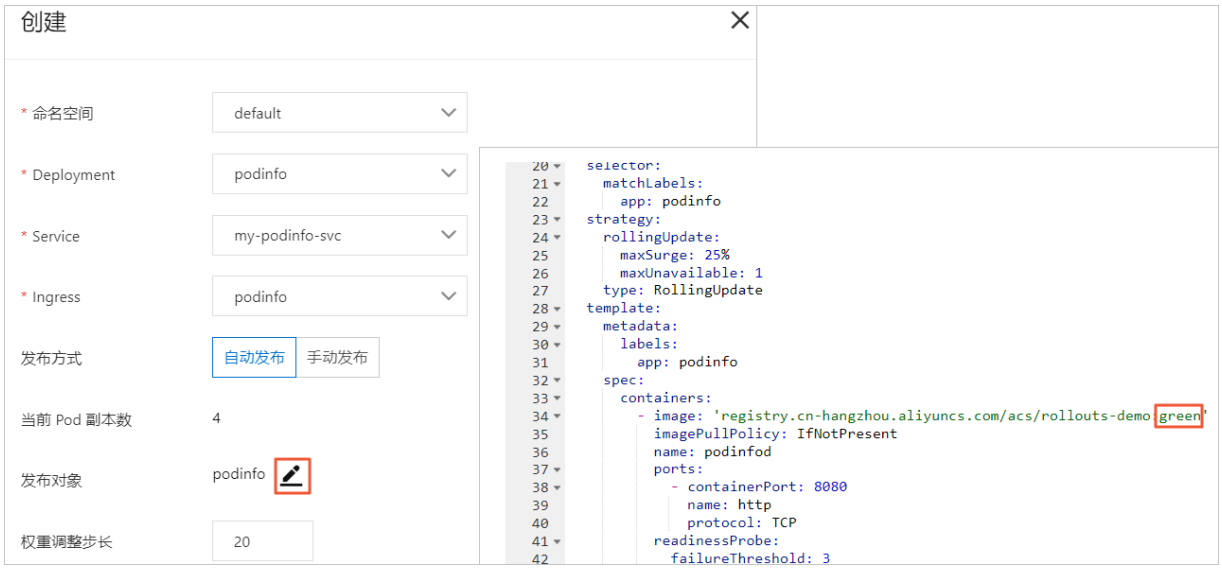
```
spec:
 replicas: 4
 minReadySeconds: 5
 revisionHistoryLimit: 5
 progressDeadlineSeconds: 60
 strategy:
  rollingUpdate:
   maxUnavailable: 1
  type: RollingUpdate
 selector:
  matchLabels:
   app: podinfo
 template:
  metadata:
   labels:
    app: podinfo
  spec:
   containers:
   - name: podinfod
    image: registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green
    imagePullPolicy: IfNotPresent
    ports:
     - name: http
       containerPort: 8080
       protocol: TCP
    readinessProbe:
     tcpSocket:
      port: 8080
     initialDelaySeconds: 5
     timeoutSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
 name: my-podinfo-svc
spec:
 selector:
  app: podinfo
 ports:
  - protocol: TCP
   port: 80
   targetPort: 8080
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: podinfo
 labels:
  app: podinfo
spec:
 rules:
  - host: app.example.com
   http:
    paths:
     - backend:
      service:
       name: my-podinfo-svc
```

```
      port:
        number: 80
  pathType: ImplementationSpecific
```

步骤二：创建灰度发布

创建灰度发布时，设置上述创建的Deployment、Service、Ingress，然后设置发布方式为**自动发布**。有关创建灰度发布的具体操作，请参见创建灰度发布。

> **注意**
> 您在配置自动发布模式时，需要注意以下两点：
>
> - 和手动发布不同，使用自动发布模式时，您需要在创建灰度发布的时候编辑**发布对象**里面的YAML以设置需要更新的新版本Deployment。本文示例将发布的版本镜像blue修改成green。
> - 您需要设置**权重调整步长**。该值设置每次以多少步长来调整新版本的权重流量。本文以20为例，起初会以0%的流量切至新版本。随着新版本的Pod创建出来后，第一次切换20%的流量至新版本Pod，然后随着新版本的Pod数量变多，流量逐步切换到40%，60%，80%，直到100%。



完成创建灰度发布后，发布流程会全自动化地进行。当一个新版本的Pod创建出来后，流量切换20%至新版本Pod，此时旧版本的一个Pod会被删除，然后系统会再继续创建一个新的Pod删除一个旧版Pod，以此类推，直到完毕。

相关文档

- 使用手动发布模式创建灰度发布

16.2. 使用OpenKruise部署云原生应用

OpenKruise是基于Kubernetes的一个标准扩展组件，可以配合原生Kubernetes使用，高效管理应用容器、Sidecar及镜像分发等功能。本文介绍如何使用OpenKruise部署云原生应用。

前提条件

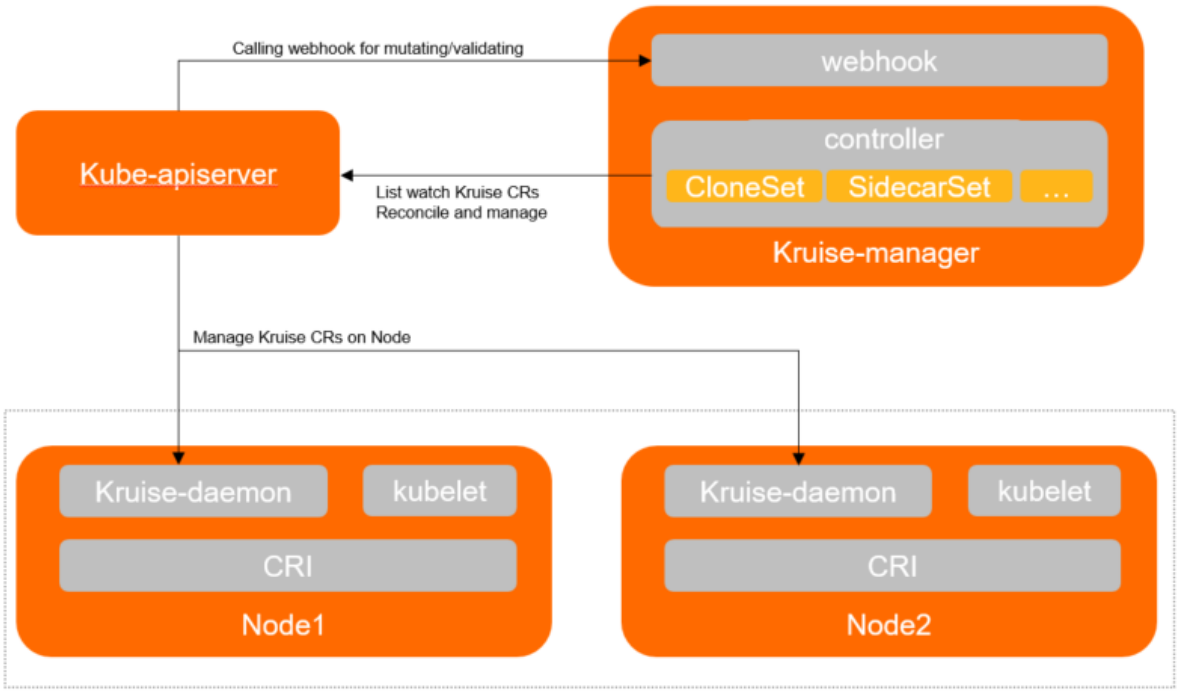
已安装Kubernetes集群，且集群版本不低于1.13。具体操作，请参见创建Kubernetes托管版集群。

背景信息

OpenKruise是阿里云开源的云原生应用自动化引擎，也是阿里巴巴经济体上云全面使用的部署基座，已正式加入CNCF Sandbox。

OpenKruise包含了多种自定义Workload，用于无状态应用、有状态应用、Sidecar容器、Daemon应用等部署管理，提供了原地升级、灰度、流式、配置优先级等扩展策略。

组件架构



OpenKruise是Kubernetes的一个标准扩展，所以也可以原生的部署到K8s集群当中，主要包含以下两个组件：

组件	说明
Kruise-manager	Kruise-manager是一个运行Controller和Webhook的中心组件，通过Deployment部署在kruise-system命名空间中。通过Controller以及Webhook实现原地升级及Sidecar管理等核心能力。
Kruise-daemon	通过DaemonSet部署到每个节点上，提供镜像预热及容器重启等功能。

使用说明

OpenKruise包含CloneSet、Advanced StatefulSet、Advanced DaemonSet等控制器。以下介绍常用控制器的功能。

常用控制器介绍

控制器	功能	推荐指数
CloneSet	管理无状态应用，对标Kubernetes原生Deployment。关于CloneSet的详细介绍，请参见Cloneset。 资源（YAML）的字段与Deployment不完全兼容，但功能上全覆盖，并提供比Deployment更丰富的策略。	推荐指数：☆☆☆☆☆
Advanced StatefulSet	管理有状态应用，对标Kubernetes原生StatefulSet。关于Advanced StatefulSet的详细介绍，请参见Advanced StatefulSet。 资源（YAML）字段与原生StatefulSet完全兼容，只需要把 `apiVersion` 改为 `apps.kruise.io/v1alpha1` ，另外提供了 `optional` 字段来扩展发布策略（原地升级、并行发布等）。	推荐指数：☆☆☆☆

控制器	功能	推荐指数
Advanced DaemonSet	管理Daemon应用，对标Kubernetes原生DaemonSet。关于Advanced DaemonSet的详细介绍，请参见Advanced DaemonSet。 资源（YAML）字段与原生DaemonSet完全兼容，只需要把 `apiVersion` 改为 `apps.kruise.io/v1alpha1` ，另外提供了 `optional` 字段来扩展发布策略（热升级、灰度、按Node标签灰度等）。	推荐指数：☆☆☆☆
SidecarSet	独立管理Sidecar容器和注入。关于SidecarSet的详细介绍，请参见SidecarSet。 在独立CR中定义Sidecar容器和Label Selector，OpenKruise会在所有符合Selector条件的Pod创建时注入定义好的Sidecar容器，并支持对已注入Sidecar容器做原地升级。	推荐指数：☆☆☆☆
UnitedDeployment	管理不同区域下的多个Sub Workload，关于UnitedDeployment的详细介绍，请参见UnitedDeployment。 目前支持将CloneSet、StatefulSet、Advanced StatefulSet作为Sub Workload，您可以用一个UnitedDeployment来定义不同区域中的Sub Workload部署Replicas。	推荐指数：☆☆☆

以下介绍CloneSet、Advanced StatefulSet、Advanced DaemonSet与社区对标控制器的功能对比。

与社区控制器功能对比

功能	CloneSet VS Deployment		Advanced StatefulSet VS StatefulSet		Advanced DaemonSet VS DaemonSet	
	CloneSet	Deployment	Advanced StatefulSet	StatefulSet	Advanced DaemonSet	DaemonSet
流式扩容	不支持（开源中，未来会支持）	不支持	不支持	不支持	支持	不支持
指定缩容	支持	不支持	支持	不支持	不支持	不支持
Pod重建升级	支持	支持	支持	支持	支持	支持
Pod原地升级	支持	不支持	支持	不支持	不支持（开源中，未来会支持）	不支持
分批灰度发布	支持	不支持	支持	支持	支持	不支持
最大可用数量	支持	支持	支持	不支持	支持	支持

功能	CloneSet VS Deployment		Advanced StatefulSet VS StatefulSet		Advanced DaemonSet VS DaemonSet	
	CloneSet	Deployment	Advanced StatefulSet	StatefulSet	Advanced DaemonSet	DaemonSet
最大弹性数量	支持	支持	不支持	不支持	支持	不支持
通过使用优先级策略和打散策略来自定义发布顺序	支持	不支持	支持	不支持	支持	不支持
通过lifecycle hook管理Pod生命周期	支持	不支持	不支持	不支持	不支持	不支持

安装OpenKruise

> **注意** 安装OpenKruise前，请确保Kubernetes版本不低于1.13。如果您使用的是Kubernetes 1.13或1.14版本，必须先在kube-apiserver中打开 `CustomResourceWebhookConversion` feature-gate。更多信息，请参见特性门控。

您可以通过ACK应用目录和组件管理安装OpenKruise。推荐通过ACK组件管理安装OpenKruise，以下为两种安装方式对比：

- 通过ACK组件管理安装：一键安装，后续也可通过组件管理升级及卸载OpenKruise。
- 通过ACK应用目录安装：一键安装，后续只能通过执行 `helm upgrade` 命令，升级OpenKruise版本。

通过ACK组件管理安装OpenKruise

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**组件管理**页面，单击**应用管理**页签。在ack-kruise区域，单击**安装**。

   在**提示**对话框确认组件信息后，单击**确定**。

通过ACK应用目录安装OpenKruise

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**阿里云应用**页签下单击**应用管理**，然后选择ack-kruise。
4. 在**应用目录 - ack-kruise**页面**创建**区域，选择安装的**集群**和**命名空间**设置**发布名称**，然后单击**创建**。

使用CloneSet部署无状态应用

1. 创建CloneSet。

i. 创建*cloneset.yaml*。

```
apiVersion: apps.kruise.io/v1alpha1
kind: CloneSet
metadata:
 name: demo-clone
spec:
 replicas: 5
 selector:
  matchLabels:
   app: guestbook
 template: #Pod template和Deployment的结构完全一致。
  metadata:
   labels:
    app: guestbook
  spec:
   containers:
   - name: guestbook
    image: registry.cn-hangzhou.aliyuncs.com/kruise-test/guestbook:v1
    env:
    - name: test
     value: foo
 updateStrategy:
  type: InPlaceIfPossible   #尽量使用原地升级，否则重建升级。
  maxUnavailable: 20%    #发布过程最多20%不可用。
  inPlaceUpdateStrategy:
   gracePeriodSeconds: 3  #每个Pod原地升级前Not Ready优雅等待时间。
```

- type：设置升级策略，支持以下3种升级方式。
  - ReCreate：控制器会删除旧Pod和PVC，然后用新版本重新创建Pod和PVC。
  - InPlaceIfPossible：控制器会优先尝试原地升级Pod，如果不行再采用重建升级。
  - InPlaceOnly：控制器只允许采用原地升级。
- maxUnavailable：发布过程中，限制最多不可用的Pod数量，可以设置为一个绝对值或者百分比。
- gracePeriodSeconds：每个Pod原地升级前Not Ready优雅等待时间。

ii. 使*cloneset.yaml*在ACK集群中生效。

```
kubectl create -f ./cloneset.yaml
```

预期输出：

```
cloneset.apps.kruise.io/demo-clone created
```

2. 查看Pod运行状态。

```
kubectl get pod
```

预期输出：

```
NAME        READY  STATUS   RESTARTS  AGE
demo-clone-5b9kl  1/1   Running  0     3s
demo-clone-6xjdg  1/1   Running  0     3s
demo-clone-bvmdj  1/1    Running  0     3s
demo-clone-dm22s  1/1    Running  0     3s
demo-clone-rbpg9  1/1   Running  0     3s
```

3. 查看CloneSet。

```
kubectl get clone
```

预期输出：

```
NAME    DESIRED  UPDATED  UPDATED_READY  READY  TOTAL  AGE
demo-clone  5    5    5       5    5    46s
```

- DESIRED：期望Pod数量（spec.replicas）。
- UPDATED：最新版本的Pod数量（status.updatedReplicas）。
- UPDATED_READY：最新版本的可用Pod数量（status.updatedReadyReplicas）。
- READY：可用Pod总量（status.readyReplicas）。
- TOTAL：实际Pod总量（status.replicas）。

使用Advanced StatefulSet部署有状态应用

1. 创建Advanced StatefulSet。

i. 创建*statefulset.yaml*。

```
apiVersion: apps.kruise.io/v1alpha1
kind: StatefulSet
metadata:
 name: demo-asts
spec:
 replicas: 3
 selector:
  matchLabels:
   app: guestbook-sts
 podManagementPolicy: Parallel
 template: #Pod template和官方StatefulSet的结构完全一致。
  metadata:
   labels:
    app: guestbook-sts
  spec:
   containers:
   - name: guestbook
    image: registry.cn-hangzhou.aliyuncs.com/kruise-test/guestbook:v1
    env:
    - name: test
     value: foo
    volumeMounts:
    - name: log-volume
     mountPath: /var/log
   readinessGates:
   - conditionType: InPlaceUpdateReady
   volumes:
   - name: log-volume
    emptyDir: {}
 updateStrategy:
  type: RollingUpdate
  rollingUpdate:
   podUpdatePolicy: InPlaceIfPossible #尽量使用原地升级，否则重建升级。
   maxUnavailable: 20% #发布过程最多20%不可用。
   inPlaceUpdateStrategy:
    gracePeriodSeconds: 3 #每个Pod原地升级前Not ready优雅等待时间。
```

- type：设置Pod升级策略，支持以下3种方式升级。
  - ReCreate：控制器会删除旧Pod和PVC，然后用新版本重新创建Pod和PVC。
  - InPlaceIfPossible：控制器会优先尝试原地升级Pod，如果不行再采用重建升级。
  - InPlaceOnly：控制器只允许采用原地升级。
- maxUnavailable：发布过程中，限制最多不可用的Pod数量，可以设置为一个绝对值或者百分比。
- gracePeriodSeconds：每个Pod原地升级前Not ready优雅等待时间。

ii. 使*statefulset.yaml*在ACK集群中生效。

```
kubectl create -f ./statefulset.yaml
```

预期输出：

```
statefulset.apps.kruise.io/demo-asts created
```

2. 查看Pod运行状态。

```
kubectl get pod
```

预期输出：

```
NAME     READY  STATUS   RESTARTS  AGE
demo-asts-0  1/1   Running  0     3h29m
demo-asts-1  1/1   Running  0     3h29m
demo-asts-2  1/1   Running  0     3h29m
```

3. 查看Advanced StatefulSet。

```
kubectl get asts
```

预期输出：

```
NAME    DESIRED  CURRENT  UPDATED  READY  AGE
demo-asts  3     3     3     3    3h30m
```

- DESIRED：期望Pod数量（spec.replicas）。
- UPDATED：最新版本的Pod数量（status.updatedReplicas）。
- READY：可用Pod总量（status.readyReplicas）。
- TOTAL：实际Pod总量（status.replicas）。

相关文档

- OpenKruise官网文档
- OpenKruise Github

16.3. 通过Ingress实现灰度发布和蓝绿发布

当对服务进行版本更新升级时，需要使用到滚动升级、分批暂停发布、蓝绿发布以及灰度发布等发布方式。本文将介绍在ACK集群中如何通过Ingress Controller来实现应用服务的灰度发布。

体验实验室

为了帮助您快速上手本教程，ACK为您提供了可以动手操作的交互式实验环境。如您需要体验，请访问通过Ingress进行灰度发布。

前提条件

您已创建一个ACK集群。具体操作，请参见创建Kubernetes托管版集群。

背景信息

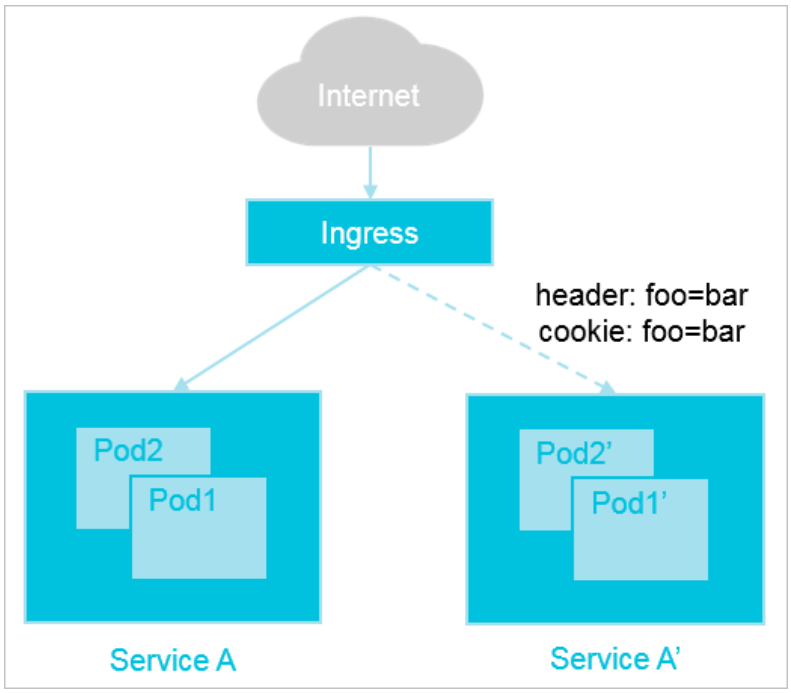
灰度及蓝绿发布是为新版本创建一个与老版本完全一致的生产环境，在不影响老版本的前提下，按照一定的规则把部分流量切换到新版本，当新版本试运行一段时间没有问题后，将用户的全量流量从老版本迁移至新版本。

其中AB测试就是一种灰度发布方式，一部分用户继续使用老版本的服务，将一部分用户的流量切换到新版本，如果新版本运行稳定，则逐步将所有用户迁移到新版本。

应用场景

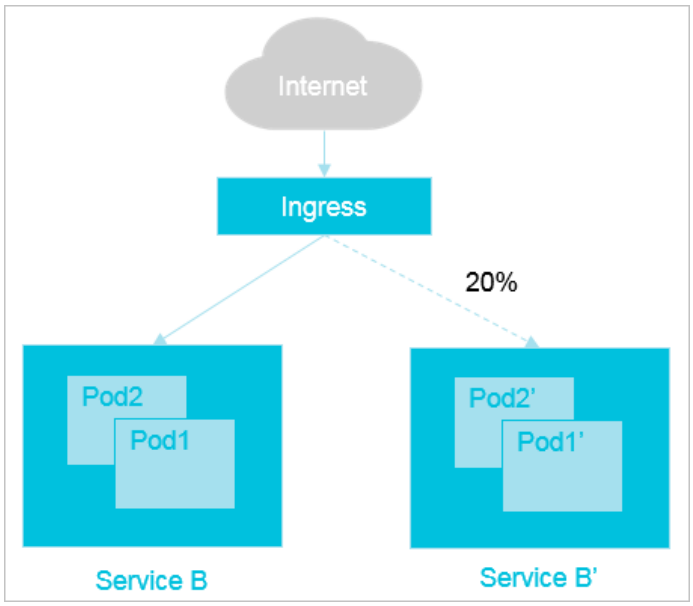
基于客户端请求的流量切分场景

假设当前线上环境，您已经有一套服务Service A对外提供7层服务，此时上线了一些新的特性，需要发布上线一个新的版本Service A'。但又不想直接替换Service A服务，而是希望将请求头中包含 `foo=bar` 或者cookie中包含 `foo=bar` 的客户端请求转发到Service A'服务中。待运行一段时间稳定后，可将所有的流量从Service A切换到Service A'服务中，再平滑地将Service A服务下线。



基于服务权重的流量切分场景

假设当前线上环境，您已经有一套服务Service B对外提供7层服务，此时修复了一些问题，需要发布上线一个新的版本Service B'。但又不想将所有客户端流量切换到新版本Service B'中，而是希望将20%的流量切换到新版本Service B'中。待运行一段时间稳定后，再将所有的流量从Service B切换到Service B'服务中，再平滑地将Service B服务下线。



针对以上多种不同的应用发布需求，阿里云容器服务Ingress Controller支持了多种流量切分方式：

- 基于Request Header的流量切分，适用于灰度发布以及AB测试场景。
- 基于Cookie的流量切分，适用于灰度发布以及AB测试场景。
- 基于Query Param的流量切分，适用于灰度发布以及AB测试场景。

- 基于服务权重的流量切分，适用于蓝绿发布场景。

注解说明

Ingress Controller通过下列Annotation来支持应用服务的灰度发布机制。

- nginx.ingress.kubernetes.io/service-match
  该注解用来配置新版本服务的路由匹配规则。

```
nginx.ingress.kubernetes.io/service-match: |
  <service-name>: <match-rule>
# 参数说明：
# service-name：服务名称，满足match-rule的请求会被路由到该服务中。
# match-rule：路由匹配规则
#
# 路由匹配规则：
# 1. 支持的匹配类型
# - header：基于请求头，支持正则匹配和完整匹配。
# - cookie：基于cookie，支持正则匹配和完整匹配。
# - query：基于请求参数，支持正则匹配和完整匹配。
#
# 2. 匹配方式
# - 正则匹配格式：/{regular expression}/，//表明采用正则方式匹配。
# - 完整匹配格式："{exact expression}"，""表明采用完整方式匹配。
```

  路由匹配规则配置示例：

```
# 请求头中满足foo正则匹配^bar$的请求被转发到新版本服务new-nginx中。
new-nginx: header("foo", /^bar$/)
# 请求头中满足foo完整匹配bar的请求被转发到新版本服务new-nginx中。
new-nginx: header("foo", "bar")
# cookie中满足foo正则匹配^sticky-.+$的请求被转发到新版本服务new-nginx中。
new-nginx: cookie("foo", /^sticky-.+$/)
# query param中满足foo完整匹配bar的请求被转发到新版本服务new-nginx中。
new-nginx: query("foo", "bar")
```

- nginx.ingress.kubernetes.io/service-weight
  该注解用来配置新旧版本服务的流量权重。

```
nginx.ingress.kubernetes.io/service-weight: |
  <new-svc-name>:<new-svc-weight>, <old-svc-name>:<old-svc-weight>
参数说明：
new-svc-name：新版本服务名称
new-svc-weight：新版本服务权重
old-svc-name：旧版本服务名称
old-svc-weight：旧版本服务权重
```

  服务权重配置示例：

```
nginx.ingress.kubernetes.io/service-weight: |
  new-nginx: 20, old-nginx: 60
```

步骤一：部署服务

部署Nginx服务并通过Ingress Controller对外提供7层域名访问。

1. 创建Deployment和Service。

i. 创建*nginx.yaml*。

```
apiVersion: apps/v1
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/acs-sample/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
```

ii. 执行以下命令，创建Deployment和Service。

```
kubectl apply -f nginx.yaml
```

2. 部署Ingress。

i. 创建*ingress.yaml*。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

ii. 执行以下命令，部署Ingress。

```
kubectl apply -f ingress.yaml
```

3. 测试访问情况。

i. 执行以下命令，获取外部IP。

```
kubectl get ingress
```

ii. 执行以下命令，查看路由访问情况。

```
curl -H "Host: www.example.com"  http://<EXTERNAL_IP>
```

预期输出：

```
old
```

步骤二：灰度发布新版本服务

发布一个新版本的Nginx服务并配置路由规则。

1. 部署新版本的Deployment和Service。

i. 创建*nginx1.yaml*

```
apiVersion: apps/v1
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/acs-sample/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
```

ii. 部署新版本的Deployment和Service。

```
kubectl apply -f nginx1.yaml
```

2. 设置访问新版本服务的路由规则。

   ACK支持设置以下三种路由规则，您可以根据实际情况选择路由规则。

   - 设置满足特定规则的客户端才能访问新版本服务。以下示例仅请求头中满足`foo=bar`的客户端请求才能路由到新版本服务。

a. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
 annotations:
  # 请求头中满足正则匹配foo=bar的请求才会被路由到新版本服务new-nginx中
  nginx.ingress.kubernetes.io/service-match: |
   new-nginx: header("foo", /^bar$/)
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

b. 查看路由访问情况。

- 执行以下命令，访问服务。

  ```
  curl -H "Host: www.example.com" http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  old
  ```

- 执行以下命令，请求头中满足 `foo=bar` 的客户端请求访问服务。

  ```
  curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  new
  ```

  重复执行以上命令，可以看到，仅请求头中满足 `foo=bar` 的客户端请求才能路由到新版本服务。

- 在满足特定规则的基础上设置一定比例的请求被路由到新版本服务中。以下示例要求请求头中满足 `foo=bar` 的客户端请求，且仅允许其中50%的流量被路由到新版本服务中。

a. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
 annotations:
  # 请求头中满足正则匹配foo=bar的请求才会被路由到新版本服务new-nginx中
  nginx.ingress.kubernetes.io/service-match: |
    new-nginx: header("foo", /^bar$/)
  # 在满足上述匹配规则的基础上仅允许50%的流量会被路由到新版本服务new-nginx中
  nginx.ingress.kubernetes.io/service-weight: |
    new-nginx: 50, old-nginx: 50
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

b. 查看路由访问情况。

- 执行以下命令，访问服务。

  ```
  curl -H "Host: www.example.com"  http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  old
  ```

- 执行以下命令，请求头中满足 `foo=bar` 的客户端请求访问服务。

  ```
  curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_IP>
  ```

  预期输出：

  ```
  new
  ```

  重复执行以上命令。可以看到，仅请求头中满足 `foo=bar` 的客户端请求，且只有50%的流量才能路由到新版本服务。

- 设置一定比例的请求被路由到新版本服务中，以下示例中仅50%的流量被路由到新版本服务中。

a. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
 annotations:
   # 允许50%的流量被路由到新版本服务new-nginx中
   nginx.ingress.kubernetes.io/service-weight: |
     new-nginx: 50, old-nginx: 50
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 老版本服务
   - path: /
    backend:
     service:
      name: old-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

b. 执行以下命令，查看路由访问情况。

```
curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

重复执行以上命令，可以看到仅50%的流量路由到新版本服务。

步骤三：删除老版本服务

系统运行一段时间后，当新版本服务已经稳定并且符合预期后，需要下线老版本的服务，仅保留新版本服务在线上运行。

1. 按照以下内容，修改步骤创建的Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: gray-release
spec:
 rules:
 - host: www.example.com
  http:
   paths:
   # 新版本服务
   - path: /
    backend:
     service:
      name: new-nginx
      port:
       number: 80
    pathType: ImplementationSpecific
```

2. 执行以下命令，查看路由访问情况。

```
curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

预期输出：

```
new
```

重复执行以上命令，可以看到请求全部被路由到了新版本的服务。

3. 删除旧版本的Deployment和Service。
   i. 执行以下命令，删除旧版本的Deployment。

   ```
   kubectl delete deploy <Deployment名称>
   ```

   ii. 执行以下命令，删除旧版本的Service。

   ```
   kubectl delete svc <Service名称>
   ```

16.4. Kubernetes集群中使用阿里云SLB实现四层金丝雀发布

在Kubernetes集群中，对于通过TCP/UDP访问的服务来说，七层的Ingress不能很好地实现灰度发布的需求。本文介绍如何使用SLB来进行四层的金丝雀发布。

前提条件

- 创建Kubernetes托管版集群
- 通过SSH连接ACK专有版集群的Master节点

步骤一：部署旧版本服务

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面单击右上角的**使用YAML创建资源**。

6. 选择**示例模板**，然后单击**创建**。

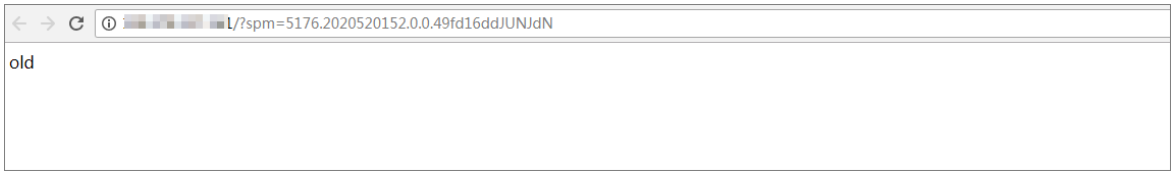


本例是一个Nginx的编排，通过SLB对外暴露的服务。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  run: old-nginx
 name: old-nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   run: old-nginx
 template:
  metadata:
   labels:
    run: old-nginx
    app: 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:
 labels:
  run: nginx
 name: nginx
spec:
 ports:
 - port: 80
   protocol: TCP
   targetPort: 80
 selector:
  app: nginx
 sessionAffinity: None
 type: LoadBalancer              #通过阿里云SLB对外暴露服务。
```

在**无状态**页面可以看到创建的应用。

7. 在集群管理页面左侧导航栏选择**网络 > 服务**，在**服务**页面，您可单击服务右侧的外部端点，进入Nginx默认欢迎页面，本示例中的Nginx欢迎页面会显示**old**，表示当前访问的服务对应后端old-nginx容器。



为了方便展示多次发布的情况，建议登录Master节点，后面都将通过 `curl` 命令查看部署的效果。

```
bash
for x in {1..10} ; do curl EXTERNAL-IP; done          #EXTERNAL-IP为服务的外部端点。
old
old
old
old
old
old
old
old
old
old
```

步骤二：上线新版本Deployment

1. 在控制台左侧导航栏中，单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 在**无状态**页面单击右上角的**使用YAML创建资源**。
5. 选择**示例模板**，然后单击**创建**。

   本示例是一个Nginx的新版本的Deployment，其中的Labels，也是含有 `app:nginx` 标签。这个标签就是为了使用与旧版本Deployment相同的Nginx服务，这样就可以将对应的流量导入进来。
   本示例的编排模板如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  run: new-nginx
 name: new-nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   run: new-nginx
 template:
  metadata:
   labels:
    run: new-nginx
    app: nginx
  spec:
   containers:
   - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
    imagePullPolicy: Always
    name: new-nginx
    ports:
    - containerPort: 80
     protocol: TCP
   restartPolicy: Always
```

6. 在**无状态**页面，可以看到名为**new-nginx**的Deployment。
7. 登录Master节点，执行**curl**命名，查看服务访问的情况。

```
bash
for x in {1..10}; do curl EXTERNAL-IP; done          #EXTERNAL-IP为服务的外部端点。
new
new
new
old
new
old
new
new
old
old
```

可看到五次访问到旧的服务，五次访问到新的服务。主要原因是Service对于流量请求是平均的负载均衡策略，而且新老Deployment均为一个Pod，因此它们的流量百分比为1：1。

步骤三：调整流量权重

基于SLB的金丝雀发布需要通过调整后端的Pod数量来调整对应的权重。例如我们这里希望新的服务权重更大一些，假如将新的Pod数量调整到4个。

> **说明** 新旧版本应用同时存在时，某个样本的Curl命令返回的结果并非严格按照设置的权重，本例中执行10次Curl命令，您可通过观察更大的样本获取接近的效果。

1. 在控制台左侧导航栏中，单击**集群**。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
4. 选择命名空间，选择所需的Deployment，单击目标应用右侧**操作**列的**伸缩**。
5. 在弹出的对话框中，设置所需容器组的数量，将其调整为4。

   > **说明** Kubernetes的Deployment资源默认的更新方式就是rollingUpdate，所以在更新过程中，会保证最小可服务的容器个数，该个数也可以在模板里面调整。

6. 待部署完成后，登录Master节点，执行Curl命令，查看效果。

```
bash
for x in {1..10}; do curl EXTERNAL-IP; done          #EXTERNAL-IP为服务的外部端点。
new
new
new
new
new
old
new
new
new
old
```

可以看到，10个请求里面，有8个请求到新的服务，2个到老的服务。
您可通过动态地调整Pod的数量来调整新老服务的权重，实现金丝雀发布。

16.5. 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，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，通过控制台对通过Helm发布的应用进行管理。

前提条件

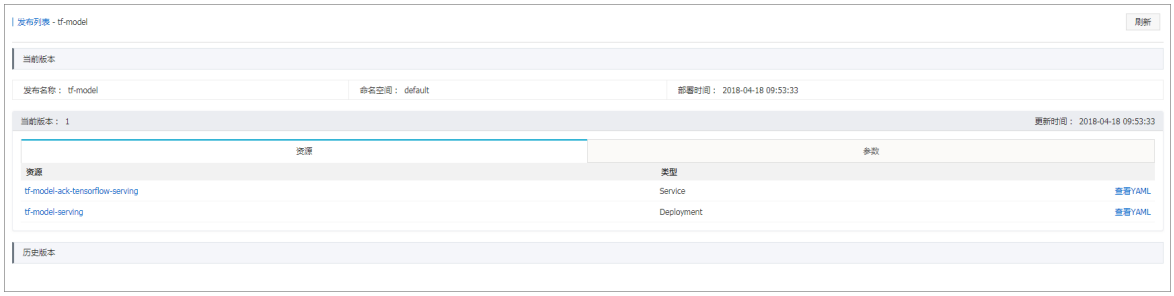
- 已成功创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已使用应用目录安装Helm应用。具体操作，请参见利用Helm简化应用部署。本例中是一个tf-model应用。

查看发布的详情

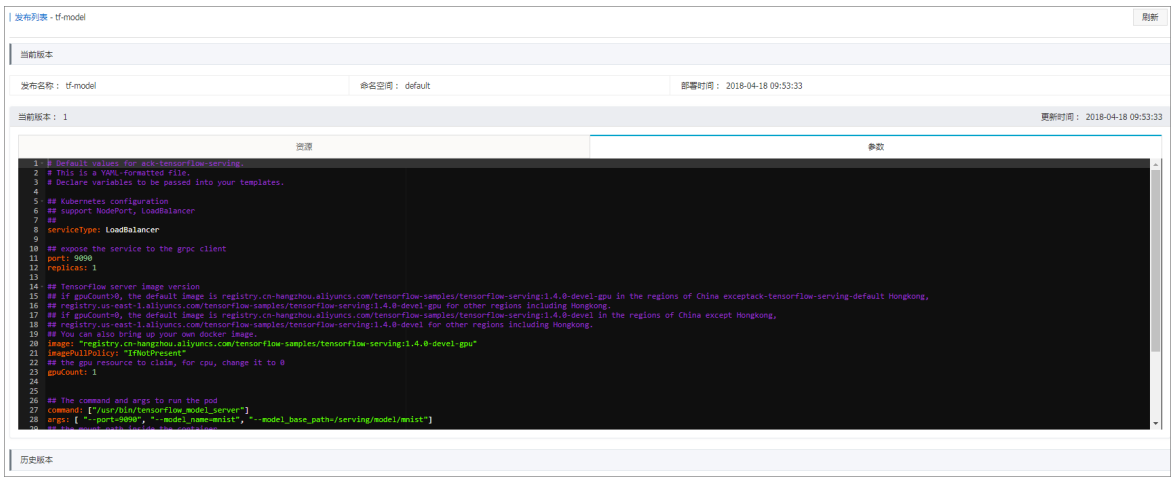
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Helm**。
5. 以tf-model为例，您可查看发布的详情信息，单击右侧的**详情**，进入该发布的详情页面。

   您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YAML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。



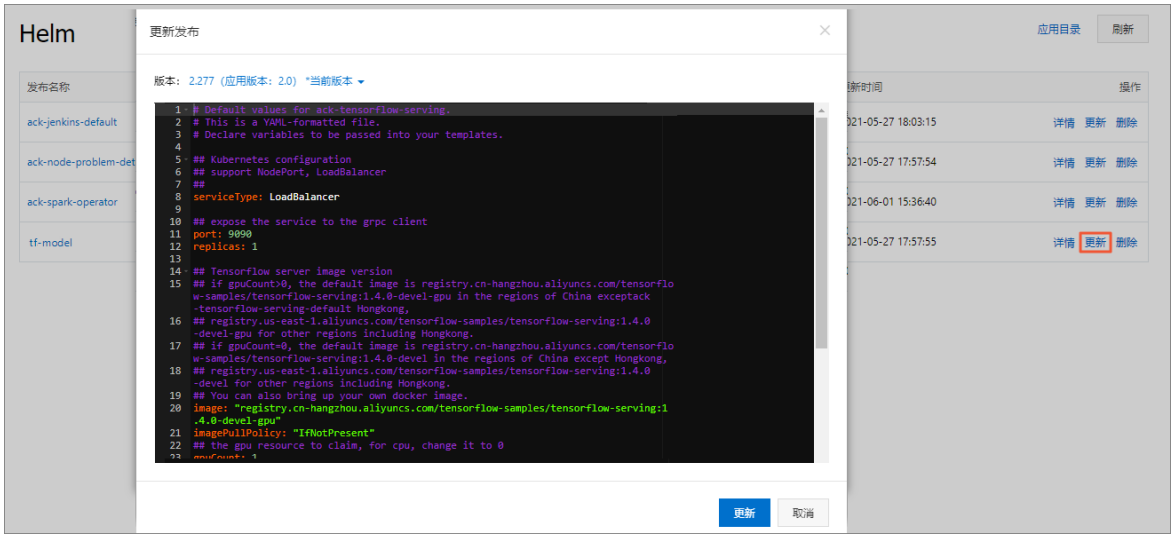
6. 单击**参数**页签，您可查看该Helm包安装的参数配置。



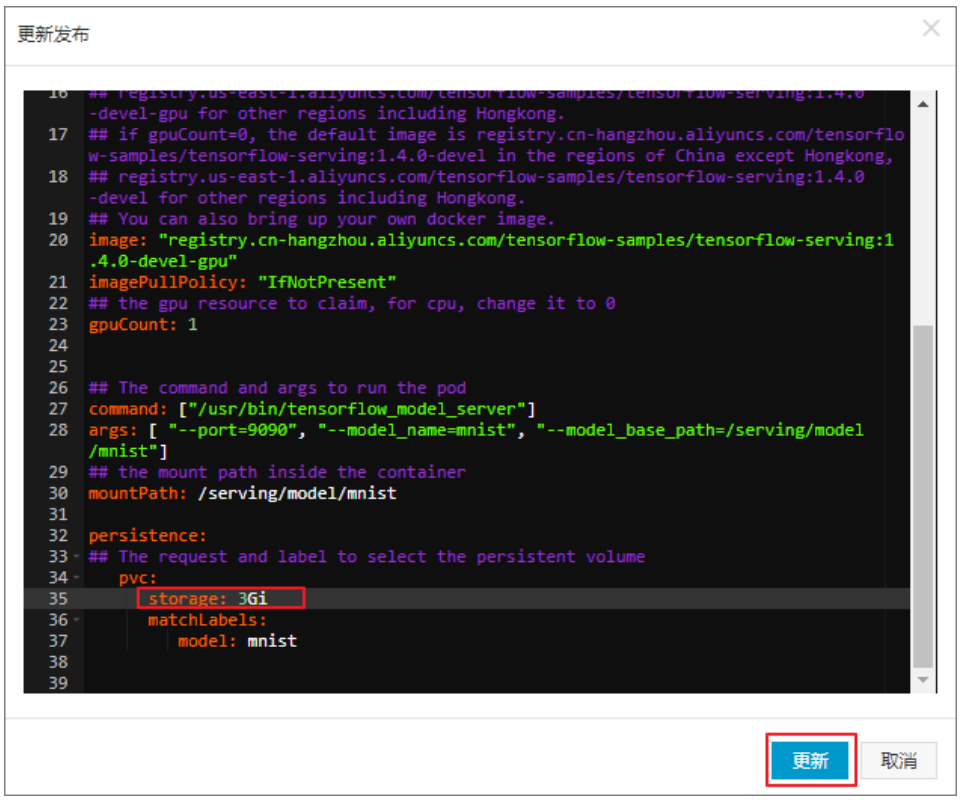
更新发布的版本

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Helm**。

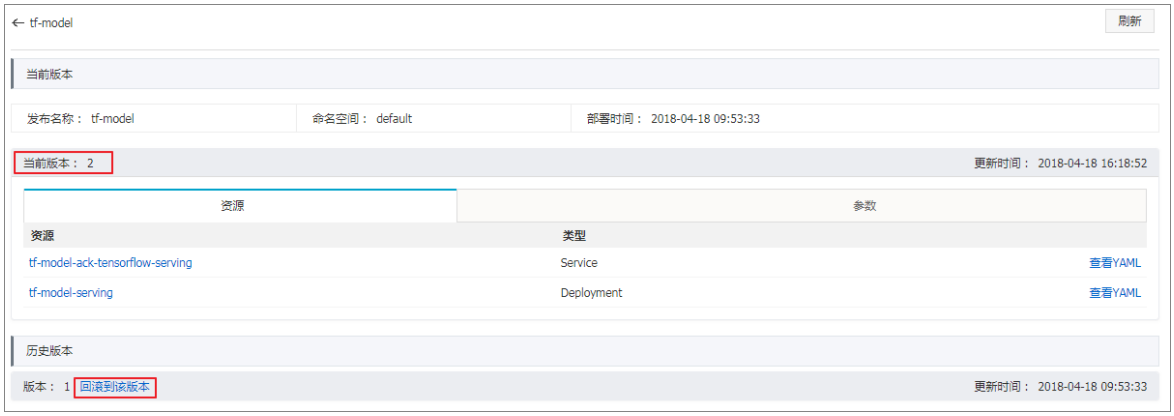
5. 以tf-model为例，您可更新该发布，单击右侧的**更新**，弹出更新发布对话框。



6. 在**更新发布**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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Helm**。
5. 以tf-model为例，您可删除该发布，单击右侧的**删除**。
6. 在**删除应用**对话框中，选中是否**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。



> **说明** 若不选中**清除发布记录**，此时该发布记录会保留在发布列表中，后续在部署应用时，可能会有重名冲突。

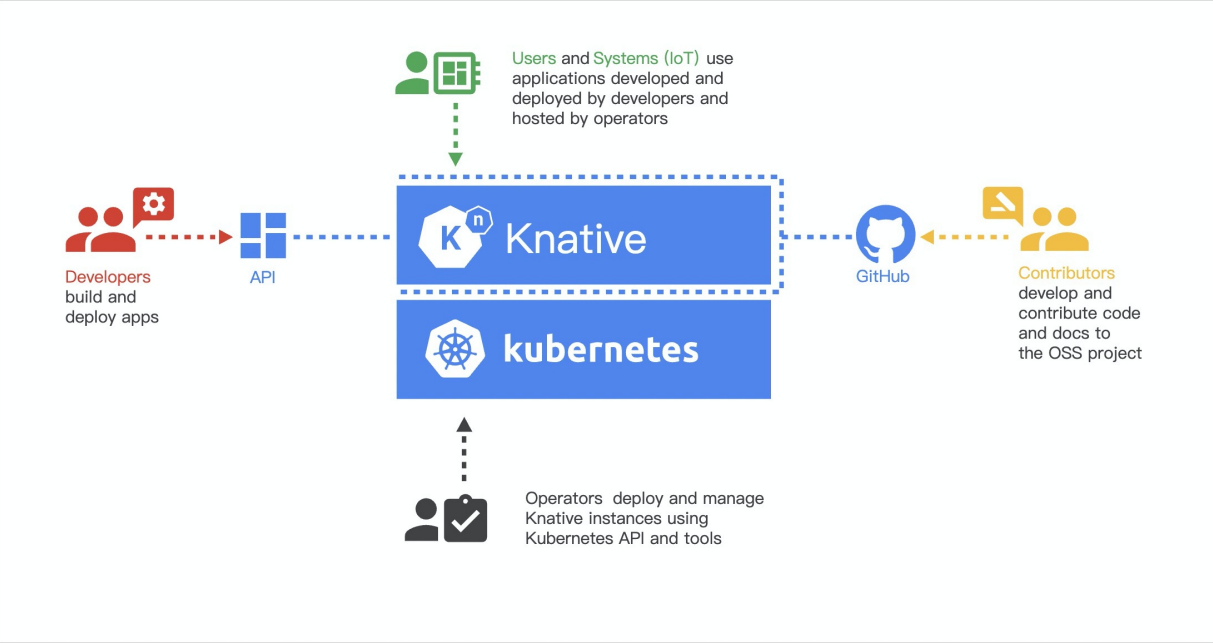
17.Knative

17.1. Knative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：

- Tekton：提供从源码到镜像的通用构建能力。
  Tekton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：提供了事件的接入、触发等一整套事件管理的能力。
  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：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。

Serving组件的职责是管理工作负载以对外提供服务。Knative Serving组件最重要的特性就是自动伸缩的能力，目前伸缩边界无限制。Serving还具有灰度发布能力。

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。

Knative安装部署

标准托管Kubernetes 、标准专有Kubernetes以及Serverless Kubernetes （ASK）集群支持部署Knative。部署方式如下：

- 在标准托管Kubernetes集群和标准专有Kubernetes集群中安装部署Knative，详细介绍请参见一键部署Knative。
- 在Serverless Kubernetes （ASK）集群中安装部署Knative，详细介绍请参见开启Knative。

> **说明** 如果您的ASK集群版本<1.16，请先升级ASK集群。

Knative计费说明

Knative本身不收取管理费用，但在使用过程中所创建的云服务器、负载均衡实例、NAT网关等，按照相应资源的价格计费。

- 云服务器ECS计费概述
- 公网NAT网关计费说明
- 负载均衡按量计费

Knative交流群

如果您对于Knative有任何疑问，欢迎使用钉钉扫描二维码或者点击链接加入钉钉交流群。

更多信息

关于Knative更多内容请参见Knative系列文章。

相关文档

- 一键部署Knative
- 快速部署Serverless应用

17.2. Knative版本发布说明

17.2.1. Knative发布0.18.3版本说明

阿里云Knative当前已经支持0.18.3 版本，本文介绍Knative 0.18.3版本所做的变更内容和支持的特性。

> **说明** Knative Service的apiVersion推荐使用V1版本，V1alpha1和V1beta1版本将在Knative 0.19.0版本之后废弃。

版本解读

- Knative 0.18.3版本功能的实现需要Kubernetes版本大于等于1.18。
- 多容器支持。Knative Service支持在一个Pod里配置多个业务容器。
- 支持基于Header的路由转发策略。在请求的Header中设置 `Knative-Serving-Tag: {revision-tag}` ，结合Kourier网关，可以将请求自动转发到指定的版本上。通过该功能可以实现基于Header的灰度发布。
- 兼容K8s的节点选择特性，该特性通过affinity，nodeSelector和tolerations参数配置。
- Knative Service支持Dryrun功能，可以快速验证当前的Revision Template是否配置正确。在Template中可通过以下任意参数开启此特性：
  - features.knative.dev/podspec-dryrun: enabled
  - features.knative.dev/podspec-dryrun: strict

> **说明** 当创建Knative Service时：
> - 将features.knative.dev/podspec-dryrun设置成 `enabled` ，表示如果K8s支持Dryrun功能，则会执行Dryrun。如果K8s不支持Dryrun功能也会继续执行创建操作。
> - 将features.knative.dev/podspec-dryrun设置成 `strict` ，表示当K8s不支持Dryrun功能时会返回 `failure` 。

17.3. Knative组件管理

17.3.1. 一键部署Knative

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。本文介绍如何部署Knative。

前提条件

创建Kubernetes托管版集群。

> **注意**
> - 从Knative v0.14.0开始，仅支持Kubernetes版本1.15及以上的集群。支持标准托管、标准专有以及Serverless Kubernetes集群。
> - 针对标准托管和标准专有Kubernetes集群，Worker节点数量需要大于等于3个。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 在**组件管理**页签，单击**一键部署Knative**。
6. 选择需要安装的Knative组件后，单击**部署**。
   - **Serving组件**：管理Serverless工作负载，可以和事件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。
   - **Tekton组件**：提供从源码到镜像的通用构建能力。

执行结果

部署完成后，您可看到部署结果信息。

- 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Knative应用相关操作。



17.3.2. 部署Knative组件

Knative部署完成之后，如果需要对未安装的组件进行部署时，可以在**组件管理**页面，可以选择未部署的组件进行部署操作。ACK支持部署核心组件以及add-on组件。本文介绍如何部署Knative组件。

前提条件

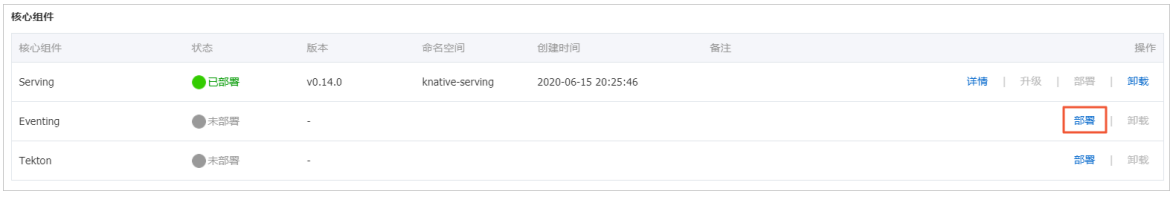
- 创建Kubernetes托管版集群
- 部署Knative

背景信息

您部署Knative时，如果您未选中需要部署的组件，您可以按照本文操作。

操作步骤

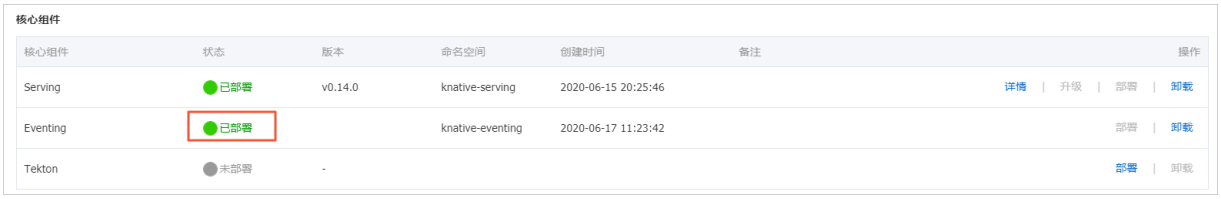
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 在**状态**为**未部署**的组件右侧，单击**部署**。



6. 在弹出的对话框中，单击**确定**。

执行结果

部署完成后，在**Knative组件管理**页面，可以看到当前组件状态为*已部署*。



相关文档

- 一键部署Knative
- 升级组件

17.3.3. 升级组件

ACK支持Knative Serving组件升级。Serving组件用于管理Serverless工作负载，可以和事件结合并且提供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可以缩容到零个实例。本文介绍如何升级Serving组件。

前提条件

- 您已部署Serving组件，请参见部署Knative组件。
- Kubernetes版本大于1.15.0。
- 仅支持Knative 0.11.0版本升级。

> **说明** 建议您在业务运行低谷时进行升级操作。

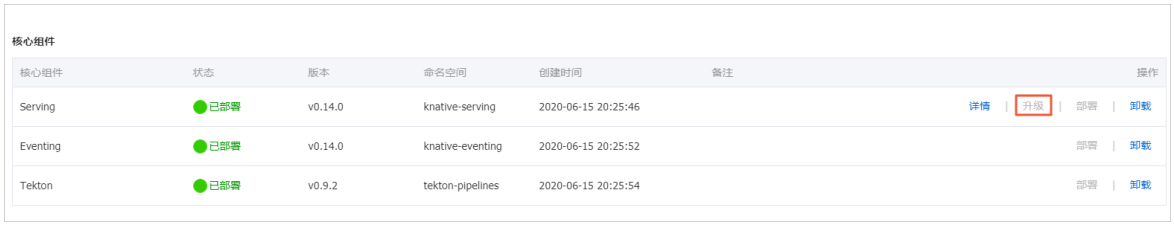
背景信息

您可以升级Serving组件至0.14.0。
Knative Serving 0.14.0版本新增以下主要特性：

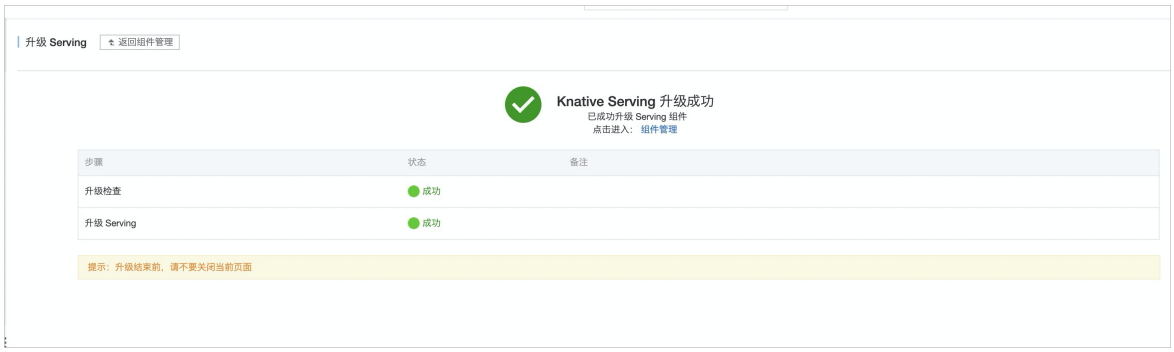
- 默认最小保留Knative服务的修订版本数为20，默认保留时间为48小时。
- 支持高可靠组件，包括controller、hpaautoscaler等。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 在**组件管理**页签的**核心组件**区域，选择**Serving**组件，然后单击**升级**。



升级完成之后，显示执行结果信息。



17.3.4. 采集组件日志

您可以采集Knative组件的日志，便于后续通过日志进行组件运行分析及问题排查。本文介绍如何通过日志服务对Knative组件进行Docker标准输出日志采集。

前提条件

创建Kubernetes托管版集群

背景信息

Knative组件包括：

- knative-serving
  - activator
  - autoscaler
  - autoscaler-hpa
  - controller
  - webhook
- knative-eventing
  - eventing-controller
  - eventing-webhook

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏单击**集群信息**，单击**集群资源**页签。
5. 在**集群资源**页签单击**日志服务Project**右侧的链接。
6. 在日志服务项目**概览**页面右上角，单击**接入数据**。
7. 在**接入数据**对话框，单击**Docker标准输出-容器**。
8. 完成Docker标准输出数据接入配置。

   以下重点描述如何配置**数据源设置**。其他步骤，请参见创建采集配置。

   本文以采集knative-serving中的controller组件为例，说明如何配置**数据源设置**。示例代码如下。

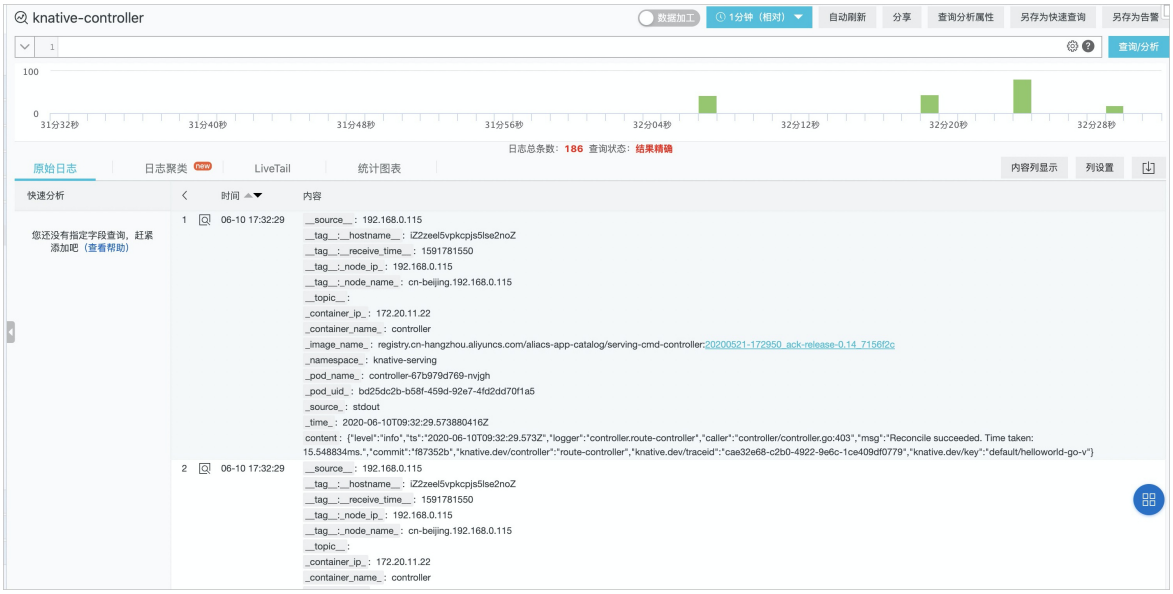
```
{
  "inputs": [
    {
      "detail": {
        "IncludeEnv": {
     "SYSTEM_NAMESPACE":"knative-serving"
    },
        "IncludeLabel": {
     "io.kubernetes.container.name": "controller"
        },
        "ExcludeLabel": {}
      },
      "type": "service_docker_stdout"
    }
  ]
}
```



说明

- IncludeEnv中SYSTEM_NAMESPACE设置对应的命名空间。
- IncludeLabel中io.kubernetes.container.name设置相应组件的名称。

9. 在**查询日志**区域，单击**立即尝试**查看采集结果。



17.3.5. 设置组件监控告警

您可以通过应用实时监控服务中的Prometheus监控设置Knative组件的监控指标。本文介绍如何设置Knative组件监控告警信息。

背景信息

Knative组件采集指标包括：

- 组件可用实例数
- 组件CPU
- 组件Memory

Knative组件包括：

- knative-serving
  - activator
  - autoscaler
  - autoscaler-hpa
  - controller
  - webhook
- knative-eventing
  - eventing-controller
  - eventing-webhook

步骤一：安装Prometheus监控组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的**阿里云应用**页签中，搜索并单击**ack-arms-prometheus**。
4. 在**应用目录 - ack-arms-prometheus**右侧的**创建**区域中，选择目标集群，然后单击**创建**。

> **说明** 命名空间和发布名称默认为arms-prom。

（可选）步骤二：查看Pod监控信息

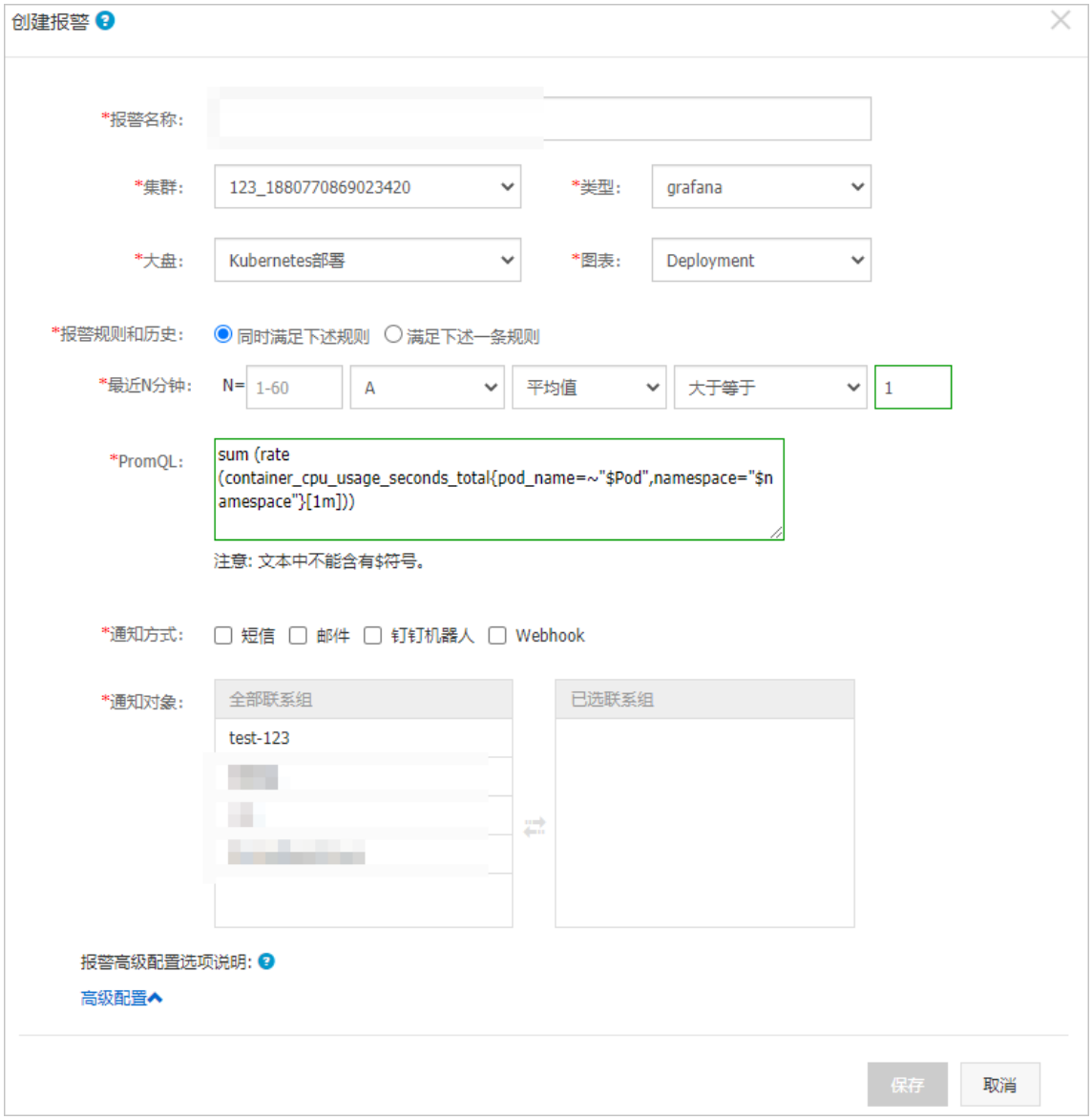
在进行报警策略设置前，您可以预先查看Pod相关监控信息。

1. 登录ARMS 控制台。
2. 在左侧导航栏中单击**Prometheus监控**。
3. 在**Prometheus监控**页面上，单击**已安装大盘**列中的**k8s state**链接，即可在浏览器新窗口中打开对应的监控仪表板，并查看Pod的CPU、Memory，以及Pod个数。

步骤三：设置组件报警策略

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节。
   - 在Prometheus Grafana大盘的**New DashBoard**页面，单击左侧的图标，跳转至Prometheus **创建报警**对话框。
   - 在控制台左侧导航栏中选择**报警管理 > 报警策略管理**，进入**报警策略管理**页面，在右上角单击**创建报警 > Prometheus**。
2. 在弹出的**创建报警**对话框中，设置相应参数。

   以Knative Serving组件activator为例，如果可用Pod数小于期望实例数（即不可用Pod数大于等于1），则进行告警通知。设置告警策略如下图。



其中，activator组件PromQL如下。

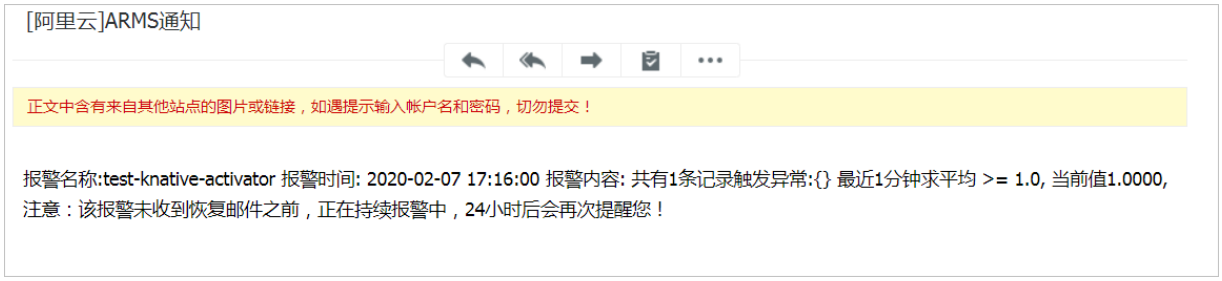
```
((sum(kube_deployment_status_replicas{deployment=~"activator"}) or vector(0))) - ((sum(kube_deployment_status_replicas_available{deployment=~"activator"}) or vector(0)))
```

详细的参数说明，请参见创建报警。

3. 单击**保存**。

执行结果

您可以设置通过短信、邮件及钉钉机器人接收通知。触发告警之后，邮件接收通知如下。



17.3.6. 卸载Knative组件

在Knative**组件管理**页面，您可以卸载已部署的组件。ACK支持卸载核心组件以及add-on组件。本文以**Eventing**为例进行介绍如何卸载组件。

前提条件

- 创建Kubernetes托管版集群
- 部署Knative

操作步骤

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
4. 在**组件管理**页签，选择目标组件，然后在其右侧单击**卸载**。
5. 在弹出的对话框中，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*。



17.3.7. 卸载Knative

本文主要为您介绍如何卸载Knative。

前提条件

一键部署Knative

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 在**组件管理**页签，单击右上角的**一键卸载**。
6. 在弹出的**卸载Knative**页面，选中**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。



17.4. Knative服务管理

17.4.1. 快速部署Serverless应用

在Knative中服务代表应用的概念，本文主要为您介绍如何创建Knative服务。

前提条件

- 创建Kubernetes托管版集群
- 一键部署Knative
- 部署Serving组件，请参见部署Knative组件。

步骤一：部署Knative服务

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 单击**服务管理**页签，然后单击右上角的**创建服务**。
6. 设置配置信息。

参数	说明
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**。本例中为*73fbdd56*。 镜像名称 registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go 镜像版本 73fbdd56
访问协议	支持HTTP和gRPC两种访问协议。
容器端口	设置暴露的容器访问端口，端口号必须介于1~65535。

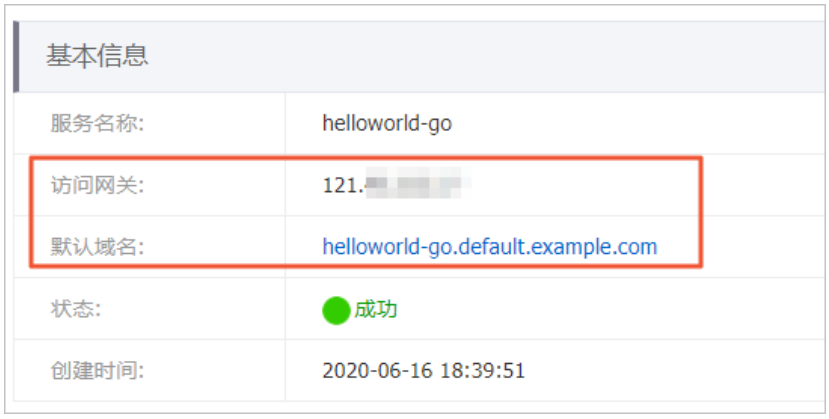
参数	说明
仅内部访问	选中**仅内部访问**后，不支持通过公网访问该服务。
请求最大并发数	容器允许的最大请求并发数。默认0，表示不限制并发数。
最小缩容实例数	在无访问请求的情况下，最小缩容的运行实例数。设置为0时，表示没有访问请求时，实例缩为0。
最大扩容实例数	允许扩容出来的最多实例个数。
资源限制	可指定该应用所能使用的资源上限，包括CPU、内存和GPU三种资源，防止占用过多资源。其中，CPU资源的单位为cores，即一个核；内存的单位为Bytes，可以为MiB。
生命周期	包含命令（Command）和参数（Args），配置说明如下： ○ 如果均不配置，则使用镜像默认的命令和参数。 ○ 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。
数据卷	支持增加本地存储和云存储声明（PVC）。 ○ 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。 ○ 云存储声明（PVC）：支持云存储。

7. 单击**创建**。

步骤二：访问服务

Knative服务创建完成后，通过绑定Host域名与访问网关，可以直接访问服务地址。操作步骤如下。

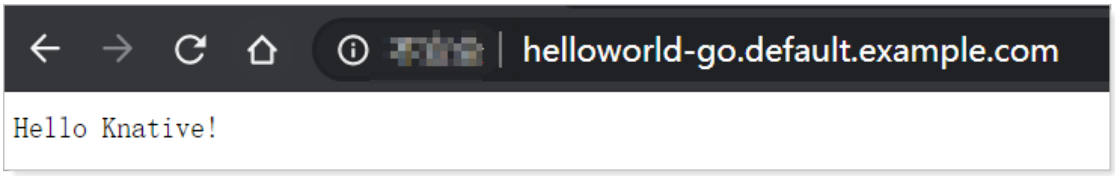
1. 在**服务管理**页签，单击服务名称。
   您可以在Knative服务详情页的**基本信息**区域，查看访问网关及域名。



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。
   绑定样例如下。

```
121.xx.xxx.xx helloworld-go.default.example.com
```

3. 完成Host绑定后，可通过域名直接对服务进行访问。



17.4.2. 创建修订版本

在Knative中，通过创建新的修订版本，实现服务的升级操作。本文主要为您介绍如何创建修订版本。通过创建修订版本，当您发布新版本应用后想要回滚到老版本时，您只需转发流量至指定版本，即可实现版本回滚。

前提条件

- 创建Kubernetes托管版集群
- 部署Knative
- 部署Knative组件
- 创建Knative服务

背景信息

安装Knative Service时，系统会创建一个stock-service-example-v1的Revision，并且是把100%的流量都打到stock-service-example-v1上。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 单击**服务管理**页签，选择目标命名空间，单击目标服务名称操作列的**详情**。
6. 在目标服务详情页面，单击右上角的**创建修订版本**。
7. 在**创建修订版本**页面，配置参数。
   i. 设置**基本信息**。
      有关创建修订版本的基本信息配置，请参见设置配置信息。
   ii. 配置**流量设置**信息。
      - 修订版本：对Knative服务进行一次发布，会创建一个Revision修订版本。
      - 流量比例：访问请求流量的比例。各个修订版本流量比例之和需要等于100。
8. 单击**创建**。
   您可以在Knative服务详情页面中的**修订版本信息**区域，看到新创建的修订版本信息。

相关信息

创建修订版本后，您还可以删除修订版本。

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
4. 单击**服务管理**页签，在页面上方设置命名空间，然后单击目标服务的名称。
5. 在**修订版本信息**区域，选择待删除的修订版本，单击**操作**列下的**删除**。
6. 在**删除修订版本**对话框中，单击**确认**。

17.4.3. 在Knative使用自定义域名

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，参见一键部署Knative。

背景信息

在Knative Serving Route路由中使用 `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}` 。

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. 通过kubectl工具连接集群。
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。

   将配置文件中的example.com改为自定义域名后，保存配置。本例中的自定义域名为：mydomain.com。

   ```
   apiVersion: v1
   data:
    mydomain.com: ""
   kind: ConfigMap
   [...]
   ```

部署应用

如果您已经部署了应用，Knative会根据域名配置configmap，自动更新所有的Service和Route。

1. 部署一个应用。以部署一个命名为*helloworld-go*的应用为例，请参见快速部署Serverless应用。
2. 执行以下操作，查看部署结果。
   - 在Knative 0.7版本下，执行以下命令，查看域名。

     ```
     kubectl get route ${SVC_NAME} --output jsonpath="{.status.url}"| awk -F/ '{print $3}'`
     ```

   - 在Knative 0.6版本下，执行以下命令，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
     ```

发布域名

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 单击**服务管理**页签，然后单击目标服务名称。
   您可以在Knative服务详情页的**基本信息**区域，查看访问网关和域名。

6. 将Knative网关IP设置到对应的域名解析。更多信息，请参见添加解析记录。

执行结果

执行以下命令，查看执行结果。

```
curl http://helloworld-go.default.mydomain.com
```

17.4.4. 删除修订版本

本文主要为您介绍如何删除修订版本。

前提条件

- 创建Kubernetes托管版集群
- 一键部署Knative
- 部署Serving组件
- 快速部署Serverless应用

操作步骤

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
4. 单击**服务管理**页签，在页面上方设置命名空间，然后单击目标服务的名称。
5. 在**修订版本信息**区域，选择待删除的修订版本，单击**操作**列下的**删除**。
6. 在弹出的**删除修订版本**对话框中，单击**确认**，删除修订版本。

17.4.5. 删除Knative服务

本文主要为您介绍如何删除Knative服务。

前提条件

- 创建Kubernetes托管版集群
- 部署Knative
- 部署Serving组件

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 单击**服务管理**页签，选择待删除的服务，单击**操作**列下的**删除**。
6. 在的**删除服务**对话框中，单击**确认**。

17.4.6. 在Knative中基于流量灰度发布服务

Knative提供了基于流量的灰度发布能力，您可以根据流量百分比灰度发布服务。本文介绍如何在Knative中基于流量灰度发布服务。

前提条件

部署Knative：

- 如果是ACK集群，关于部署Knative具体操作，请参见部署Knative。
- 如果是ASK集群，关于部署Knative具体操作，请参见开启Knative。

步骤一：创建服务

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 选择**服务管理**页签，然后单击页面右上角的**创建服务**。
6. 设置**命名空间**和**服务名称**，选择所要使用的镜像和镜像版本等配置信息。

参数	说明
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，本例服务名称为*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**。本例中镜像版本为*73fbdd56*。
访问协议	支持**HTTP**和**gRPC**两种访问协议。 **说明** gRPC是基于HTTP2协议标准设计和ProtoBuf（Protocol Buffers）序列化协议开发的，且支持众多开发语言。与HTTP相比，HTTP2在发送和接收方面更紧凑和高效。
容器端口	设置暴露的容器访问端口，端口号必须介于1~65535。

关于创建服务的其他参数详细信息，请参见参数说明。

7. 单击**创建**。

创建完成后，您可以在**服务管理**页签的服务列表中，看到新创建的服务。

8. 执行以下命令，访问服务。

```
curl -H "<默认域名>" http://<访问网关>
```

- 默认域名：本例为“host: helloworld-go.default.example.com”。
- 访问网关：本例为“39.106.XX.XX”。

预期输出：

```
Hello World!
```

步骤二：通过创建修订版本灰度发布服务

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

4. 在集群管理页左侧导航栏中，选择**应用 > Knative**。
5. 创建修订版本。
   i. 选择**服务管理**页签，然后单击目标服务右侧**操作**列下的**详情**。
   ii. 单击**创建修订版本**。
   iii. 在**基本信息**配置向导步骤，单击**高级设置**，设置最新修订版本的环境变量为 `TARGET=Knative` 。



   iv. 单击**下一步**。
   v. 在**流量设置**配置向导步骤，设置最新修订版本的**流量比例**为0，单击**创建**。

> **说明** 所有修订版本的流量比例之和需要等于100。

   vi. 创建完成后，选择**服务管理**页签，可以看到新创建服务版本的详细信息。
   vii. 执行以下命令，访问服务。

> **说明** 由于新版本服务的**流量比例**为0，所以访问helloworld-go服务，还是请求到旧版本的服务。

```
curl -H "host: helloworld-go.default.example.com" http://39.106.XX.XX
```

   预期输出：

```
Hello World!
```

6. 修改流量比例灰度发布服务。
   i. 在**服务管理**页面，单击目标服务右侧**操作**列下的**详情**。
   ii. 单击**设置流量比例**。
   iii. 在**设置流量比例**对话框中，将新版本和旧版本的**流量比例**都设置为50%，然后单击**确定**。
   iv. 服务**流量比例**设置成功后，选择**服务管理**页签，可以看到新版本和旧版本服务的详细信息。

v. 执行以下命令，访问服务

```
curl -H "host: helloworld-go.default.example.com" http://39.106.XX.XX
```

预期输出：

```
Hello Knative!
Hello Knative!
Hello World!
Hello Knative!
Hello World!
Hello Knative!
Hello World!
Hello World!
Hello World!
Hello Knative!
Hello Knative!
Hello World!
Hello Knative!
Hello Knative!
Hello Knative!
```

由于新、旧版本的**流量比例**各为50%，所以当前访问helloworld-go服务的话，基本是按50%进行流量分配。
您可以通过调整**流量比例**继续灰度发布服务，直到新版本服务的**流量比例**为100%，完成灰度发布服务。在这个过程中，如果发现新版本有问题，您可以随时通过调整**流量比例**的方式进行回滚操作。

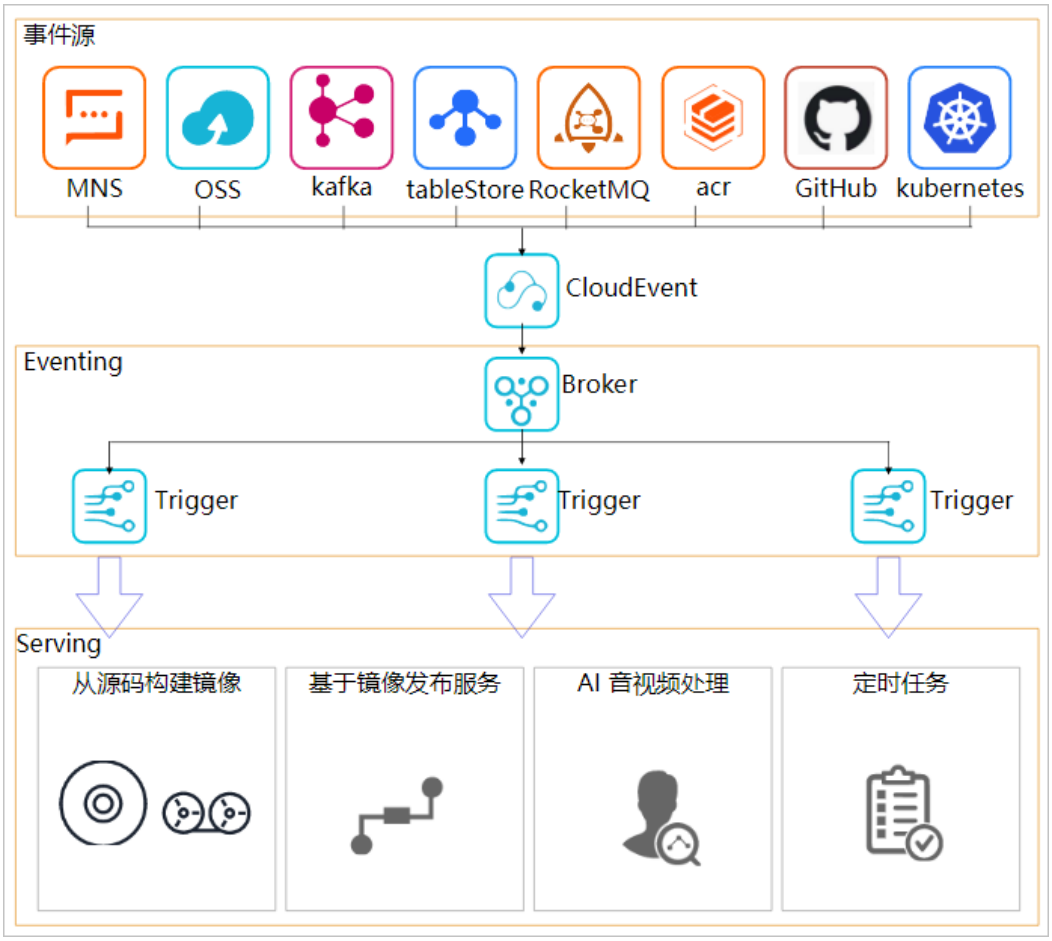
17.5. Knative事件驱动

17.5.1. 事件驱动概述

Knative Eventing设计目标是满足云原生开发的常见需求，并提供可组合的方式绑定事件源和事件消费者进行事件处理。本文主要对事件接入、事件处理和事件消费场景作简单介绍。

功能介绍

Knative Eventing在满足云原生开发的常见需求的基础上对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、事件流转和订阅、以及对事件的过滤等功能。事件驱动的整体框架图如下所示：



- 事件接入
  - Knative社区提供了丰富的事件源，如Kafka、GitHub等。
  - 接入消息云产品事件源，如MNS、RocketMQ等。
- 事件处理
  - Knative Eventing内部通过Broker/Trigger模型实现事件的订阅、过滤和路由机制。
  - 事件可以通过Knative管理的Serverless服务进行直接消费处理。
- 事件消费场景
  - ACR镜像更新自动发布服务。
  - 代码提交自动构建镜像。
  - AI音视频处理、定时任务等。

事件处理的使用

关于使用事件处理的具体实例，请参见部署Knative组件。

相关文档

- 在Knative上实现GitHub事件处理
- 在Knative上使用MnsOss事件源

17.5.2. 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，参见部署Knative组件。
- 您已经在Knative上自定义域名并完成域名配置。参见在Knative使用自定义域名。

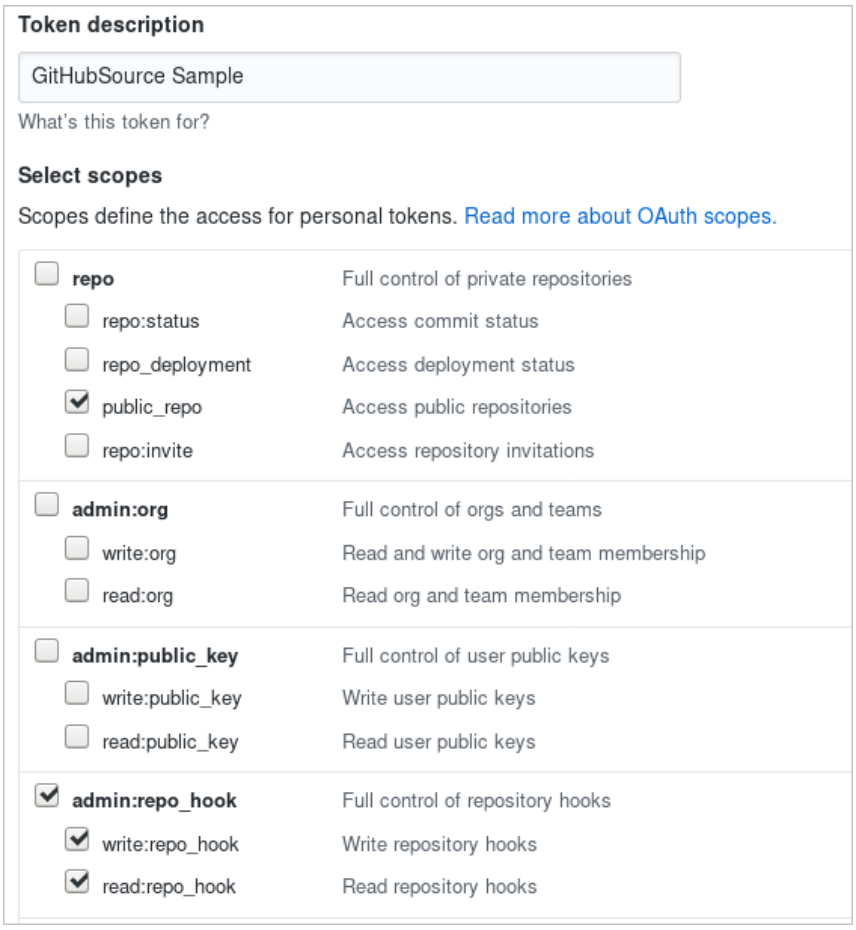
步骤一：创建服务

关于创建服务的详细步骤，请参见快速部署Serverless应用。

步骤二：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工具连接集群。
3. secretToken内容可以通过下述方式生成随机字符串。

   ```
   head -c 8 /dev/urandom | base64
   ```

4. 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
```

5. 执行以下命令，创建GitHub Token。

```
kubectl --namespace default apply githubsecret.yaml
```

步骤三：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工具连接集群。
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
```

执行结果

在GitHub repository中选择**Settings > Webhooks**，可以看到有一个验证成功的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-e383-11e8-8069-46e3c8ad****
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: 12*.*.*.*, 12*.*.*.*
X-Forwarded-Proto: http
X-Request-Id: 8a2201af-5075-9447-b593-ec3a243a****
{"action":"opened","number":1,"pull_request": ...}
```

17.5.3. 在Knative上使用MnsOss事件源

通过MnsOss事件源采集OSS事件，然后驱动服务对事件进行处理，可用于人脸识别等场景。本文介绍如何在Knative上使用MnsOss事件源。

前提条件

- 已安装Knative Serving和Knative Eventing。具体操作，请参见一键部署Knative。
- 已在OSS控制台创建Bucket。具体操作，请参见创建存储空间。
- 已开通MNS服务。具体操作，请参见开通消息服务MNS并授权。

步骤一：安装MnsOss事件源

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页面选择**应用 > Knative**。
5. 在**组件管理**页签下单击MnsOss**操作**列的**部署**。
6. 在**部署MnsOSS**对话框中单击**确定**。

步骤二：创建OSS事件通知

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，之后单击目标Bucket名称。
3.
4.
5.
6. 单击**确定**。

   OSS事件通知完成后，会在消息服务MNS控制台的**主题**页面生成一条主题。

步骤三：创建MNS Token

1. 登录消息服务MNS控制台。
2. 在控制台左侧导航栏单击**主题列表**。
3. 在顶部菜单栏，选择地域。
4. 在**主题列表**页面单击目标主题。
5. 在**主题**页面单击**获取Endpoint**。
6. 在**主题详情**页面**接入点**区域复制公网访问链接。
7. 获取AK和SK信息。具体操作，请参见如何获取AccessKeyId和AccessKeySecret。
8. 执行以下命令，对获取到的公网访问链接、AK和SK进行Base64编码处理，生成访问Token。

   ```
   echo '{ "url":"https://xxxx.mns.cn-shanghai.aliyuncs.com/", "accessKeyId":"xxx","accessKeySecret":"xx" }' | base64
   ```

9. 创建Secret，用于管理和存储Token。
   i. 创建*mnsoss-secret.yaml*。

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    name: mnsoss-secret
   type: Opaque
   data:
    mns: eyAidXJsIjoiaHR0cHM6Ly94eHh4Lm1ucy5jbi1zaGFuZ2hhaS5hbGl5dW5jcy5jb20vIiwgImFjY2Vzc0tleUlkIjoieHh4IiwiYWNjZXNzS2V5U2VjcmV0IjoieHgiIH0K
   ```

   将 `mns` 替换成步骤生成的Token。
   ii. 执行以下命令，创建Secret。

   ```
   kubectl apply -f mnsoss-secret.yaml
   ```

步骤四：创建Service Account和Broker

1. 创建Service Account。
   i. 创建*mnsoss-sa.yaml*。

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
    name: eventing-sources-mnsoss
   subjects:
   - kind: ServiceAccount
    name: mnsoss-sa
    namespace: default
   roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: eventing-sources-mnsoss-controller
   ---
   apiVersion: v1
   kind: ServiceAccount
   metadata:
    name: mnsoss-sa
   ```

ii. 执行以下命令，创建Service Account。

```
kubectl apply -f mnsoss-sa.yaml
```

2. 执行以下命令，创建Broker。

```
kubectl label namespace default knative-eventing-injection=enabled
```

步骤五：创建MnsOss事件源

为了接收MnsOss产生的事件， 需要创建MnsOss Source用于接收事件。

1. 创建*mnsoss-source.yaml*。

```
apiVersion: sources.eventing.knative.dev/v1alpha1
kind: MnsOssSource
metadata:
 labels:
  controller-tools.k8s.io: "1.0"
 name: mnsoss-face
spec:
 # Add fields here
 serviceAccountName: mnsoss-sa
 accessToken:
  secretKeyRef:
   name: mnsoss-secret
   key: mns
 sink:
  apiVersion: eventing.knative.dev/v1alpha1
  kind: Broker
  name: default
 topic: mns-en-topics-oss-face-image-2381221888dds9129
```

`topic`：表示MNS主题名称。

2. 执行以下命令，创建MnsOss事件源。

```
kubectl  apply -f mnsoss-source.yaml
```

步骤六：创建Knative Service

为了验证MnsOss Source是否可以正常工作，需要创建Knative Service，以下使用的是event-display示例。

1. 创建*service.yaml*。

```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
 name: event-dispaly
 namespace: default
spec:
 template:
  spec:
   containers:
   - image: registry.cn-hangzhou.aliyuncs.com/knative-sample/event-display:1215
```

2. 执行以下命令，创建Knative Service。

```
kubectl apply -f service.yaml
```

步骤七：创建Trigger

创建Trigger， 订阅事件信息。

1. 创建*trigger.yaml*。

```
apiVersion: eventing.knative.dev/v1alpha1
kind: Trigger
metadata:
 name: oss-trigger
 namespace: default
spec:
 subscriber:
  ref:
   apiVersion: serving.knative.dev/v1alpha1
   kind: Service
   name: event-dispaly
```

2. 执行以下命令，创建Trigger。

```
kubectl apply -f trigger.yaml
```

执行结果

往OSS上传文件，会在Pod中收到事件通知。

```
2020/12/16 13:04:19 receive cloudevents.Event:
{"events": [{
        "eventName": "ObjectCreated:PostObject",
        "eventSource": "acs:oss",
        "eventTime": "2019-06-18T06:44:16.000Z",
        "eventVersion": "1.0",
        "oss": {
          "bucket": {
            "arn": "acs:oss:cn-beijing:1041208914252405:testjian",
            "name": "testjian",
            "ownerIdentity": "1041208914252405",
            "virtualBucket": ""},
          "object": {
            "deltaSize": 0,
            "eTag": "137138904F2E18D307D04EB38EA44CDA",
            "key": "timg.jpg",
            "size": 12990},
          "ossSchemaVersion": "1.0",
          "ruleId": "demo-i****"},
        "region": "cn-beijing",
        "requestParameters": {"sourceIPAddress": "42.120.7*.***"},
        "responseElements": {"requestId": "5D08884070BC12B192C6****"},
        "userIdentity": {"principalId": "104120891425****"}}]}
```

17.6. Knative最佳实践

17.6.1. 服务监控告警

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便的设置监控告警。

前提条件

- 您已经部署Knative Service服务，请参见快速部署Serverless应用。

- 您已连接到Kubernetes集群的Master节点，请参见通过kubectl工具连接集群。

操作步骤

1. 设置查询分析。

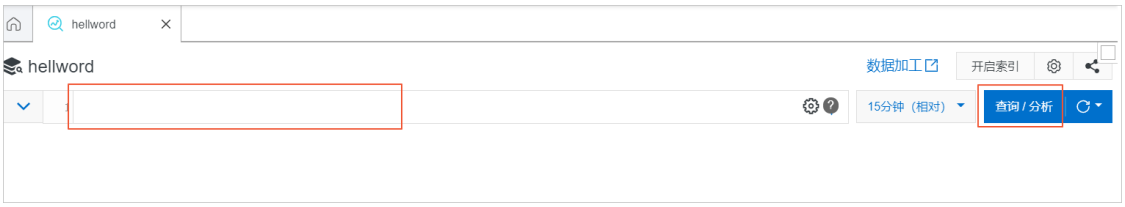
   请参见分析概述。

   i. 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。

   ii. 单击目标Logstore名称。

   iii. 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。

   请参见设置告警。

   i. 单击右上角的**另存为告警**。

   > **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，可以参见告警条件表达式语法，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。

ii. 在**告警规则**面板中填写告警配置信息。

配置项	说明
规则名称	告警规则的名称。名称长度为1~64个字符。
检查频率	服务端每次执行告警检查的时间。 **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。
仪表盘	选择或添加一个仪表盘。
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。
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。
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。
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。
通知策略	选择通知方式。可选择短信、语音、邮件、钉钉、WebHook或通知中心。

iii. 设置通知策略，完成后单击**确定**。

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1~256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1~500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**，请查看设置告警。 **说明** 每个机器人每分钟最多发送20条。
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。

17.6.2. 在Knative上实现日志采集

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，请参见通过日志服务采集Kubernetes容器日志。
- 您已经部署Knative Service服务，请参见快速部署Serverless应用。

操作步骤

1. 为Helloworld接入日志采集。

   请参见通过DaemonSet-控制台方式采集容器标准输出。

   i. 登录日志服务管理控制台。

   ii. 在**Project列表**区域，单击**创建Project**创建名称为hellword的项目。详细操作请参见管理Logstore。

   iii. 在helloworld项目页面的右上角，单击**接入数据**。

iv. 在**接入数据**页面选择**Docker标准输出**，进入**Docker标准输出**配置页面。



v. 在**选择日志空间**配置中，确认日志空间信息并单击**下一步**。

vi. 设置**创建机器组**。

若您已经安装Logtail日志组件，请点击**使用现有机器组**。
您可以选择*Kubernetes*或者*标准Docker采集*进行日志采集。具体操作，请参见通过日志服务采集Kubernetes容器日志和采集标准Docker容器日志。

vii. 单击**确认安装完毕**。

viii. 设置**机器组配置**。

ix. 设置**数据源设置**。

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"
  }
 ]
}
```

x. 数据源设置完成后，单击**下一步**。

xi. 在**查询分析配置**页面，开启**全文索引**，设置查询字段索引属性。

xii. 单击**下一步**进入**结束**页面。

2. 执行以下命令，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
```

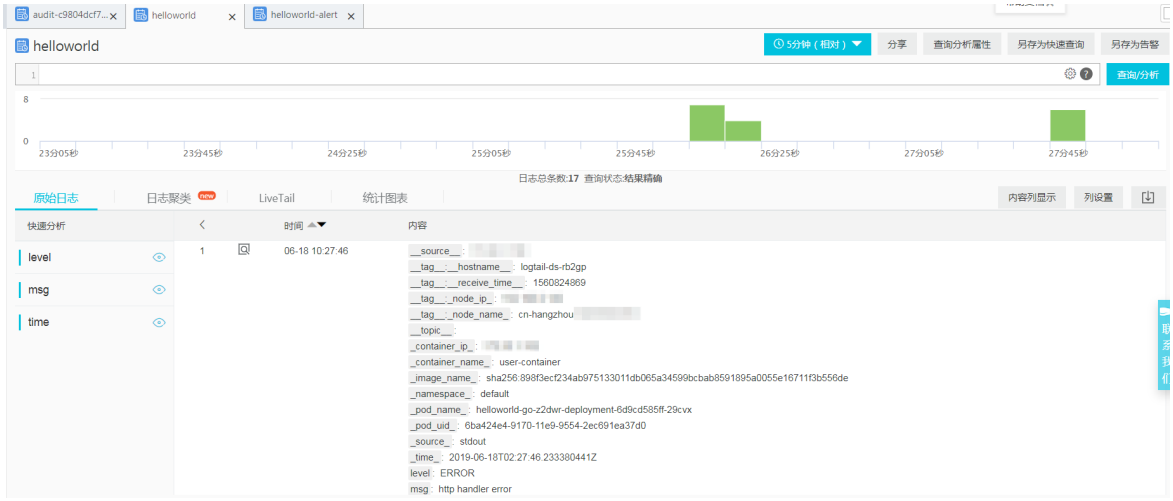
```
Hello Knative!
```

3. 在**日志库**的目标日志右侧悬浮鼠标至图标上，选择 > **查询分析**，可以看到该logstore的日志信息。

4. 设置查询分析。

请参见分析概述。

为了便于查看，您可以通过**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。

17.6.3. 在Knative中使用ECI

通过Knative和虚拟节点组件使用弹性容器实例ECI（Elastic Container Instance），可以实现按需使用资源。本文介绍两种方式说明如何在Knative中使用ECI。

前提条件

- 一键部署Knative
- 步骤一：在ACK集群中部署ack-virtual-node组件
- 通过kubectl工具连接集群

方式一：Knative中直接使用ECI

- 如果没有在namespace中配置Virtual Kubelet的标签，您只需在创建Knative Service资源对象时，定义以下配置直接创建ECI Pod：
  - 指定annotation规格。
  - 在节点亲和性nodeAffinity中指定virtual-kubelet。
  - 设置节点容忍标签为virtual-kubelet.io/provider。

  示例配置模板如下。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: helloworld-go
 namespace: vk
spec:
 template:
  metadata:
   annotations:
    k8s.aliyun.com/eci-use-specs: "2-4Gi"
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
       - key: type
         operator: In
         values:
         - virtual-kubelet
   tolerations:
   - key: virtual-kubelet.io/provider
    operator: Exists
   containers:
   - env:
    - name: TARGET
      value: "Knative"
    image: registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go:73fbdd56
```

- 如果在namespace中配置了以下Virtual Kubelet标签，您只需配置annotation指定ECI规格，然后直接创建ECI Pod。

  ```
  kubectl label namespace vk virtual-node-affinity-injection=enabled
  ```

  示例模板如下。

  ```
  apiVersion: serving.knative.dev/v1
  kind: Service
  metadata:
   name: helloworld-go
   namespace: vk
  spec:
   template:
    metadata:
     annotations:
      k8s.aliyun.com/eci-use-specs: "2-4Gi"
    spec:
     containers:
     - env:
      - name: TARGET
        value: "Knative"
      image: registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go:73fbdd56
  ```

方式二：Knative中指定ECS节点且支持自动创建ECI

通过设置节点亲和性nodeAffinity， 在Knative中可以指定添加标签的ECS节点，且Knative支持自动创建ECI，实现同一个服务的Pod既可以部署在ECS中，也可以部署在ECI中。

1. 给ECS节点添加标签。具体步骤请参见管理节点标签。

   本文给ECS节点添加标签示例为：*resource-role:ecs*。

2. 在Service资源对象模板中，配置节点亲和性nodeAffinity。

   通过配置nodeAffinity，指定添加标签的ECS和Virtual Kubelete。
   示例模板如下。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: helloworld-go
spec:
 template:
  metadata:
   annotations:
    k8s.aliyun.com/eci-use-specs: "2-4Gi"
  spec:
   containerConcurrency: 10
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
       - key: resource-role
        operator: In
        values:
        - ecs
      - matchExpressions:
       - key: type
        operator: In
        values:
        - virtual-kubelet
   tolerations:
   - key: virtual-kubelet.io/provider
    operator: Exists
   containers:
   - env:
    - name: TARGET
     value: "Knative"
    image: registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go:73fbdd56
```

执行结果

执行以下命令，可以看到成功创建2个ECI Pod。

```
kubectl -n vk get pod -o wide
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE  IP        NODE          NOMINATED NODE  READINESS G
ATES
helloworld-go-dqqhv-deployment-6d54c9c8dc-hkjwn  2/2   Running  0      40s  192.1xx.x.xx  virtual-node-eci-0  <n
one>       <none>
```

17.6.4. 在Knative中通过Prometheus采集Pod并发请求数

在实际服务中，您经常需要根据当前服务并发请求数来进行业务指标判断，而Knative正是通过queue-proxy容器实现了对请求指标的采集。本文介绍如何结合Prometheus呈现Knative服务中Pod请求指标。

步骤一：安装Prometheus监控组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的**阿里云应用**页签中，搜索并单击**ack-arms-prometheus**。
4. 在**应用目录 - ack-arms-prometheus**右侧的**创建**区域中，选择目标集群，然后单击**创建**。

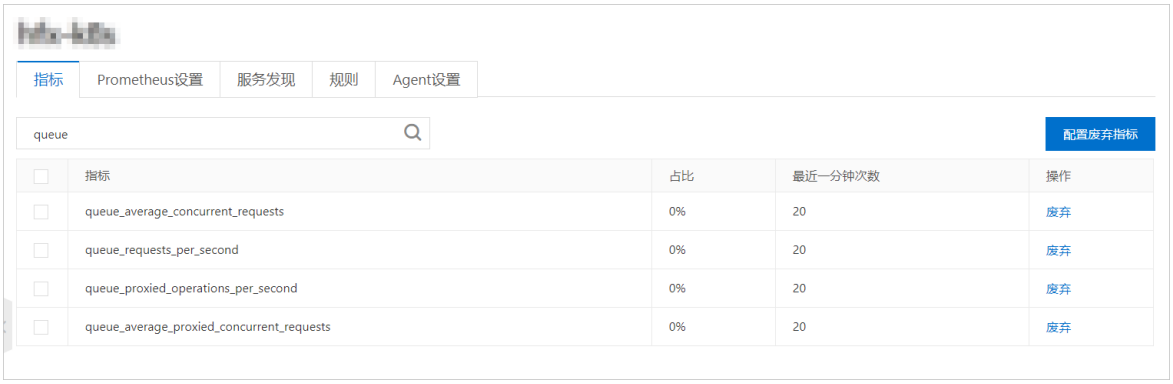
> **说明** 命名空间和发布名称默认为arms-prom。

步骤二：设置queue-proxy采集Job规则

1. 登录ARMS 控制台。
2. 在左侧导航栏中单击**Prometheus监控**。
3. 在**Prometheus监控**页面，单击目标集群**操作**列下的**设置**。
4. 单击**Prometheus设置**页签，在*Prometheus.yaml*文件中添加以下内容，然后单击**保存**。

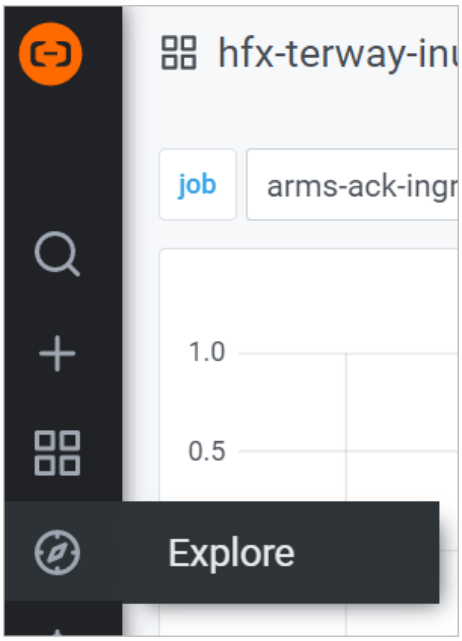
```
global:
  scrape_interval: 30s
  scrape_timeout: 10s
  evaluation_interval: 30s
scrape_configs:
- job_name: queue-proxy
  scrape_interval: 3s
  scrape_timeout: 3s
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  # Rename metadata labels to be reader friendly
  - source_labels: [__meta_kubernetes_pod_label_serving_knative_dev_revision, __meta_kubernetes_pod_container_port_name]
    action: keep
    regex: .+;http-autometric
  - source_labels: [__meta_kubernetes_namespace]
    target_label: namespace
  - source_labels: [__meta_kubernetes_pod_name]
    target_label: pod
  - source_labels: [__meta_kubernetes_service_name]
    target_label: service
```

5. 单击**指标**页签，查看queue-proxy相关指标。



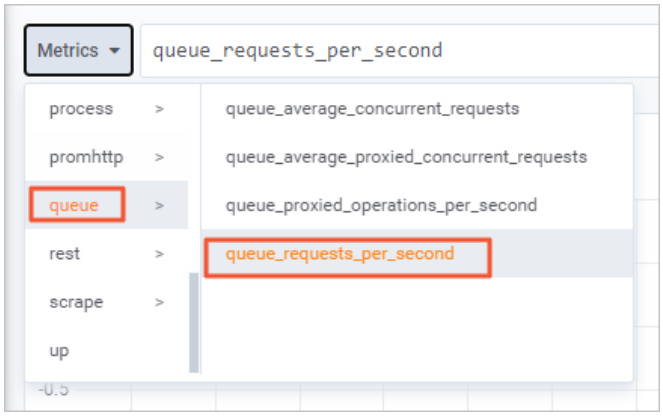
步骤三：获取Pod请求并发数

1. 在集群配置左侧导航栏，单击**大盘列表**。
2. 在**大盘列表**页面单击**Prometheus**。
3. 在**Prometheus**页面左侧，单击**Explore**图标。



4. 在**Metrics**指标下拉框中，选择queue-proxy指标进行查询。

   本文举例查询Pod每秒请求并发数指标**queue_requests_per_second**。



   您可以通过设置Namespace、Revision以及Pod名称查询Pod每秒请求连接数。示例如下。

```
queue_requests_per_second{destination_configuration="helloworld-go",destination_namespace="default",destination_pod="helloworld-go-ttf52-deployment-5778d86bd6-dnxw2"}
```



17.6.5. 在Knative中使用HPA

Knative支持HPA的弹性能力。您可以在Knative Service中设置CPU指标阈值，满足在突发高负载的场景下，自动扩缩容资源的诉求。本文介绍如何在Knative中使用HPA。

前提条件

一键部署Knative

操作步骤

1. 创建*ksvc-hpa.yaml*。

   在Knative Service指定使用HPA弹性策略，以下为配置示例。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: helloworld-go-hpa
spec:
 template:
  metadata:
   labels:
    app: helloworld-go-hpa
   annotations:
    autoscaling.knative.dev/class: "hpa.autoscaling.knative.dev"
    autoscaling.knative.dev/metric: "cpu"
    autoscaling.knative.dev/target: "75"
    autoscaling.knative.dev/minScale: "1"
    autoscaling.knative.dev/maxScale: "10"
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/knative-samples/helloworld-go:160e4dc8
      resources:
       requests:
        cpu: '200m'
```

   - 通过 `autoscaling.knative.dev/class: "hpa.autoscaling.knative.dev"` 指定HPA弹性插件。
   - 通过 `autoscaling.knative.dev/metric` 设置HPA CPU指标。
   - 通过 `autoscaling.knative.dev/target` 设置HPA CPU指标的阈值。
   - 通过 `autoscaling.knative.dev/minScale: "1"` 设置弹性策略实例数的最小值。

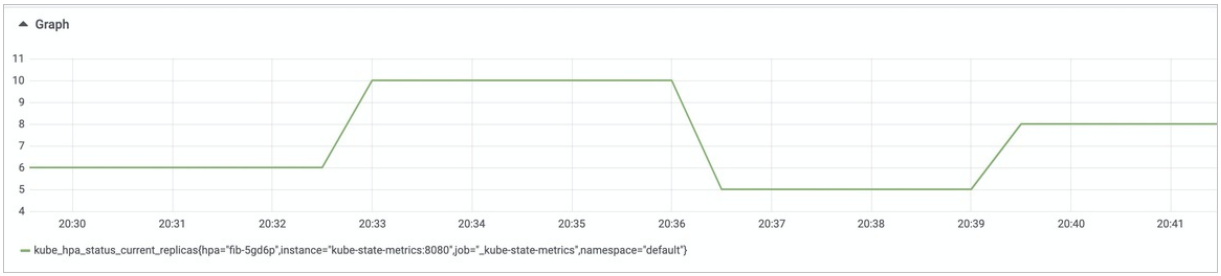
- 通过 `autoscaling.knative.dev/maxScale: "10"` 设置弹性策略实例数的最大值。

2. 执行HPA弹性策略。

```
kubectl apply -f ksvc-hpa.yaml
```

执行结果

Knative Service开启HPA弹性后，实例变化趋势如下图所示。

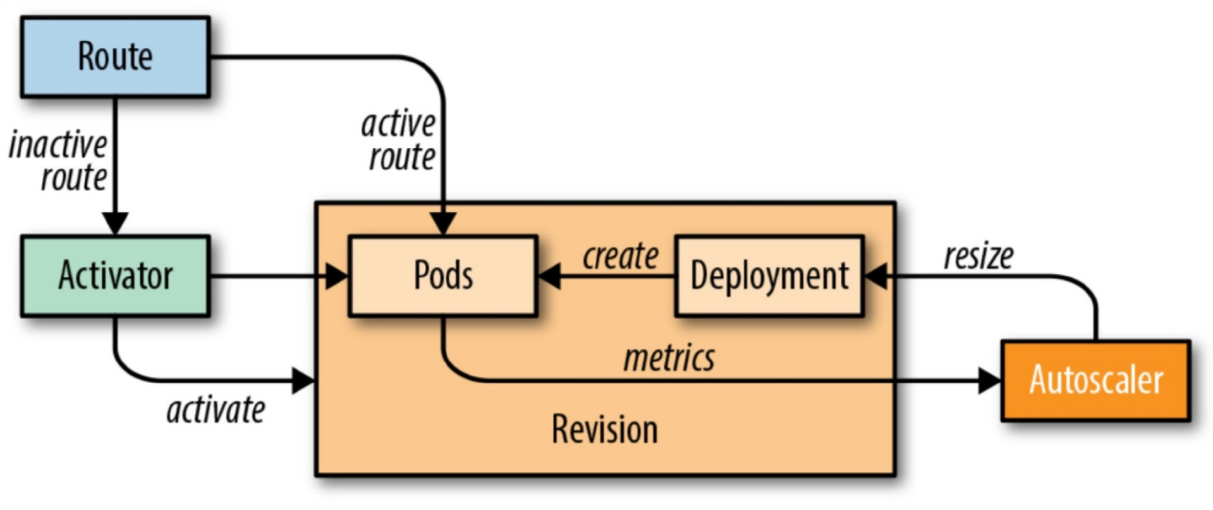


17.6.6. 基于流量请求数实现服务自动扩缩容

Knative中提供了开箱即用、基于流量请求的自动扩缩容KPA（Knative Pod Autoscaler）功能。本文介绍如何基于流量请求数实现服务自动扩缩容。

背景信息

Knative Serving为每个Pod注入QUEUE代理容器（queue-proxy），该容器负责向Autoscaler报告业务容器的并发指标。Autoscaler接收到这些指标之后，会根据并发请求数及相应的算法，调整Deployment的Pod数量，从而实现自动扩缩容。



并发数和QPS

- 并发数是同一时刻Pod的接收的请求数。
- QPS指的是Pod每秒响应的请求数，也就是最大吞吐能力。

并发数的增加并不一定会导致QPS增加。应用在访问压力较大的情况下，如果并发数增加，系统的QPS反而会下降。因为系统超负荷工作，CPU、内存等其他消耗导致系统性能下降，从而导致响应延迟。

算法

Autoscaler基于每个Pod的平均请求数（并发数）进行自动扩缩容，默认并发数为100。Pod数=并发请求总数/容器并发数。如果服务中并发数设置为10，并且加载了50个并发请求的服务，则Autoscaler就会创建5个Pod。

Autoscaler实现了两种操作模式的缩放算法，Stable稳定模式和Panic恐慌模式：

- Stable稳定模式。
  在稳定模式下，Autoscaler调整Deployment的大小，以实现每个Pod所需的平均并发数。Pod的并发数是根据60秒窗口内接收所有数据请求的平均数来计算的。
- Panic恐慌模式。
  Autoscaler计算60秒窗口内的平均并发数，系统需要1分钟稳定在所需的并发级别。但是，Autoscaler也会计算6秒的恐慌窗口，如果该窗口达到目标并发的2倍，则会进入恐慌模式。在恐慌模式下，Autoscaler在更短、更敏感的紧急窗口上工作。一旦紧急情况持续60秒后，Autoscaler将返回初始的60秒稳定窗口。

```
                        |
                Panic Target---> +--| 20
                        | |
                        |<------Panic Window
                        | |
    Stable Target---> +-------------------------|--| 10  CONCURRENCY
            |           | |
            |          <-----------Stable Window
            |           | |
------------------------+-------------------------+--+ 0
120             60              0
          TIME
```

KPA配置介绍

- 配置KPA，需要配置config-autoscaler，该参数默认已配置，以下为重点参数介绍。
  执行以下命令，查看config-autoscaler。

```
kubectl -n knative-serving get cm config-autoscaler
```

预期输出：

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: config-autoscaler
 namespace: knative-serving
data:
 container-concurrency-target-default: "100"
 container-concurrency-target-percentage: "0.7"
 enable-scale-to-zero: "true"
 max-scale-up-rate: "1000"
 max-scale-down-rate: "2"
 panic-window-percentage: "10"
 panic-threshold-percentage: "200"
 scale-to-zero-grace-period: "30s"
 scale-to-zero-Pod-retention-period: "0s"
 stable-window: "60s"
 target-burst-capacity: "200"
 requests-per-second-target-default: "200"
```

- 为KPA配置缩容至0。
  - scale-to-zero-grace-period：表示在缩为0之前，`inactive revison`保留的运行时间（最小是30s）。

    ```
    scale-to-zero-grace-period: 30s
    ```

  - stable-window：当在Stable模式运行中，Autoscaler在稳定窗口期下平均并发数下的操作。

    ```
    stable-window: 60s
    ```

    stable-window也可以在Revision注释中配置。

    ```
    autoscaling.knative.dev/window: 60s
    ```

  - enable-scale-to-zero：设置`enable-scale-to-zero`参数为`true`。

    ```
    enable-scale-to-zero: "true"
    ```

- 配置Autoscaler的并发数。
  - `target`
    `target`定义在给定时间（软限制）需要多少并发请求，是Knative中Autoscaler的推荐配置。在ConfigMap中默认配置的并发target为100。

    ```
    `container-concurrency-target-default: 100`
    ```

    这个值可以通过Revision中的`autoscaling.knative.dev/target`注释进行修改。

    ```
    autoscaling.knative.dev/target: 50
    ```

  - `containerConcurrencv`
    `containerConcurrency`限制在给定时间允许并发请求的数量（硬限制），并在Revision模板中配置。

    > **说明**
    > 使用`containerConcurrency`，需要满足以下条件：
    > - 只有在明确需要限制在给定时间有多少请求到达应用程序时，才应该使用`containerConcurrency`。
    > - 只有当应用程序需要强制的并发约束时，才建议使用`containerConcurrency`。

    ```
    containerConcurrency: 0 | 1 | 2-N
    ```

    - 1：确保一次只有一个请求由Revision给定的容器实例处理。
    - 2-N：请求的并发值限制为2或更多。
    - 0：表示不作限制，有系统自身决定。

  - container-concurrency-target-percentage
    这个参数为并发百分比或称为并发因子，并且会直接参与扩缩容并发数计算。例如，如果并发数target或者`containerConcurrency`设置值为100，并发因子container-concurrency-target-percentage为0.7。那么实际担当并发数达到70（100*0.7）时就会触发扩容操作。
    因此，实际扩缩容并发数=target（或者`containerConcurrency`）*container-concurrency-target-percentage

- 配置扩缩容边界，通过`minScale`和`maxScale`可以配置应用程序提供服务的最小和最大Pod数量。通过这两个参数配置可以控制服务冷启动或者控制计算成本。

> **说明**
> - 如果未设置 `minScale` 注释，Pods将缩放为零.。如果设置config-autoscaler的 `enable-scale-to-zero` 为 `false` ，则缩放为1。
> - 如果未设置 `maxScale` 注释，则创建的Pod数量将没有上限。

`minScale` 和 `maxScale` 可以在Revision模板中按照以下方式进行配置：

```
spec:
 template:
  metadata:
   autoscaling.knative.dev/minScale: "2"
   autoscaling.knative.dev/maxScale: "10"
```

场景一：设置并发请求数实现自动扩缩容

设置并发请求数，通过KPA实现自动扩缩容。

1. 快速部署Serverless应用。
2. 创建*autoscale-go.yaml*。

   设置当前最大并发请求数为10。

   ```
   apiVersion: serving.knative.dev/v1alpha1
   kind: Service
   metadata:
    name: autoscale-go
    namespace: default
   spec:
    template:
     metadata:
      labels:
       app: autoscale-go
      annotations:
       autoscaling.knative.dev/target: "10"
     spec:
      containers:
       - image: registry.cn-hangzhou.aliyuncs.com/knative-sample/autoscale-go:0.1
   ```

3. 执行以下命令，获取Kourier访问网关。

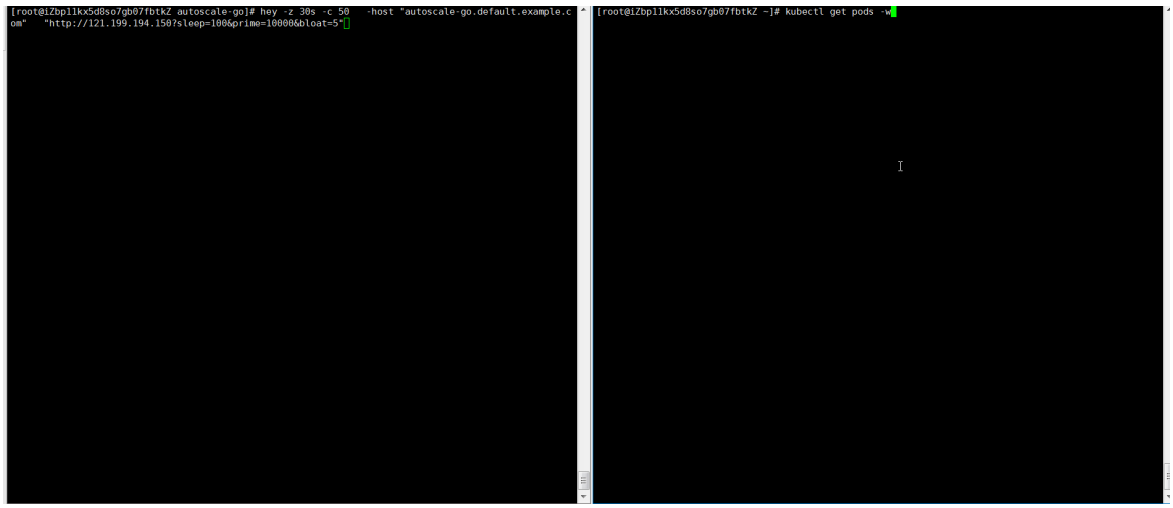
   ```
   kubectl -n knative-serving get svc kourier --output jsonpath="{.status.loadBalancer.ingress[*]['ip']}"
   ```

4. 使用Hey压测工具，执行30s内保持50个并发请求。

   > **说明** Hey压测工具的详细介绍，请参见Hey

   ```
   hey -z 30s -c 50  -host "autoscale-go.default.example.com"  "http://121.199.194.150?sleep=100&prime=10000&bloat=5"
   ```

   预期输出：



结果正如所预期的，扩容了5个Pod。

场景二：设置扩缩容边界实现自动扩缩容

扩缩容边界指应用程序提供服务的最小和最大Pod数量。通过设置应用程序提供服务的最小和最大Pod数量实现自动扩缩容。

1. 快速部署Serverless应用。
2. 创建*autoscale-go.yaml*。

   设置最大并发请求数为10，`minScale`最小保留实例数为1，`maxScale`最大扩容实例数为3。

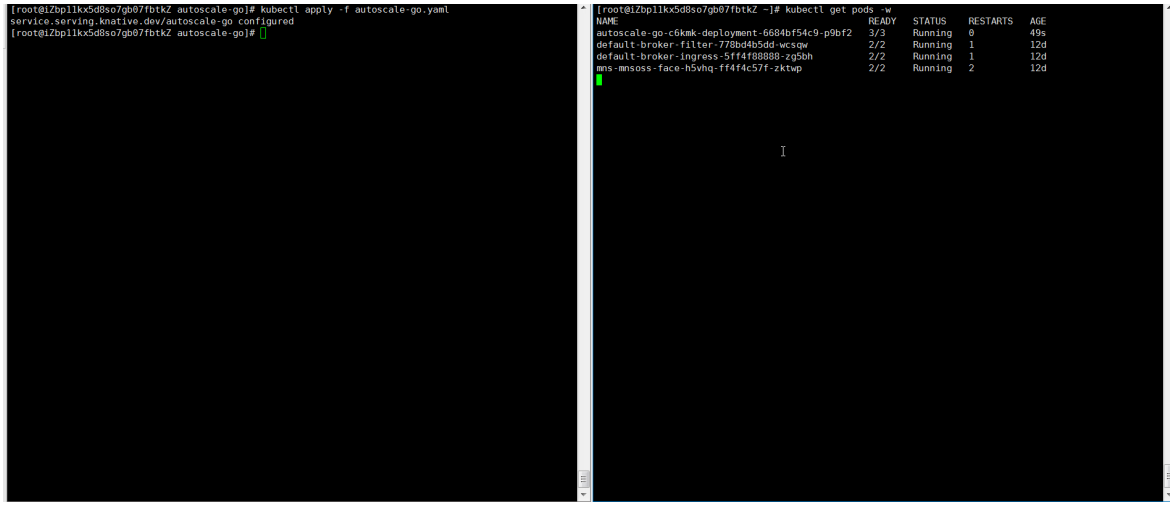
```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
 name: autoscale-go
 namespace: default
spec:
 template:
  metadata:
   labels:
    app: autoscale-go
   annotations:
    autoscaling.knative.dev/target: "10"
    autoscaling.knative.dev/minScale: "1"
    autoscaling.knative.dev/maxScale: "3"
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/knative-sample/autoscale-go:0.1
```

3. 使用Hey压测工具，执行30s内保持50个并发请求。

   > **说明** Hey压测工具的详细介绍，请参见Hey。

```
hey -z 30s -c 50  -host "autoscale-go.default.example.com"  "http://121.199.194.150?sleep=100&prime=10000&bloat=5"
```

预期输出：



结果正如所预期，最多扩容出来了3个Pod，并且即使在无访问请求流量的情况下，保持了1个运行的Pod。

18.GPU/NPU

18.1. 创建异构计算集群

18.1.1. 创建托管GPU集群

您可以通过容器服务管理控制台非常方便的创建异构计算托管集群。

背景信息

创建集群过程中，容器服务会进行如下操作：

- 创建ECS，配置管理节点到其他节点的SSH的公钥登录，通过CloudInit安装配置Kubernetes集群。
- 创建安全组，该安全组允许VPC入方向全部ICMP端口的访问。
- 如果您不使用已有的VPC网络，会为您创建一个新的VPC及VSwitch，同时为该VSwitch创建SNAT。
- 创建VPC路由规则。
- 创建NAT网关及EIP。
- 创建RAM用户和AccessKey，该RAM用户拥有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集群会根据您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。
- 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。如果您需要提高配额，请提交工单申请。
  - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。

    注意 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的Kubernetes集群的网络模式是Flannel时，集群的节点数最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多节点数，您需要先对目标VPC提交工单，申请提高配额。

  - 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：
  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管理控制台将按量付费转预付费，请参见按量付费转包年包月。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。

4. 在**选择集群模板**页面的**托管集群**区域，选择**异构计算托管集群**，并单击**创建**。
5. 在**ACK托管版**页面，完成集群配置。
   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 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（-）。
集群规格	选择所需版本，默认选中标准版。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。
资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择集群所在的资源组。这里显示选择的资源组。 
Kubernetes版本	选择Kubernetes版本，支持1.18.8-aliyun.1和1.16.9-aliyun.1两个版本。
容器运行时	支持**Containerd**、**Docker**和**安全沙箱**。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。

配置项	描述
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体可参见使用Terway网络插件。 ■ 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，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。 说明 ■ 在这两种模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 说明 ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Terway模式	网络插件选择**Terway**时，需要配置**Terway模式**。 配置**Terway模式**，需选择是否选中**Pod独占弹性网卡以获得最佳性能**。 ■ 选中**Pod独占弹性网卡以获得最佳性能**，Pod将独占一个专有的弹性网卡。 ■ 没有选中**Pod独占弹性网卡以获得最佳性能**，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。 说明 当前只有白名单用户可使用该功能。提交工单申请使用。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。

配置项	描述
配置SNAT	设置是否为专有网络配置SNAT网关。 ■ 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，可以取消选中页面下方的**为专有网络配置SNAT**。此时您需要自行配置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。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 **注意** 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。
标签	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。
集群本地域名	设置是否配置集群本地域名。 说明 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。 具体操作，请参见自定义集群API Server证书SAN。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影
Secret落盘加密	在异构计算集群中，选中**选择KMS密钥**可以使用在阿里云密钥管理服务KMS（Key Management Service）中创建的密钥加密Kubernetes Secret密钥。设置Secret落盘加密的详情，请参见使用阿里云KMS进行Secret的落盘加密。

6. 单击**下一步：节点池配置**，完成Worker节点配置。

i. 选择Worker实例。

■ 如果您选择**新增实例**，需要进行以下配置。

配置项	描述

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	选择**异构计算GPU/FPGA/NPU**和**GPU计算型**，详细的实例规格请参见实例规格族。 **说明** 当在实例规格列表里没有可用实例时，可在上一步**集群配置**页面选择其他虚拟交换机。 
已选规格	呈现选中的规格。
数量	新增Worker实例（ECS实例）的数量。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。 ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**云盘备份**。

配置项	描述
操作系统	ACK支持的节点操作系统包括： ■ Alibaba Cloud Linux 2.x（ACK默认的操作系统） 选中Alibaba Cloud Linux 2.x操作系统时，您可以进行以下安全加固的配置： ■ 不开启：不对Alibaba Cloud Linux 2.x操作系统进行安全加固。 ■ 等保加固：对Alibaba Cloud Linux 2操作系统进行等保加固。关于等保加固的详细信息，请参见ACK等保加固使用说明。 注意 等保加固在保障原生镜像兼容性和性能的基础上进行了等保合规适配，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》。 等保加固会禁止Root用户通过SSH远程登录。您可通过ECS控制台使用VNC方式登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。 ■ CIS加固：对Alibaba Cloud Linux 2.x操作系统进行CIS加固。关于CIS加固的使用说明，请参见ACK CIS加固使用说明。 ■ CentOS 7.x 说明 暂不支持CentOS 8.x及以上的操作系统。
登录方式 **密钥对**	■ 设置密钥。 您需要在创建集群的时候选择密钥对登录方式，单击**新建密钥对**，跳转到ECS云服务器控制台，创建密钥对，参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 说明 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

■ 如果您选择**添加已有实例**，则需要预先在此地域下创建ECS云服务器，然后配置**操作系统**、**登录方式**和**密钥对**，配置描述参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义镜像	允许您选择一个自定义ECS镜像。选择自定义镜像后，集群所有节点将基于此镜像进行部署 。有关创建自定义镜像操作，请参见使用自定义镜像创建Kubernetes集群。 **说明** ■ 目前自定义镜像的操作系统仅支持CentOS 7.x和Alibaba Cloud Linux 2.x。 ■ 当前只有白名单用户可使用该功能。提交工单申请使用。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

7. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认选中**安装Ingress组件**，请参见Ingress高级用法。 **说明** 如果您选中**创建Ingress Dashboard**，则需要同步安装日志服务组件。
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume和存储管理-CSI。
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。

配置项	描述
报警配置	选中**使用默认报警模板配置报警**，开启默认报警规则。详细介绍，请参见容器服务报警管理。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**使用日志服务**，您还可以选择是否在日志服务控制台中**创建Ingress Dashboard**或者**安装node-problem-detector并创建事件中心**。
控制平面组件日志	选中**开启**，将从ACK控制层收集托管集群控制平面组件日志到您账号中的SLS日志服务的Log Project中。更多信息，请参见收集托管集群控制平面组件日志。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

8. 单击**下一步：确认配置**，
9. 选中**服务协议**并单击**创建集群**。

说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

相关操作

集群创建成功后，在**集群列表**页面中，单击目标集群名称或者目标集群**操作**列下的**详情**。在左侧导航栏中，选择**节点管理 > 节点**，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。

18.1.2. 创建专有GPU集群

您可以通过容器服务管理控制台非常方便的创建异构计算专有集群。

背景信息

创建集群过程中，容器服务会进行如下操作：

- 创建ECS，配置管理节点到其他节点的SSH的公钥登录，通过CloudInit安装配置Kubernetes集群。
- 创建安全组，该安全组允许VPC入方向全部ICMP端口的访问。
- 如果您不使用已有的VPC网络，会为您创建一个新的VPC及VSwitch，同时为该VSwitch创建SNAT。
- 创建VPC路由规则。
- 创建NAT网关及EIP。
- 创建RAM用户和AccessKey，该RAM用户拥有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集群会根据您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。
- 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。如果您需要提高配额，请提交工单申请。
  - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。

    > **注意** 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的Kubernetes集群的网络模式是Flannel时，集群的节点数最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多节点数，您需要先对目标VPC提交工单，申请提高配额。

  - 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：
  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管理控制台将按量付费转预付费，请参见按量付费转包年包月。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**页面的**其他集群**区域，选择**异构计算专有集群**，并单击**创建**。
5. 在**ACK专有版**页面，完成集群配置。
   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。

配置项	描述
资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择集群所在的资源组。这里显示选择的资源组。 
Kubernetes版本	选择Kubernetes版本。
容器运行时	支持**Containerd**、**Docker**和**安全沙箱**。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体请参见Terway与Flannel对比。 ■ 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，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。 **说明** ■ 在Terway模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。 ■ 网络插件选择**Terway**时，会使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。

配置项	描述
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 说明 ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Terway模式	网络插件选择**Terway**时，需要配置**Terway模式**。 配置**Terway模式**，需选择是否选中**Pod独占弹性网卡以获得最佳性能**。 ■ 选中**Pod独占弹性网卡以获得最佳性能**，Pod将独占一个专有的弹性网卡。 ■ 没有选中**Pod独占弹性网卡以获得最佳性能**，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。 说明 当前只有白名单用户可使用该功能。提交工单申请使用。
Pod虚拟交换机	网络插件选择**Terway**时，需要配置**Pod虚拟交换机**。**Pod虚拟交换机**指定Terway网络插件需要配置Pod的VSwitch，由于Pod的弹性网卡需要和所在的机器在同一个可用区，所以需要为上面所选的集群节点的VSwitch相对应可用区的Pod选择另一个VSwitch，集群运行时Pod的IP地址将会从所选择的VSwitch中分配，为了保障Pod可用数量，建议选择掩码不大于19的VSwitch。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。
配置SNAT	设置是否为专有网络配置SNAT网关。 ■ 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，可以取消选中页面下方的**为专有网络配置SNAT**。此时您需要自行配置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。

配置项	描述
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 注意 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
SSH登录	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，具体操作参见通过SSH连接ACK专有版集群的Master节点。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。
标签	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。
自定义镜像	选择自定义镜像时，将取代默认系统镜像。
集群本地域名	设置是否配置集群本地域名。 说明 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	自定义配置集群apiserver服务端证书中的SAN。多个IP或域名以英文逗号（,）分隔。
服务帐户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用ServiceAccount遇到的安全性问题。详情请参见部署服务账户令牌卷投影。
集群CA	如果选中自定义集群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。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。

6. 单击**下一步：Master配置**，完成Master节点配置。

配置项	描述
Master实例数量	设置您所需的Master节点数量。目前支持创建3个或者5个Master节点。

配置项	描述
实例规格	选择Master节点的实例规格，详细的实例规格请参见实例规格族 **说明** 当在实例规格列表里没有可用实例时，可在**集群配置**页面选择其他虚拟交换机。
系统盘	默认为Master节点挂载系统盘， 支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ○ 支持选中**开启云盘备份**以备份云盘数据。 ○ **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。

7. 单击**下一步：节点池配置**，完成Worker节点配置。
   i. 选择Worker实例。
      - 如果您选择**新增实例**，需要进行以下配置。

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	选择**异构计算GPU/FPGA/NPU**和**GPU计算型**，详细的实例规格请参见实例规格族。 **说明** 当在实例规格列表里没有可用实例时，可在**集群配置**页面选择其他虚拟交换机。
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**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** - 支持选中**开启云盘备份**以备份云盘数据。 - **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。

配置项	描述
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**云盘备份**。
操作系统	ACK支持的节点操作系统包括： ■ Alibaba Cloud Linux 2.x（ACK默认的操作系统） 选中Alibaba Cloud Linux 2.x操作系统时，您可以进行以下安全加固的配置： ■ 不开启：不对Alibaba Cloud Linux 2.x操作系统进行安全加固。 ■ 等保加固：对Alibaba Cloud Linux 2操作系统进行等保加固。关于等保加固的详细信息，请参见ACK等保加固使用说明。 **注意** 等保加固在保障原生镜像兼容性和性能的基础上进行了等保合规适配，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》。 等保加固会禁止Root用户通过SSH远程登录。您可通过ECS控制台使用VNC方式登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。 ■ CIS加固：对Alibaba Cloud Linux 2.x操作系统进行CIS加固。关于CIS加固的使用说明，请参见ACK CIS加固使用说明。 ■ CentOS 7.x **说明** 暂不支持CentOS 8.x及以上的操作系统。
登录方式	■ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

- 如果您选择**添加已有实例**，则需要预先在此地域下创建ECS云服务器，然后配置**操作系统**、**登录方式**和**密钥对**，配置描述参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀 + 节点IP地址子串 + 后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和-，且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.168.0.55，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
节点服务端口范围	设置节点服务端口范围。默认端口范围为30000~32767。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

8. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认选中**安装Ingress组件** ，请参见Ingress高级用法 。
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。 NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume 和存储管理-CSI。
监控插件	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。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。 选中**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，详情参见通过日志服务采集Kubernetes容器日志。 选中**使用日志服务**后，您还可以选择是否在日志服务控制台中**创建Ingress Dashboard**或者**安装node-problem-detector并创建事件中心**。

配置项	描述
工作流引擎	设置是否使用AGS。 **说明** 目前只有白名单用户才能使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果没有选中AGS，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

9. 单击**下一步：确认配置**，
10. 选中**服务协议**并单击**创建集群**。

说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

相关操作

集群创建成功后，在**集群列表**页面中，单击目标集群名称或者目标集群**操作**列下的**详情**。在左侧导航栏中，选择**节点管理 > 节点**，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。

18.1.3. 创建NPU集群

您可以通过容器服务管理控制台非常方便的创建带有NPU资源的Kubernetes集群。

背景信息

创建集群过程中，容器服务会进行如下操作：

- 创建ECS，配置管理节点到其他节点的SSH的公钥登录，通过CloudInit安装配置Kubernetes集群。
- 创建安全组，该安全组允许VPC入方向全部ICMP端口的访问。
- 如果您不使用已有的VPC网络，会为您创建一个新的VPC及VSwitch，同时为该VSwitch创建SNAT。
- 创建VPC路由规则。
- 创建NAT网关及EIP。
- 创建RAM用户和AccessKey，该RAM用户拥有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集群会根据您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。
- 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。如果您需要提高配额，请提交工单申请。
  - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。

注意 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的Kubernetes集群的网络模式是Flannel时，集群的节点数最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多节点数，您需要先对目标VPC提交工单，申请提高配额。

- 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：
  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管理控制台将按量付费转预付费，请参见按量付费转包年包月。

创建NPU型Kubernetes集群

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**页面，选择**异构计算专有集群**页面，并单击**创建**。

   本例中创建的是异构计算专有集群，您也可以在选择集群模板页面选择**异构计算托管集群**并进行以下操作。

   > **说明** 为了创建NPU集群，通常情况下，Worker节点使用NPU类型的ECS。集群其他参数配置，请参见创建Kubernetes托管版集群。

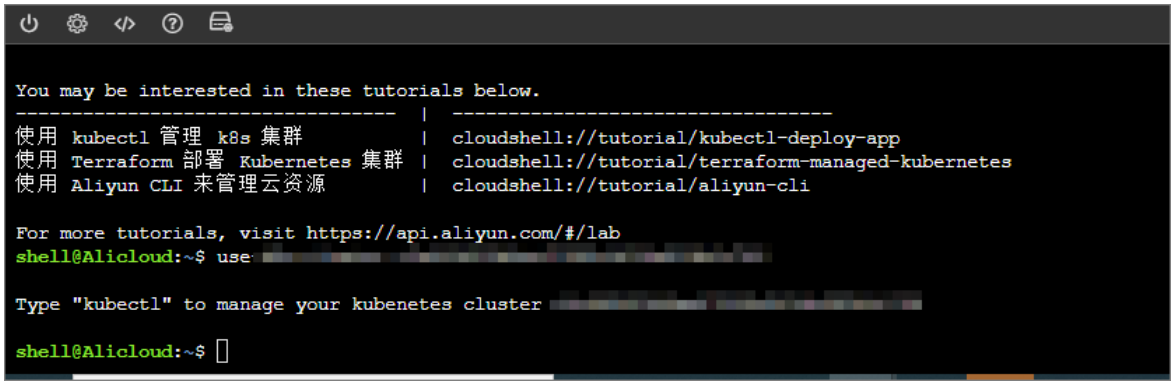
5. 设置节点池的配置信息。本例中将Worker节点作为NPU工作节点，选择NPU计算型实例规格ecs.ebman1.26xlarge。
   - 若您选择新增实例，则需要选择Worker节点的系列和规格，以及需要创建的Worker节点的数量（本示例创建2个NPU节点，实例类型为ecs.ebman1.26xlarge）。
   - 若您选择添加已有实例，则需要预先在此地域下创建NPU云服务器。请参见实例规格族。
6. 完成其他配置后，单击**创建集群**，启动部署。
7. 查看节点挂载的NPU设备。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
   iii. 在集群管理页面单击左侧导航栏选择**节点管理 > 节点**。
   iv. 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NPU设备。

配置私有镜像密钥

如果您需要使用阿里云提供的NPU类型的Docker镜像，可以联系与您对接的客户经理或销售人员，获取一个已授权的账号。下载Docker镜像并在Kubernetes集群中配置私有镜像密钥。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面单击目标集群右侧**操作**列的**更多 > 通过CloudShell管理集群**。

   集群连接成功后，界面显示如下。



4. 执行以下命令，创建一个docker-registry类型的Secret。

```
kubectl create secret \
docker-registry regsecret \
--docker-server=registry.cn-shanghai.aliyuncs.com \
--docker-username=<your_username>
--docker-password=<your_password>
```

> 说明
> - regsecret：指定密钥的键名称，可自行定义。
> - --docker-server：指Docker仓库地址。
> - --docker-username：获取的账号。
> - --docker-password：获取的密码。

5. 在Pod的配置文件中添加Secret，拉取NPU类型的私有镜像。

```
apiVersion: v1
kind: Pod
metadata:
 name: test-npu
spec:
 containers:
 - name: <容器名称>
  image: registry.cn-shanghai.aliyuncs.com/hgai/<NPU类型的docker镜像>
 imagePullSecrets:
 - name: <secret名称>
```

> 说明
> - imagePullSecrets 是声明拉取镜像时需要指定密钥。
> - regsecret 必须和步骤生成密钥的键名一致。
> - image 中的Docker仓库名称必须和 --docker-server 中的Docker仓库名一致。

使用NPU集群

通过Kubernetes管理ALI NPU设备，能够帮助您快速部署AI推理任务，监控NPU资源使用情况等。具体操作，请参见Kubernetes集群支持NPU调度。

18.1.4. 创建托管FPGA集群

您可以通过容器服务管理控制台非常方便的创建托管FPGA集群。

前提条件

- 根据各个地域和可用区支持的FPGA实例规格情况，选择创建托管FPGA集群的地域。关于ECS实例规格可购买地域总览，请参见ECS 实例规格可购买地域总览。
- FPGA实例需要使用包括Xilinx开发环境的镜像，目前只能通过共享镜像的方式提供，请向FaaS（FPGA as a service）产品提交工单申请共享镜像。
- 基于共享镜像，复制一个自定义镜像*faas_f3*。具体操作，请参见复制镜像。

背景信息

创建集群过程中，容器服务会进行如下操作：

- 创建ECS，配置管理节点到其他节点的SSH的公钥登录，通过CloudInit安装配置Kubernetes集群。
- 创建安全组，该安全组允许VPC入方向全部ICMP端口的访问。
- 如果您不使用已有的VPC网络，会为您创建一个新的VPC及VSwitch，同时为该VSwitch创建SNAT。
- 创建VPC路由规则。
- 创建NAT网关及EIP。
- 创建RAM用户和AccessKey，该RAM用户拥有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集群会根据您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。
- 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。如果您需要提高配额，请提交工单申请。
  - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。

    注意 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的Kubernetes集群的网络模式是Flannel时，集群的节点数最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多节点数，您需要先对目标VPC提交工单，申请提高配额。
  - 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：
  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管理控制台将按量付费转预付费，请参见按量付费转包年包月。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**页面的**托管集群**区域，选择**异构计算托管集群**，并单击**创建**。
5. 在**ACK托管版**页面，完成集群配置。

i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 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（-）。
集群规格	选择所需版本，默认选中标准版。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。
资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择集群所在的资源组。这里显示选择的资源组。 
Kubernetes版本	选择Kubernetes版本，支持1.18.8-aliyun.1和1.16.9-aliyun.1两个版本。
容器运行时	支持**Containerd**、**Docker**和**安全沙箱**。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。

配置项	描述
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体可参见使用Terway网络插件。 ■ 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，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。 说明 ■ 在这两种模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 说明 ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Terway模式	网络插件选择**Terway**时，需要配置**Terway模式**。 配置**Terway模式**，需选择是否选中**Pod独占弹性网卡以获得最佳性能**。 ■ 选中**Pod独占弹性网卡以获得最佳性能**，Pod将独占一个专有的弹性网卡。 ■ 没有选中**Pod独占弹性网卡以获得最佳性能**，使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。 说明 当前只有白名单用户可使用该功能。提交工单申请使用。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。

配置项	描述
配置SNAT	设置是否为专有网络配置SNAT网关。 ■ 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，可以取消选中页面下方的**为专有网络配置SNAT**。此时您需要自行配置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。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 **注意** 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。

配置项	描述
标签	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。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。 具体操作，请参见自定义集群API Server证书SAN。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影
Secret落盘加密	在异构计算集群中，选中**选择KMS密钥**可以使用在阿里云密钥管理服务KMS（Key Management Service）中创建的密钥加密Kubernetes Secret密钥。设置Secret落盘加密的详情，请参见使用阿里云KMS进行Secret的落盘加密。

6. 单击**下一步：节点池配置**，完成Worker节点配置。
   i. 选择Worker实例。
   - 如果您选择**新增实例**，需要进行以下配置。

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
付费类型	支持**按量付费**和**包年包月**两种节点付费类型。
购买时长	选择**包年包月**时，需设置购买时长。目前支持选择1、2、3、6个月和1~5年。
自动续费	选择**包年包月**，需设置是否**自动续费**。

配置项	描述
实例规格	选择**异构计算GPU/FPGA/NPU**和**FPGA计算型**，详细的实例规格请参见实例规格族。 **说明** 当在实例规格列表里没有可用实例时，可在上一步**集群配置**页面选择其他虚拟交换机。 
已选规格	呈现选中的规格。
数量	新增Worker实例（ECS实例）的数量。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。 ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**云盘备份**。

配置项	描述
操作系统	ACK支持的节点操作系统包括： ■ Alibaba Cloud Linux 2.x（ACK默认的操作系统） 选中Alibaba Cloud Linux 2.x操作系统时，您可以进行以下安全加固的配置： ■ 不开启：不对Alibaba Cloud Linux 2.x操作系统进行安全加固。 ■ 等保加固：对Alibaba Cloud Linux 2操作系统进行等保加固。关于等保加固的详细信息，请参见ACK等保加固使用说明。 注意 等保加固在保障原生镜像兼容性和性能的基础上进行了等保合规适配，使其满足国家信息安全部发布的《GB/T22239-2019信息安全技术网络安全等级保护基本要求》。 等保加固会禁止Root用户通过SSH远程登录。您可通过ECS控制台使用VNC方式登录系统创建可使用SSH的普通用户。具体操作，请参见通过密码认证登录Linux实例。 ■ CIS加固：对Alibaba Cloud Linux 2.x操作系统进行CIS加固。关于CIS加固的使用说明，请参见ACK CIS加固使用说明。 ■ CentOS 7.x 说明 暂不支持CentOS 8.x及以上的操作系统。
登录方式 **密钥对**	■ 设置密钥。 您需要在创建集群的时候选择密钥对登录方式，单击**新建密钥对**，跳转到ECS云服务器控制台，创建密钥对，参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 说明 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

■ 如果您选择**添加已有实例**，则需要预先在此地域下创建ECS云服务器，然后配置**操作系统**、**登录方式**和**密钥对**，配置描述参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义镜像	允许您选择一个自定义ECS镜像。选择自定义镜像*faas_f3*后，集群所有节点将基于此镜像进行部署 。有关创建自定义镜像操作，请参见使用自定义镜像创建Kubernetes集群。  **说明** ■ 目前自定义镜像的操作系统仅支持CentOS 7.x和Alibaba Cloud Linux 2.x。 ■ 当前只有白名单用户可使用该功能。提交工单申请使用。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

7. 单击**下一步：组件配置**，完成组件配置。

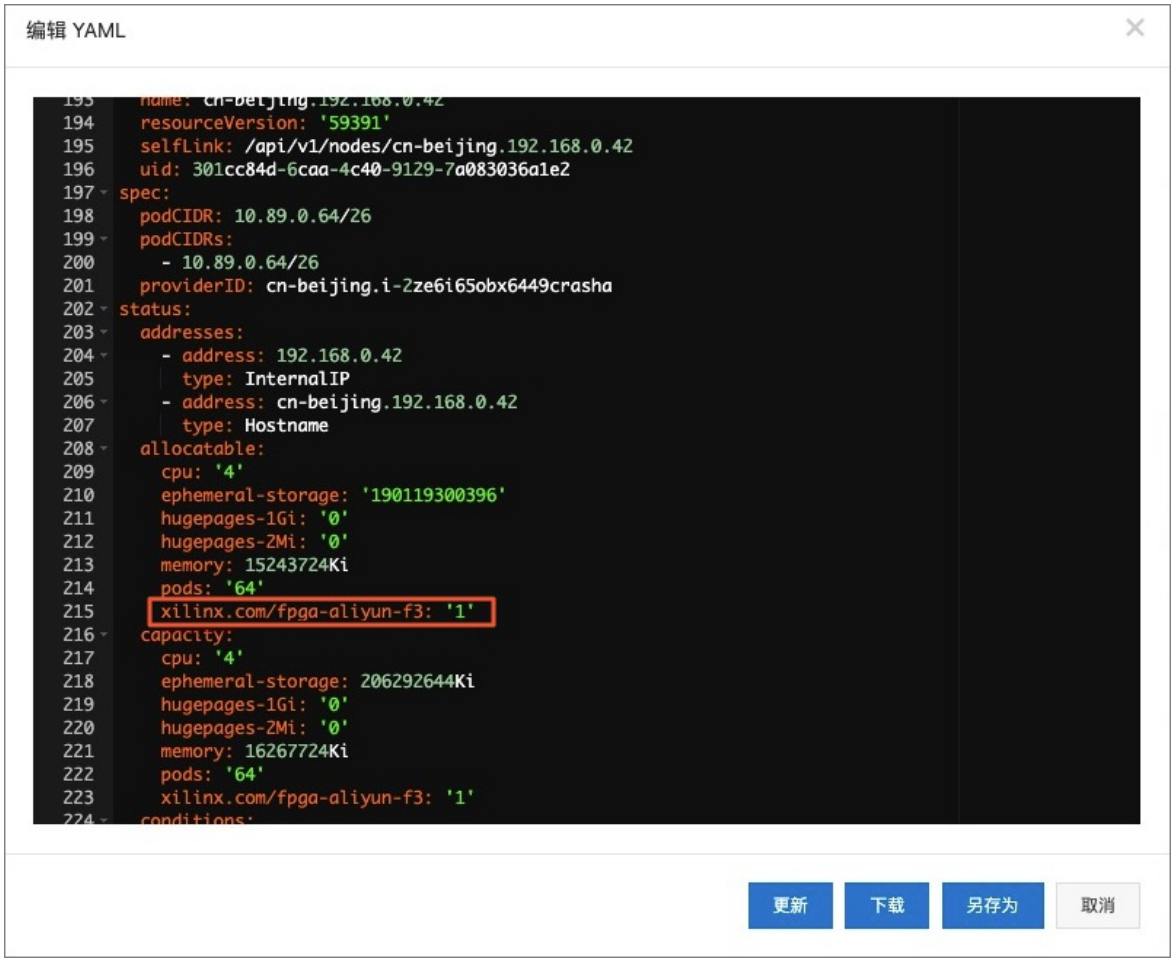
配置项	描述
Ingress	设置是否安装Ingress组件。默认选中**安装Ingress组件**，请参见Ingress高级用法。 **说明** 如果您选中**创建Ingress Dashboard**，则需要同步安装日志服务组件。

配置项	描述
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume和存储管理-CSI。
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。
报警配置	选中**使用默认报警模板配置报警**，开启默认报警规则。详细介绍，请参见容器服务报警管理。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**使用日志服务**，您还可以选择是否在日志服务控制台中**创建Ingress Dashboard**或者**安装node-problem-detector并创建事件中心**。
控制平面组件日志	选中**开启**，将从ACK控制层收集托管集群控制平面组件日志到您账号中的SLS日志服务的Log Project中。更多信息，请参见收集托管集群控制平面组件日志。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

8. 单击**下一步：确认配置**，
9. 选中**服务协议**并单击**创建集群**。

说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

集群创建成功后，在**集群列表**页面中，单击目标集群名称或者目标集群**操作**列下的**详情**。在左侧导航栏中，选择**节点管理 > 节点**，选择创建集群时配置的Worker节点，单击**操作**列的**更多 > 详情**，单击右上角的**YAML**查看该节点挂载的FPGA设备资源。



18.2. GPU调度

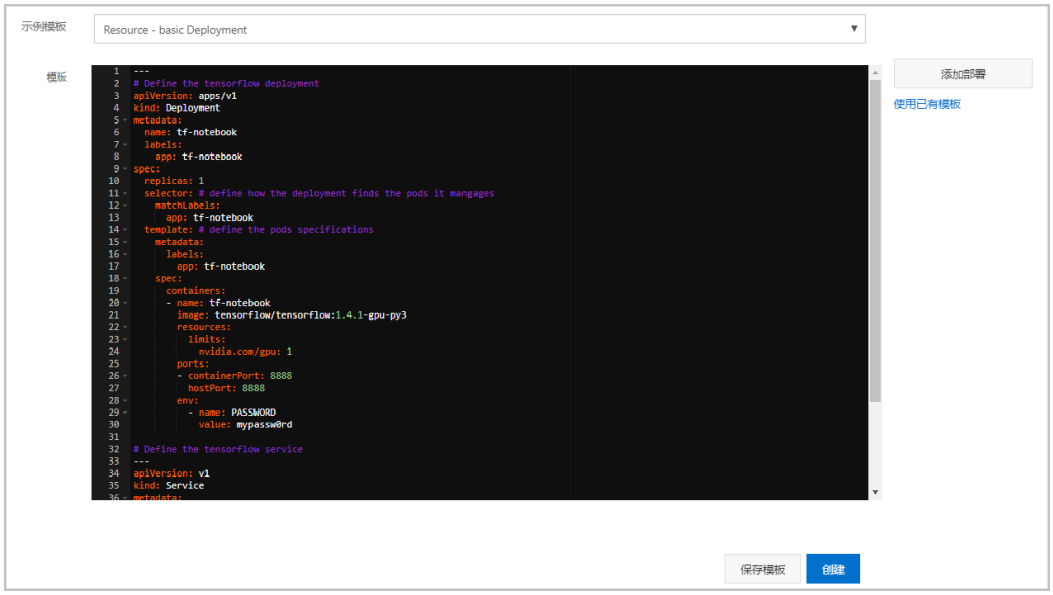
18.2.1. 使用Kubernetes默认GPU调度

本文通过运行TensorFlow的GPU实验环境为您介绍支持GPU调度的Kubernetes GPU集群。

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，本文用一个例子向您展示如何快速部署一个Jupyter应用。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在左侧导航栏选择**工作负载 > 无状态**。
5. 在**无状态**页面右上角单击**使用YAML创建资源**。
6. 选择所需命名空间，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: mypassword
# 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，您必须要定义以下的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
```

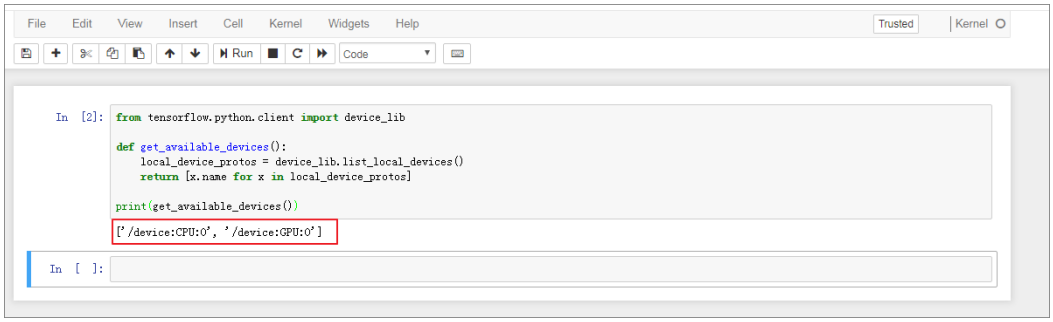
这需要您在编写部署文件时，强依赖于所在的集群，导致缺乏可移植性。但是在Kubernetes 1.9.3及之后的版本中，无需指定hostPath，nvidia的插件会自发现驱动所需的库链接和执行文件。

7. 在左侧导航栏选择**网络 > 服务**，选择所需命名空间，选择tf-notebook服务，查看外部端点。



8. 在浏览器中访问Jupyter实例，访问地址是 `http://外部端点` ，输入模板中配置的密码。
9. 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())
```



18.2.2. 使用节点标签自定义GPU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，参见使用Kubernetes默认GPU调度。
- 您已连接到Master节点，方便快速查看节点标签等信息，参见通过kubectl工具连接集群。

背景信息

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

- 可以快速筛选GPU节点。
- 部署时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. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏选择**节点管理 > 节点**。
5. 在**节点列表**页面中，选择GPU节点，单击**操作**列的**更多 > 详情**。

   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。
执行命令：

```
kubectl get nodes
```

返回值：

```
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
```

返回值：

```
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-2ze8o1a45qdv5q8a7luz
.....
```

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

key	value
aliyun.accelerator/nvidia_mem	GPU显存，单位为MiB
aliyun.accelerator/nvidia_name	nvidia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。
执行命令：

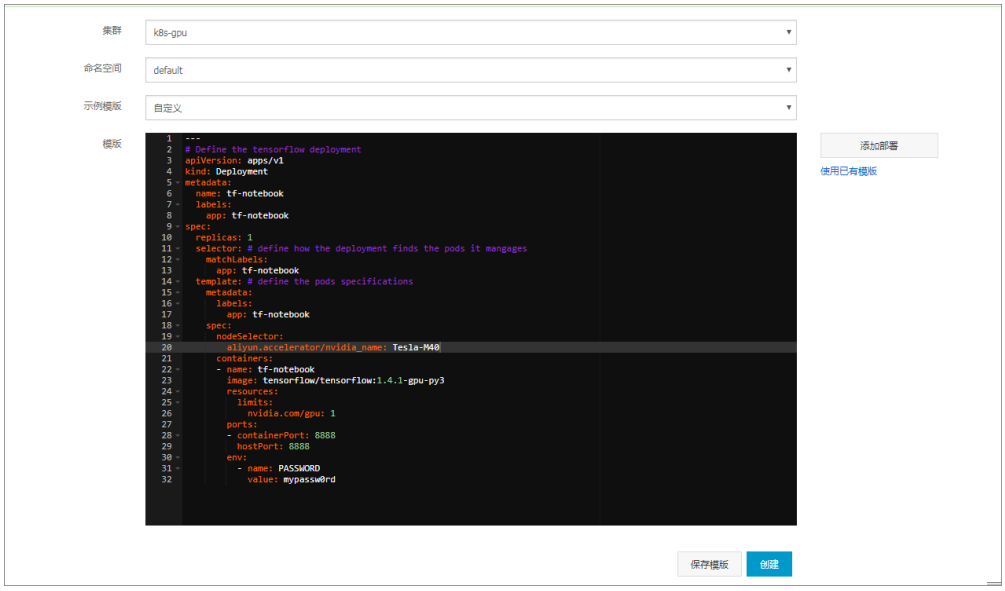
```
kubectl get no -l aliyun.accelerator/nvidia_name=Tesla-M40
```

返回值：

```
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
```

6. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**，然后在**无状态**页面中，单击**使用模板创建**。

i. 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: #指定节点标签，该应用将被调度到具有标签aliyun.accelerator/nvidia_name=Tesla-M40的节点上
    aliyun.accelerator/nvidia_name: Tesla-M40
   containers:
   - name: tf-notebook
    image: tensorflow/tensorflow:1.4.1-gpu-py3
    resources:
     limits:
      nvidia.com/gpu: 1 #该应用需要申请一张GPU卡
    ports:
    - containerPort: 8888
     hostPort: 8888
    env:
     - name: PASSWORD
      value: mypassw0rdv # 设置该应用的登录密码
```

ii. 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，具体参见创建无状态工作负载Deployment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
```

7. 在集群管理页左侧导航栏选择**工作负载 > 容器组**。
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。

18.2.3. ACK调度GPU使用的节点标签说明

安装ack-ai-installer组件之后，您可以为GPU节点打上一些特定标签，从而启用不同的功能，包括GPU共享调度、GPU显存隔离和GPU拓扑感知功能。本文介绍GPU节点的特定标签和GPU功能。

节点标签说明

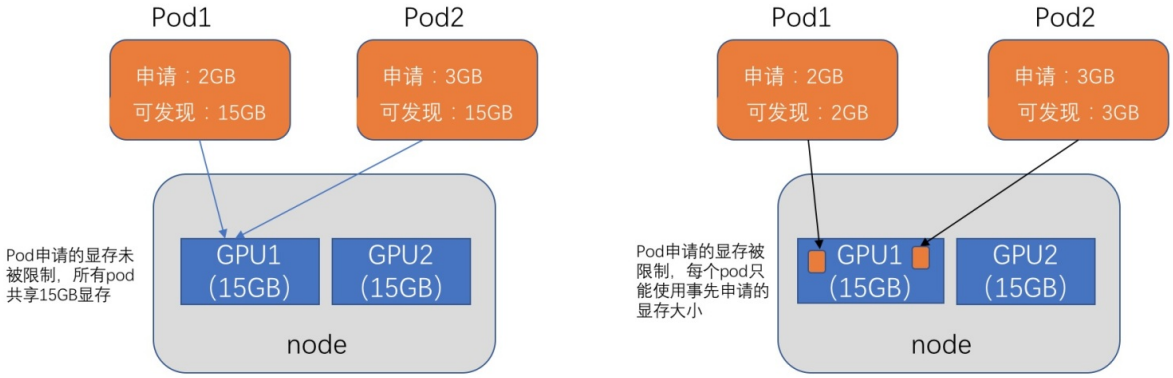
如果需要使用共享GPU或GPU拓扑感知功能，需要为GPU节点打上一些特定标签。

> **注意** 节点切换互斥标签时，必须要保证该节点上申请扩展资源的Pod已经运行完成。例如，节点的标签需要由ack.node.gpu.schedule=share切换到ack.node.gpu.schedule=topology，那么必须保证该节点上申请了aliyun.com/gpu-mem资源的Pod全部运行完成后，才能切换。

标签	扩展资源名称	说明
ack.node.gpu.schedule=share	aliyun.com/gpu-mem	在节点上开启GPU共享调度能力，不开启GPU显存隔离能力。
ack.node.gpu.schedule=cgpu	aliyun.com/gpu-mem	在节点上开启GPU共享调度能力，同时开启GPU显存隔离能力。
ack.node.gpu.schedule=topology	aliyun.com/gpu	在节点上开启GPU拓扑感知能力。
ack.node.gpu.schedule=default	nvidia.com/gpu	在节点使用默认GPU调度能力。
ack.node.gpu.placement=binpack	\	在节点上为Pod选择GPU卡时，使用Binpack算法（仅对共享GPU生效）。
ack.node.gpu.placement=spread	\	在节点上为Pod选择GPU卡时，使用Spread算法（仅对共享GPU生效）。

GPU共享调度和显存隔离

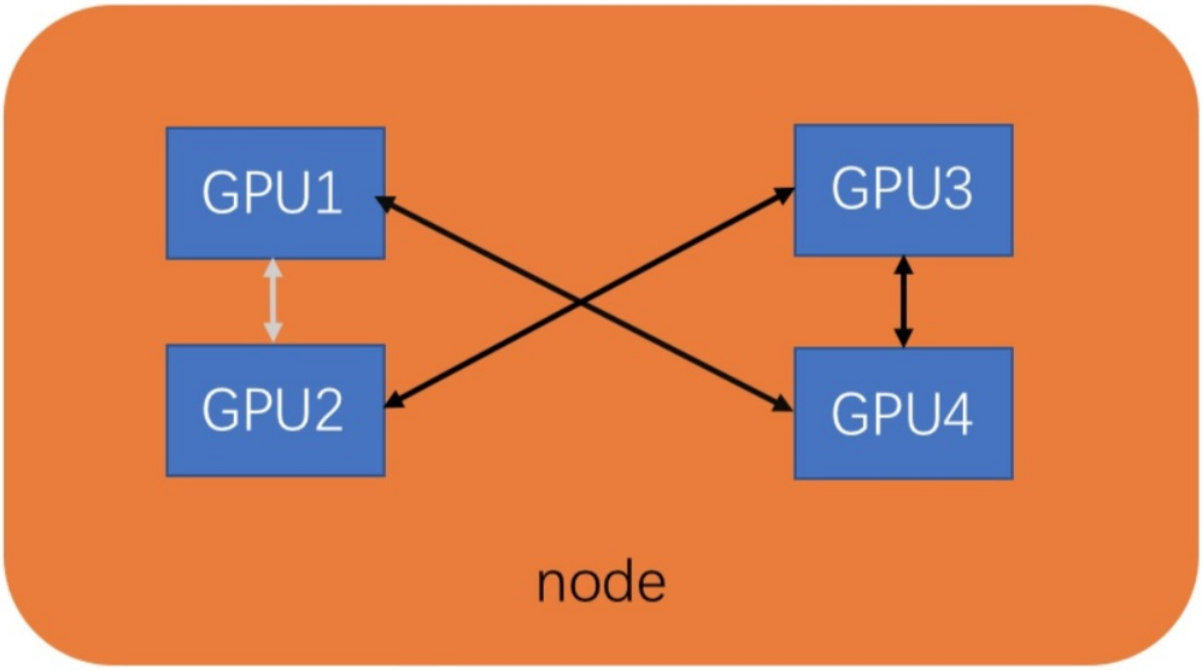
- GPU共享调度指的是多个Pod可以共享同一节点的同一张GPU卡。例如有一个节点拥有两张GPU卡（GPU1和GPU2，显存都为15 GiB），某一时刻有两个Pod都在使用GPU1，此时可以称为Pod1和Pod2共享使用GPU1。
- GPU显存隔离指的是当多个Pod共享同一张GPU卡时，每个Pod使用的显存进行了隔离限制。
  - 如下图左边部分所示，Pod1和Pod2申请的显存大小分别为2 GiB和3 GiB，但是每个Pod可使用的显存都是GPU1的显存（15 GiB），这种情况就是没有显存隔离。如果Pod没有设置显存隔离，那就存在Pod运行失败的风险。例如，某个Pod申请的显存为2 GiB，但是实际使用了15 GiB，将会导致共享GPU1的Pod都运行失败。
  - 如下图的右边部分所示，每个Pod使用的显存都是独占的，而且是被限制的。例如，某个Pod申请了2 GiB显存，那么也只能使用2 GiB显存。如果该Pod使用3 GiB显存，则程序会异常退出。



不支持显存隔离 VS 支持显存隔离

GPU拓扑感知调度

如下图所示，某一个节点有4张GPU卡，分别为GPU1、GPU2、GPU3、GPU4。GPU1和GPU2两张卡在通信时，它们的通信带宽最大，而GPU1和GPU4在通信时，它们的通信带宽最小。假设某个Pod申请了2张GPU卡，那么GPU拓扑感知调度的目标是优先选出GPU1和GPU2的组合给该Pod，因为这个组合的通信带宽最大，Pod中应用程序在两张卡之间交换数据的时延最小。

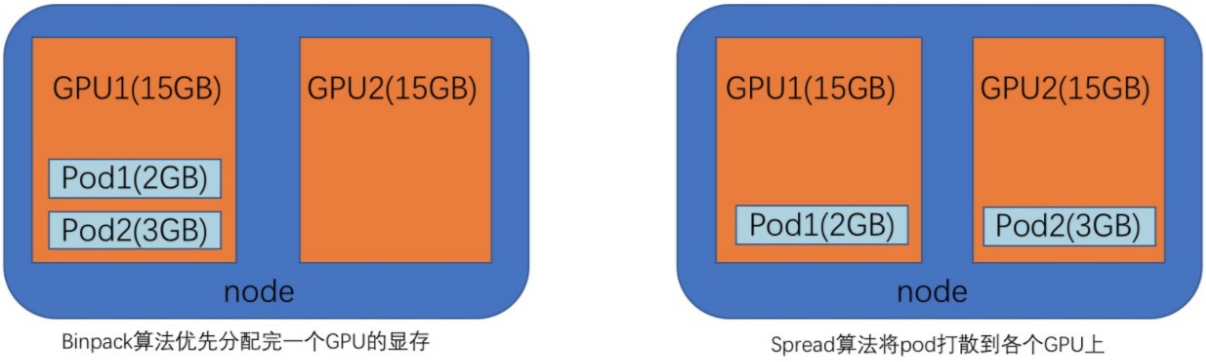


Binpack和Spread算法

GPU共享调度中，在节点上支持按照Binpack和Spread算法为Pod分配GPU显存。

如下图所示，节点上有2张GPU卡，每张卡的显存为15 GiB，Pod1申请了2 GiB的显存，Pod2申请了3 GiB显存。

- 如果您设置的是Binpack算法，系统会优先将一张GPU卡的分配完后，再分配另一张GPU卡。如下图左边部分所示，两个Pod被分配到GPU1上，优先将GPU1上的显存分配完。
- 如果您设置的是Spread算法，系统会尽量将Pod申请的显存分散到各个GPU上。如下图中右边部分所示，Pod1和Pod2被分散到了GPU1和GPU2上。



Binpack算法优先分配完一个GPU的显存　　Spread算法将pod打散到各个GPU上

Binpack VS Spread

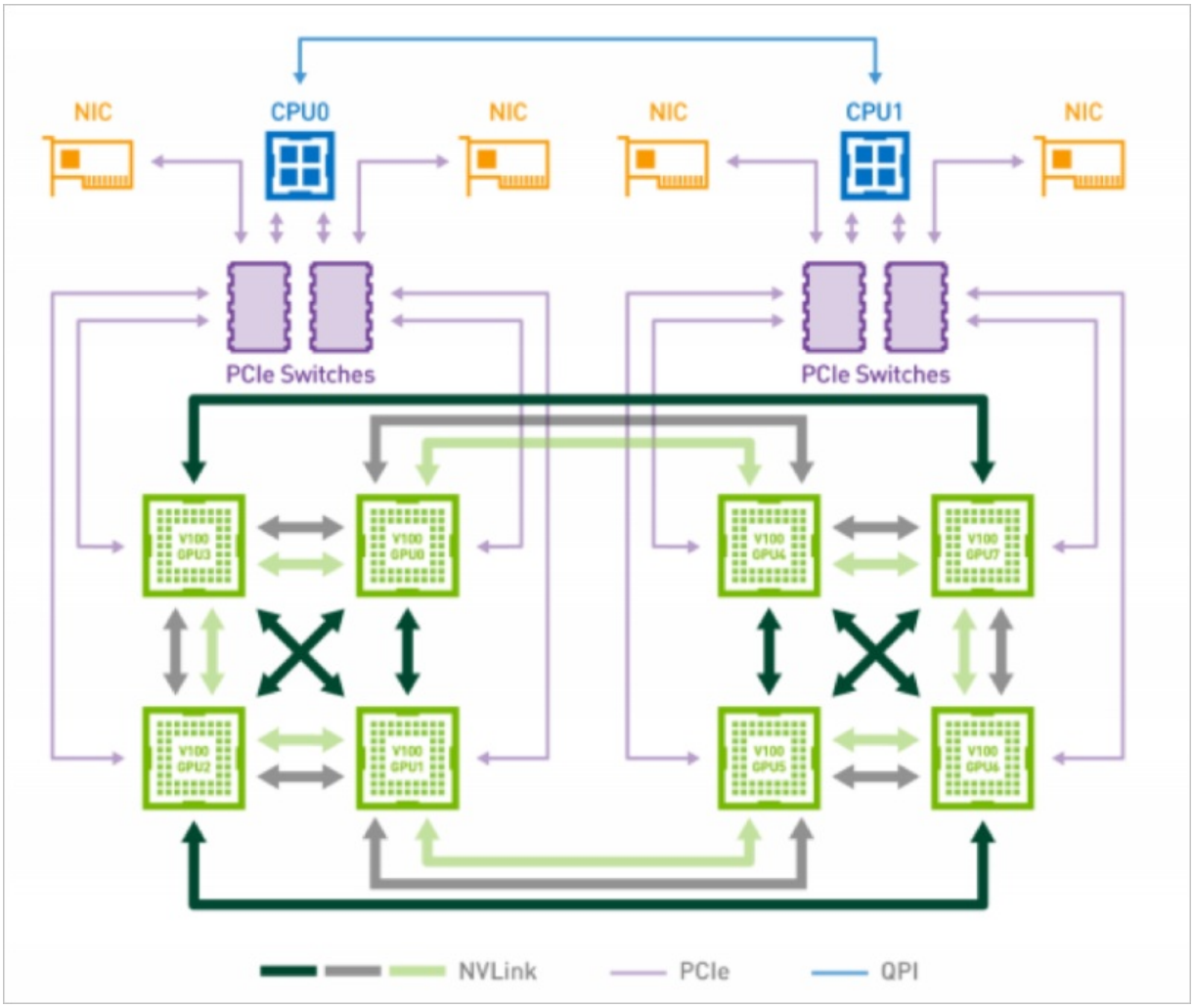
18.2.4. GPU拓扑感知调度

18.2.4.1. GPU拓扑感知调度概述

本文简要描述了GPU拓扑分布情况，以及GPU拓扑感知调度的优势。

GPU拓扑分布

下图为NVLink连接8个Tesla V100的混合立体网络拓扑。每块V100 GPU有6个NVLink通道，8块GPU间无法做到全连接，2块GPU间最多只能有2条NVLink连接。其中GPU0和GPU3，GPU0和GPU4之间有2条NVLink连接，GPU0和GPU1之间有一条NVLink连接，GPU0和6之间没有NVLink连接，故GPU0与GPU6之间仍然需要通过PCIe进行通信。



GPU拓扑感知调度优势

NVlink连接的单向通信带宽为25 GB/s，双向通信带宽为50 GB/s，PCIe连接的通信带宽为16 GB/s。在训练过程中，选择不同的GPU组合，会得到不同的训练速度，所以在GPU的调度过程中，选择最优的GPU组合，可得到最优的训练速度。

Kubernetes对节点的GPU拓扑信息不感知，调度过程中对GPU的选择比较随机，选择不同的GPU组合训练速度会存在较大的差异。基于以上问题，ACK基于Scheduling Framework机制，实现GPU拓扑感知调度，在节点的GPU组合中选择具有最优训练速度的组合。

相关文档

- 安装GPU拓扑感知组件
- Tensorflow分布式训练使用GPU拓扑感知调度
- Pytorch分布式训练使用GPU拓扑感知调度

18.2.4.2. 安装GPU拓扑感知组件

本文主要介绍在激活GPU拓扑资源调度前，您需要安装的组件和部署的配置。

前提条件

- 创建Kubernetes Pro集群。Kubernetes Pro集群的实例规格类型需要设置为**异构计算GPU**，其它操作配置，请参见创建ACK Pro版集群。

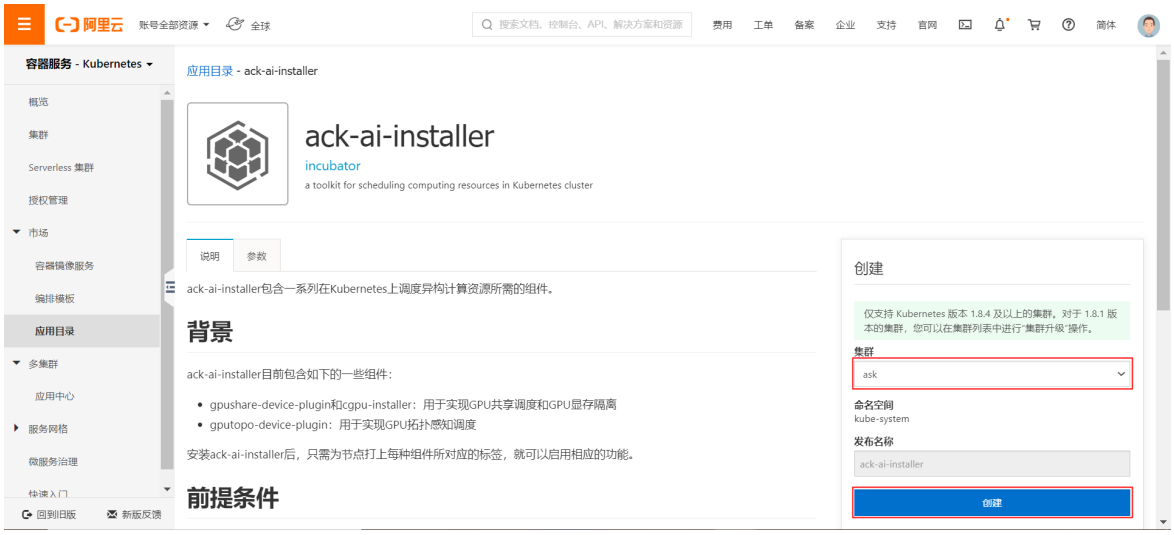
  > **注意** 目前仅支持Kubernetes Pro集群，如果是专有版集群，请提交工单申请白名单。

- 通过kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。
- 系统组件版本要求具体如下表所示。

组件	版本要求
Kubernetes	1.18.8及以上版本
Helm	3.0及以上版本
Nvidia	418.87.01及以上版本
训练框架NCCL版本	2.7+
Docker	19.03.5
操作系统	CentOS 7.6、CentOS 7.7、Ubuntu 16.04和Ubuntu 18.04、Alibaba Cloud Linux 2
显卡	V100

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的右上角**名称**文本框输入*ack-ai-installer*。
4. 在**应用目录-ack-ai-installer**页面右侧的**创建**面板中，在**集群**列表框中选择目标集群，并单击**创建**。



相关文档

- GPU拓扑感知调度概述
- Tensorflow分布式训练使用GPU拓扑感知调度
-

18.2.4.3. Tensorflow分布式训练使用GPU拓扑感知调度

ACK基于Scheduling Framework机制，实现GPU拓扑感知调度，即在节点的GPU组合中选择具有最优训练速度的组合。本文主要介绍如何使用GPU拓扑感知调度来提升Tensorflow分布式训练的训练速度。

前提条件

- 创建ACK Pro版集群。
- 安装Arena。
- 安装GPU拓扑感知组件。
- 系统组件版本要求具体如下表所示。

组件	版本要求
Kubernetes	1.18.8及以上版本
Helm	3.0及以上版本
Nvidia	418.87.01及以上版本
训练框架NCCL版本	2.7+
Docker	19.03.5
操作系统	CentOS 7.6、CentOS 7.7、Ubuntu 16.04和Ubuntu 18.04、Alibaba Cloud Linux 2
显卡	V100

应用限制

- 仅支持MPI作业的分布式训练。
- 只有当提交作业的所有Pod对资源请求都满足条件时，才能创建Pod并启动作业，否则请求会处于资源等待状态。

操作步骤

节点配置

执行以下命令设置节点Label，显式激活节点GPU拓扑感知调度。

```
kubectl label node <Your Node Name> ack.node.gpu.schedule=topology
```

> **说明** 当节点激活GPU拓扑感知调度后，不再支持普通GPU资源调度。可执行以下命令更改Label，恢复普通GPU资源调度功能。
>
> ```
> kubectl label node <Your Node Name> ack.node.gpu.schedule=default --overwrite
> ```

提交作业

提交MPI作业，并设置--gputopology为 `true` 。

```
arena submit mpi --gputopology=true ***
```

示例一：训练Vgg16

> **说明** 本文测试集群有2台8卡V100机器。

使用GPU拓扑感知调度训练Vgg16

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=tensorflow-topo-4-vgg16 \
 --gpus=1 \
 --workers=4 \
 --gputopology=true \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/tensorflow-benchmark:tf2.3.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /tensorflow/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=vgg16 --batch_size=64 --variable_update=horovod"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get tensorflow-topo-4-vgg16 --type mpijob
```

预期输出：

```
Name:    tensorflow-topo-4-vgg16
Status:  RUNNINGNamespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 2m
Instances:
 NAME                    STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                    ------  --- -------- -------------- ----
 tensorflow-topo-4-vgg16-launcher-lmhjl Running 2m  true    0        cn-shanghai.192.168.16.172
 tensorflow-topo-4-vgg16-worker-0     Running 2m  false   1        cn-shanghai.192.168.16.173
 tensorflow-topo-4-vgg16-worker-1     Running 2m  false   1        cn-shanghai.192.168.16.173
 tensorflow-topo-4-vgg16-worker-2     Running 2m  false   1        cn-shanghai.192.168.16.173
 tensorflow-topo-4-vgg16-worker-3     Running 2m  false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

```
arena logs -f tensorflow-topo-4-vgg16
```

预期输出：

```
total images/sec: 991.92
```

使用普通GPU调度训练Vgg16

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=tensorflow-4-vgg16 \
 --gpus=1 \
 --workers=4 \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/tensorflow-benchmark:tf2.3.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /tensorflow/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=vgg16 --batch_size=64 --variable_update=horovod"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get tensorflow-4-vgg16 --type mpijob
```

预期输出：

```
Name:    tensorflow-4-vgg16
Status:  RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 9s
Instances:
 NAME                STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                ------  --- -------- -------------- ----
 tensorflow-4-vgg16-launcher-xc28k Running 9s  true    0        cn-shanghai.192.168.16.172
 tensorflow-4-vgg16-worker-0     Running 9s  false   1        cn-shanghai.192.168.16.172
 tensorflow-4-vgg16-worker-1     Running 9s  false   1        cn-shanghai.192.168.16.173
 tensorflow-4-vgg16-worker-2     Running 9s  false   1        cn-shanghai.192.168.16.172
 tensorflow-4-vgg16-worker-3     Running 9s  false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

```
arena logs -f tensorflow-4-vgg16
```

预期输出：

```
total images/sec: 200.47
```

示例二：训练Resnet50

使用GPU拓扑感知调度训练Resnet50

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=tensorflow-topo-4-resnet50 \
 --gpus=1 \
 --workers=4 \
 --gputopology=true \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/tensorflow-benchmark:tf2.3.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /tensorflow/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=resnet50 --batch_size=64  --variable_update=horovod"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get tensorflow-topo-4-resnet50 --type mpijob
```

预期输出：

```
Name:    tensorflow-topo-4-resnet50
Status:  RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 8s
Instances:
 NAME                      STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                      ------  --- -------- -------------- ----
 tensorflow-topo-4-resnet50-launcher-7ln8j Running 8s  true    0        cn-shanghai.192.168.16.172
 tensorflow-topo-4-resnet50-worker-0     Running 8s  false   1        cn-shanghai.192.168.16.173
 tensorflow-topo-4-resnet50-worker-1     Running 8s  false   1        cn-shanghai.192.168.16.173
 tensorflow-topo-4-resnet50-worker-2     Running 8s  false   1        cn-shanghai.192.168.16.173
 tensorflow-topo-4-resnet50-worker-3     Running 8s  false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

```
arena logs -f tensorflow-topo-4-resnet50
```

预期输出：

```
total images/sec: 1471.55
```

使用普通GPU调度训练Resnet50

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=tensorflow-4-resnet50 \
 --gpus=1 \
 --workers=4 \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/tensorflow-benchmark:tf2.3.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /tensorflow/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=resnet50 --batch_size=64  --variable_update=horovod"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get tensorflow-4-resnet50 --type mpijob
```

预期输出：

```
Name:    tensorflow-4-resnet50
Status:  RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 9s
Instances:
 NAME                   STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                   ------  --- -------- -------------- ----
 tensorflow-4-resnet50-launcher-q24hv Running 9s  true    0        cn-shanghai.192.168.16.172
 tensorflow-4-resnet50-worker-0     Running 9s  false   1        cn-shanghai.192.168.16.172
 tensorflow-4-resnet50-worker-1     Running 9s  false   1        cn-shanghai.192.168.16.173
 tensorflow-4-resnet50-worker-2     Running 9s  false   1        cn-shanghai.192.168.16.172
 tensorflow-4-resnet50-worker-3     Running 9s  false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

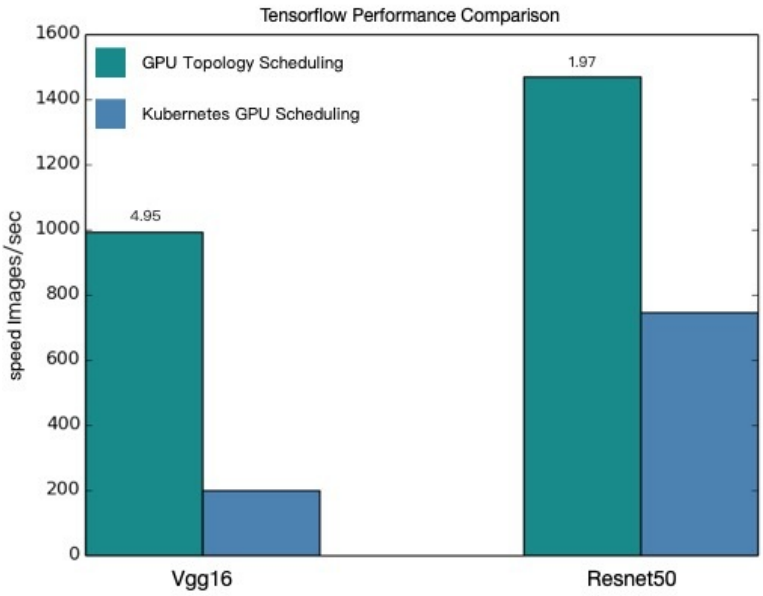
```
arena logs -f tensorflow-4-resnet50
```

预期输出：

```
total images/sec: 745.38
```

性能对比

基于以上4个测试用例性能对比结果如下：



基于对比图，可知经过GPU拓扑感知调度后，Tensorflow分布式训练的效果有了很大的提升。

> **说明** GPU拓扑感知调度提升结果与您使用的模型以及集群的环境有一定关系，您可以参考上述使用示例，评测自己的模型。

相关文档

- GPU拓扑感知调度概述
- 安装GPU拓扑感知组件

18.2.4.4. Pytorch分布式训练使用GPU拓扑感知调度

ACK基于Scheduling Framework机制，实现GPU拓扑感知调度，即在节点的GPU组合中选择具有最优训练速度的组合。本文主要介绍如何使用GPU拓扑感知调度来提升Pytorch分布式训练的训练速度。

前提条件

- 创建ACK Pro版集群。
- 安装Arena。
- 安装GPU拓扑感知组件。
- 系统组件版本要求具体如下表所示。

组件	版本要求
Kubernetes	1.18.8及以上版本
Helm	3.0及以上版本
Nvidia	418.87.01及以上版本
训练框架NCCL版本	2.7+
Docker	19.03.5
操作系统	CentOS 7.6、CentOS 7.7、Ubuntu 16.04和Ubuntu 18.04、Alibaba Cloud Linux 2
显卡	V100

应用限制

- 仅支持MPI作业的分布式训练。
- 只有当提交作业的所有Pod对资源请求都满足条件时，才能创建Pod并启动作业，否则请求会处于资源等待状态。

操作步骤

节点配置

您需执行以下命令设置节点Label，显式激活节点GPU拓扑感知调度。

```
kubectl label node <Your Node Name> ack.node.gpu.schedule=topology
```

> **说明** 当节点激活GPU拓扑感知调度后，不再支持普通GPU资源调度。可执行以下命令更改Label，恢复普通GPU资源调度功能。
>
> ```
> kubectl label node <Your Node Name> ack.node.gpu.schedule=default --overwrite
> ```

提交作业

您在提交MPI作业时，执行以下命令设置--gputopology为 `true` 。

```
arena submit mpi --gputopology=true ***
```

示例一：训练Vgg16

> **说明** 本文测试集群有2台8卡V100机器。

使用GPU拓扑感知调度训练Vgg16

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=pytorch-topo-4-vgg16 \
 --gpus=1 \
 --workers=4 \
 --gputopology=true \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/pytorch-benchmark:torch1.6.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /examples/pytorch_synthetic_benchmark.py --model=vgg16 --batch-size=64"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get pytorch-topo-4-vgg16 --type mpijob
```

预期输出：

```
Name:    pytorch-topo-4-vgg16
Status:  RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 11s
Instances:
 NAME                  STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                  ------  --- -------- -------------- ----
 pytorch-topo-4-vgg16-launcher-mnjzr Running 11s true    0        cn-shanghai.192.168.16.173
 pytorch-topo-4-vgg16-worker-0     Running 11s false   1        cn-shanghai.192.168.16.173
 pytorch-topo-4-vgg16-worker-1     Running 11s false   1        cn-shanghai.192.168.16.173
 pytorch-topo-4-vgg16-worker-2     Running 11s false   1        cn-shanghai.192.168.16.173
 pytorch-topo-4-vgg16-worker-3     Running 11s false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

```
arena logs -f pytorch-topo-4-vgg16
```

预期输出：

```
Model: vgg16
Batch size: 64
Number of GPUs: 4
Running warmup...
Running benchmark...
Iter #0: 205.5 img/sec per GPU
Iter #1: 205.2 img/sec per GPU
Iter #2: 205.1 img/sec per GPU
Iter #3: 205.5 img/sec per GPU
Iter #4: 205.1 img/sec per GPU
Iter #5: 205.1 img/sec per GPU
Iter #6: 205.3 img/sec per GPU
Iter #7: 204.3 img/sec per GPU
Iter #8: 205.0 img/sec per GPU
Iter #9: 204.9 img/sec per GPU
Img/sec per GPU: 205.1 +-0.6
Total img/sec on 4 GPU(s): 820.5 +-2.5
```

使用普通GPU调度训练Vgg16

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=pytorch-4-vgg16 \
 --gpus=1 \
 --workers=4 \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/pytorch-benchmark:torch1.6.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /examples/pytorch_synthetic_benchmark.py --model=vgg16 --batch-size=64"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get pytorch-4-vgg16 --type mpijob
```

预期输出：

```
Name:    pytorch-4-vgg16
Status:   RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 10s
Instances:
 NAME                STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                ------  --- -------- -------------- ----
 pytorch-4-vgg16-launcher-qhnxl Running 10s true    0        cn-shanghai.192.168.16.173
 pytorch-4-vgg16-worker-0     Running 10s false   1        cn-shanghai.192.168.16.173
 pytorch-4-vgg16-worker-1     Running 10s false   1        cn-shanghai.192.168.16.173
 pytorch-4-vgg16-worker-2     Running 10s false   1        cn-shanghai.192.168.16.173
 pytorch-4-vgg16-worker-3     Running 10s false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

```
arena logs -f pytorch-4-vgg16
```

预期输出：

```
Model: vgg16
Batch size: 64
Number of GPUs: 4
Running warmup...
Running benchmark...
Iter #0: 113.1 img/sec per GPU
Iter #1: 109.5 img/sec per GPU
Iter #2: 106.5 img/sec per GPU
Iter #3: 108.5 img/sec per GPU
Iter #4: 108.1 img/sec per GPU
Iter #5: 111.2 img/sec per GPU
Iter #6: 110.7 img/sec per GPU
Iter #7: 109.8 img/sec per GPU
Iter #8: 102.8 img/sec per GPU
Iter #9: 107.9 img/sec per GPU
Img/sec per GPU: 108.8 +-5.3
Total img/sec on 4 GPU(s): 435.2 +-21.1
```

示例二：训练Resnet50

使用GPU拓扑感知调度训练Resnet50

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=pytorch-topo-4-resnet50 \
 --gpus=1 \
 --workers=4 \
 --gputopology=true \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/pytorch-benchmark:torch1.6.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /examples/pytorch_synthetic_benchmark.py --model=resnet50 --batch-size=64"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get pytorch-topo-4-resnet50 --type mpijob
```

预期输出：

```
Name:    pytorch-topo-4-resnet50
Status:  RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 8s
Instances:
 NAME                    STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                    ------  --- -------- -------------- ----
 pytorch-topo-4-resnet50-launcher-x7r2n Running 8s  true    0        cn-shanghai.192.168.16.173
 pytorch-topo-4-resnet50-worker-0     Running 8s  false   1        cn-shanghai.192.168.16.173
 pytorch-topo-4-resnet50-worker-1     Running 8s  false   1        cn-shanghai.192.168.16.173
 pytorch-topo-4-resnet50-worker-2     Running 8s  false   1        cn-shanghai.192.168.16.173
 pytorch-topo-4-resnet50-worker-3     Running 8s  false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

```
arena logs -f pytorch-topo-4-resnet50
```

预期输出：

```
Model: resnet50
Batch size: 64
Number of GPUs: 4
Running warmup...
Running benchmark...
Iter #0: 331.0 img/sec per GPU
Iter #1: 330.6 img/sec per GPU
Iter #2: 330.9 img/sec per GPU
Iter #3: 330.4 img/sec per GPU
Iter #4: 330.7 img/sec per GPU
Iter #5: 330.8 img/sec per GPU
Iter #6: 329.9 img/sec per GPU
Iter #7: 330.5 img/sec per GPU
Iter #8: 330.4 img/sec per GPU
Iter #9: 329.7 img/sec per GPU
Img/sec per GPU: 330.5 +-0.8
Total img/sec on 4 GPU(s): 1321.9 +-3.2
```

使用普通GPU调度训练Resnet50

1. 使用以下命令向集群提交作业。

```
arena submit mpi \
 --name=pytorch-4-resnet50 \
 --gpus=1 \
 --workers=4 \
 --image=registry.cn-hangzhou.aliyuncs.com/kubernetes-image-hub/pytorch-benchmark:torch1.6.0-py3.7-cuda10.1 \
 "mpirun --allow-run-as-root -np "4" -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH --mca pml ob1 --mca btl_tcp_if_include eth0 --mca oob_tcp_if_include eth0 --mca orte_keep_fqdn_hostnames t --mca btl ^openib python /examples/pytorch_synthetic_benchmark.py --model=resnet50 --batch-size=64"
```

2. 执行以下命令，查看当前作业运行情况。

```
arena get pytorch-4-resnet50 --type mpijob
```

预期输出：

```
Name:    pytorch-4-resnet50
Status:  RUNNING
Namespace: default
Priority: N/A
Trainer:  MPIJOB
Duration: 10s
Instances:
 NAME                 STATUS  AGE IS_CHIEF GPU(Requested) NODE
 ----                 ------  --- -------- -------------- ----
 pytorch-4-resnet50-launcher-qw5k6 Running 10s true    0        cn-shanghai.192.168.16.173
 pytorch-4-resnet50-worker-0     Running 10s false   1        cn-shanghai.192.168.16.173
 pytorch-4-resnet50-worker-1     Running 10s false   1        cn-shanghai.192.168.16.173
 pytorch-4-resnet50-worker-2     Running 10s false   1        cn-shanghai.192.168.16.173
 pytorch-4-resnet50-worker-3     Running 10s false   1        cn-shanghai.192.168.16.173
```

3. 执行以下命令，查看当前日志信息。

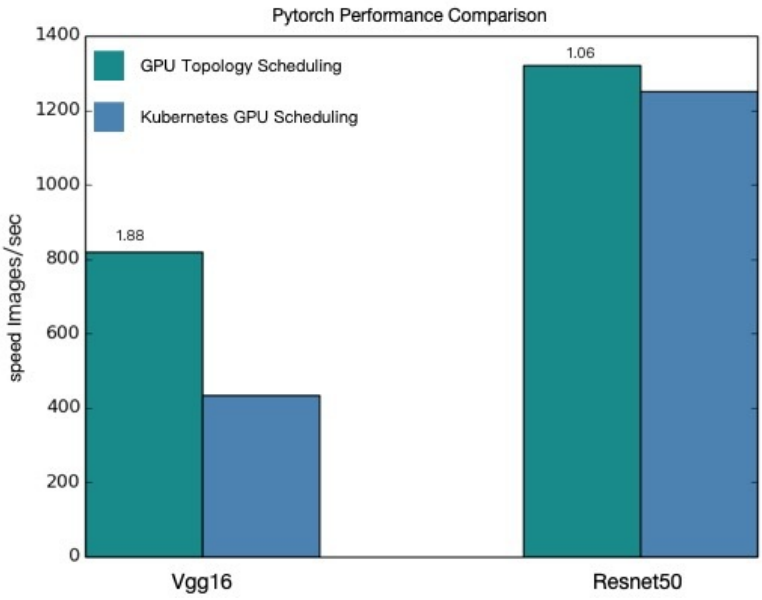
```
arena logs -f pytorch-4-resnet50
```

预期输出：

```
Model: resnet50
Batch size: 64
Number of GPUs: 4
Running warmup...
Running benchmark...
Iter #0: 313.1 img/sec per GPU
Iter #1: 312.8 img/sec per GPU
Iter #2: 313.0 img/sec per GPU
Iter #3: 312.2 img/sec per GPU
Iter #4: 313.7 img/sec per GPU
Iter #5: 313.2 img/sec per GPU
Iter #6: 313.6 img/sec per GPU
Iter #7: 313.0 img/sec per GPU
Iter #8: 311.3 img/sec per GPU
Iter #9: 313.6 img/sec per GPU
Img/sec per GPU: 313.0 +-1.3
Total img/sec on 4 GPU(s): 1251.8 +-5.3
```

性能对比

基于如上4个测试用例性能对比结果如下：



基于上图性能对比，可知经过GPU拓扑感知调度后，Pytorch分布式训练的效果有了很大的提升。

> **说明** GPU拓扑感知调度提升结果与您使用的模型以及集群的环境有一定关系，您可以参考上述使用示例，评测自己的模型。

相关文档

- GPU拓扑感知调度概述
- 安装GPU拓扑感知组件

18.3. 共享GPU调度

18.3.1. 共享GPU调度概述

本文为您介绍阿里云共享GPU方案、共享GPU专业版的优势、共享GPU的基础版与专业版的功能对比及使用场景，帮助您了解和更好地使用共享GPU的能力。

背景介绍

阿里云容器服务Kubernetes版ACK（Container Service for Kubernetes）开源了GPU共享调度之后，您能在阿里云、AWS、GCE和自己数据中心的容器集群上通过GPU共享调度框架实现多个容器运行在同一个GPU设备上的目标。ACK开源GPU共享调度降低了使用GPU的经济成本，但是如何能在节省经济成本同时也能让GPU上的容器运行更稳定呢？

隔离是一个关键的需求。如何限制运行在同一个GPU上的多个容器能够按照自己申请的资源使用量运行，避免因为其资源用量超标影响同一个GPU上的其他容器的正常工作，对此业界也做了很多探索。NVIDIA vGPU、MPS，和vCUDA方案，都为更小颗粒度的使用GPU提供了可能。

基于以上的需求，阿里云容器服务团队提供共享GPU方案，既能够实现一个GPU供多个任务使用，同时也能够实现一个GPU上对各应用申请的显存实现隔离以及GPU的算力分割。

功能介绍

阿里云提供的共享GPU方案通过自主研发的宿主机内核驱动， 实现对NVIDIA GPU的底层nv驱动更有效的利用。共享GPU功能如下：

- 更加开放：适配开源标准的Kubernetes和NVIDIA Docker方案。
- 更加简单：优秀的用户体验。AI应用无需重编译，无需构建新的容器镜像进行CUDA库替换。
- 更加稳定：针对NVIDIA设备的底层操作更加稳定和收敛，而CUDA层的API变化多端，同时一些Cudnn非开放的API也不容易捕获。
- 完整隔离：同时支持GPU的显存和算力隔离。

阿里云提供的共享GPU方案是一套低成本、可靠、用户友好的规模化GPU调度和隔离方案，欢迎使用。

共享GPU专业版优势

优势	说明
支持共享调度和显存隔离。	• 单Pod单GPU卡共享调度和显存隔离，常用于支持模型推理场景。 • 单Pod多GPU卡共享调度和显存隔离，常用于支持分布式模型训练代码的开发。

支持共享和隔离策略的灵活配置。	• 支持按GPU卡的Binpack和Spread算法分配策略。
	  ◦ Binpack：多个Pod会优先集中共享使用同一GPU卡，适用于需要提升GPU卡利用率的场景。
	  ◦ Spread：多个Pod会尽量分散使用不同GPU卡，适用于GPU高可用场景。尽量避免将同一个应用的副本放置到同一个GPU设备。
	• 支持只共享不隔离策略，适配于已有深度学习应用内已自建应用层隔离能力的场景。
	• 同时支持多卡共享和显存隔离策略。

GPU资源全方位监控。	同时支持监控独占GPU和共享GPU。

共享GPU基础版和专业版对比

功能	专业版	基础版
GPU单卡共享调度。	支持	支持
GPU多卡共享调度。	支持	不支持
GPU单卡隔离。	支持	支持
GPU多卡隔离。	支持	不支持
独享和共享GPU的监控和弹性伸缩。	支持	支持
多策略节点池。	支持，包含只共享不隔离和共享隔离策略。	支持，只共享不隔离和共享隔离策略，协同Binpack与Spread算法。
根据算法为Pod分配GPU显存。	支持，按Binpack和Spread算法分配节点的GPU，您可以根据实际需求选择分配算法。	支持，默认按Binpack算法分配节点的GPU。

使用说明

在不同集群上安装共享GPU组件，组件能力不同，安装的操作步骤也不同：

- 专有版GPU集群：安装共享GPU基础版。更多信息，请参见：
  - 安装共享GPU组件
  - 运行共享GPU示例
  - 监控和隔离GPU资源
  - 关闭共享GPU显存隔离能力
  - 基于节点池管理共享GPU
- ACK Pro版集群：安装共享GPU专业版。更多信息，请参见：
  - 安装并使用共享GPU组件和资源工具
  - 运行共享GPU示例
  - 通过共享GPU实现多卡共享策略
  - 基于节点池管理共享GPU
  - 若ACK专有版集群已安装共享GPU基础版，在迁移至ACK Pro托管版集群后，需要将共享GPU基础版升级为共享GPU专业版。具体操作，请参见在ACK Pro版集群中将共享GPU基础版升级为共享GPU专业版。

关于ACK专有版集群与ACK Pro版集群的差异，请参见集群类型。

18.3.2. 基础版

18.3.2.1. 安装共享GPU组件

目前阿里云ACK提供了GPU共享调度能力，服务于使用共享单GPU的模型预测场景，同时也可以通过NVIDIA驱动在内核态的控制进而保障GPU显存的隔离。本文介绍如何在专有版GPU集群中安装GPU隔离模块及安装GPU分配查询工具，实现GPU的共享和隔离能力。

适用场景

- 本文所描述的组件，仅支持专有版GPU集群，不支持托管版GPU集群。
- 若您想在ACK Pro版集群安装共享GPU组件，请参见安装并使用共享GPU组件和资源工具。

前提条件

- 支持共享GPU调度的节点不能设置CPU Policy为 `static` 。
- 已创建专有版GPU集群。具体操作，请参见创建专有GPU集群。
- 已通过kubectl工具连接专有版GPU集群。具体操作，请参见通过kubectl工具连接集群。

使用须知

配置	支持版本
Kubernetes	1.12.6及其以上，仅支持专有版集群
Helm版本	3.0及以上版本
NVIDIA驱动版本	418.87.01及以上版本
Docker版本	19.03.5
操作系统	CentOS 7.x、Alibaba Cloud Linux 2.x、Ubuntu 16.04和Ubuntu 18.04
支持显卡	Tesla P4、Tesla P100、 Tesla T4和Tesla v100

步骤一：为带有GPU设备的节点打标签

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点管理**页面单击右上角的**标签与污点管理**。
6. 在**标签与污点管理**页面，批量选择节点，然后单击**添加标签**。
7. 在弹出的**添加**对话框中，填写标签**名称**和**值**。

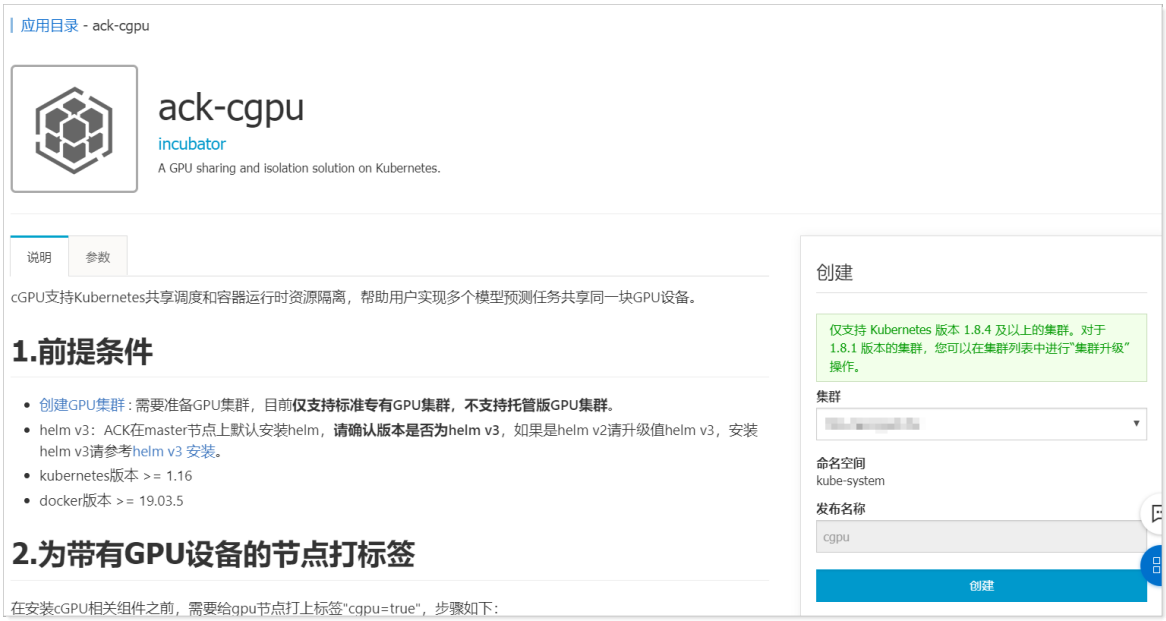
   > **注意**
   > - 您可以设置**名称**为**cgpu**，**值**为**true**，使节点共享GPU功能生效。
   > - 删除**cgpu**标签并不能关闭共享，您可以设置**名称**为**cgpu**，**值**为**false**，关闭节点共享GPU功能。

8. 单击**确定**。

步骤二：为添加标签的节点安装共享GPU组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面，搜索ack-cgpu，然后单击ack-cgpu。
4. 在**应用目录 - ack-cgpu**页面右侧的**创建**面板中，选中目标集群，然后单击**创建**。

   您无需设置**命名空间**和**发布名称**，系统显示默认值。



您可以执行命令`helm get manifest cgpu -n kube-system | kubectl get -f -`查看cGPU组件是否安装成功。当出现以下命令详情时，说明cGPU组件安装成功。

```
helm get manifest cgpu -n kube-system | kubectl get -f -
```

```
NAME                    SECRETS  AGE
serviceaccount/gpushare-device-plugin  1     39s
serviceaccount/gpushare-schd-extender  1     39s
NAME                                   AGE
clusterrole.rbac.authorization.k8s.io/gpushare-device-plugin  39s
clusterrole.rbac.authorization.k8s.io/gpushare-schd-extender  39s
NAME                                    AGE
clusterrolebinding.rbac.authorization.k8s.io/gpushare-device-plugin  39s
clusterrolebinding.rbac.authorization.k8s.io/gpushare-schd-extender  39s
NAME                 TYPE    CLUSTER-IP   EXTERNAL-IP  PORT(S)      AGE
service/gpushare-schd-extender  NodePort  10.6.13.125  <none>     12345:32766/TCP  39s
NAME                      DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR   AGE
daemonset.apps/cgpu-installer        4     4     4    4       4      cgpu=true     39s
daemonset.apps/device-plugin-evict-ds   4     4     4    4       4      cgpu=true     39s
daemonset.apps/device-plugin-recover-ds  0     0     0    0       0      cgpu=false  39s
daemonset.apps/gpushare-device-plugin-ds  4     4     4    4       4      cgpu=true     39s
NAME                    READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/gpushare-schd-extender  1/1    1       1      38s
NAME                COMPLETIONS  DURATION  AGE
job.batch/gpushare-installer  3/1 of 3    3s      38s
```

相关文档

- 共享GPU调度概述
- 运行共享GPU示例
- 监控和隔离GPU资源

18.3.2.2. 运行共享GPU示例

在安装完共享GPU组件后，本文将介绍如何在专有版GPU集群中使用共享GPU能力。

前提条件

已安装共享GPU组件。具体操作，请参见安装共享GPU组件。

适用场景

- 本文所描述的组件，仅支持专有版GPU集群，不支持托管版GPU集群。
- 如果您想在ACK Pro版集群运行共享GPU示例，请参见运行共享GPU示例。

操作步骤

1. 登录到Master节点，执行以下命令查询集群的GPU共享能力。

   > **说明**
   > - 关于如何登录到Master节点，请参见通过密码认证登录Linux实例或通过密码认证登录Windows实例。
   > - 如果您需要在本地客户端执行命令查询集群的GPU共享能力，请安装共享GPU组件和资源工具。具体操作，请参见步骤四：安装和使用GPU资源查询工具。

   ```
   kubectl inspect cgpu
   ```

   预期输出：

   ```
   NAME             IPADDRESS    GPU0(Allocated/Total)  GPU1(Allocated/Total)  GPU Memory(GiB)
   cn-beijing.192.168.0.4  192.168.0.4  0/7                    0/7                    0/14
   ------------------------------------------------------------------------------
   Allocated/Total GPU Memory In Cluster:
   0/14 (0%)
   ```

   > **说明** 如果想查询GPU共享能力详细信息，执行命令kubectl inspect cgpu -d。

2. 部署共享GPU示例应用，该应用申请3 GiB显存。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: binpack
 labels:
  app: binpack
spec:
 replicas: 1
 serviceName: "binpack-1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: binpack-1
 template: # define the pods specifications
  metadata:
   labels:
    app: binpack-1
  spec:
   containers:
   - name: binpack-1
    image: registry.cn-beijing.aliyuncs.com/ai-samples/gpushare-sample:tensorflow-1.5
    command:
     - python
     - cgpu/main.py
    resources:
     limits:
      # GiB
      aliyun.com/gpu-mem: 3
```

> **说明** aliyun.com/gpu-mem：设置GPU显存大小。

3. 执行以下命令，查看集群GPU显存使用情况。

```
kubectl inspect cgpu
```

预期输出：

```
NAME                IPADDRESS     GPU0(Allocated/Total)  GPU Memory(GiB)
cn-beijing.192.168.1.105  192.168.1.105  3/14                   3/14
--------------------------------------------------------------------------
Allocated/Total GPU Memory In Cluster:
3/14 (21%)
```

可以看到节点cn-beijing.192.168.1.105总共有14 GiB显存，已分配3 GiB显存。

执行结果

您可以通过以下方式验证节点GPU显存隔离能力是否生效。

- 执行以下命令查看步骤2中部署应用的日志。
  通过查看日志，您可以确定cGPU显存隔离是否成功部署。

```
kubectl logs binpack-0 --tail=1
```

预期输出：

```
2020-03-13 09:14:13.931003: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1326] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 2832 MB memory) -> physical GPU (device: 0, name: Tesla T4, pci bus id: 0000:00:07.0, compute capability: 7.5)
```

可以看到容器申请的显存为2832 MiB。

- 执行以下命令登录容器查看容器被分配显存总量。

```
kubectl exec -it binpack-0 nvidia-smi
```

预期输出：

```
Fri Mar 13 09:32:18 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 T4            On   | 00000000:00:07.0 Off |                    0 |
| N/A   41C    P0    26W /  70W |   3043MiB /  3231MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

可以看到该容器被分配显存总量为3231 MiB 。

- 执行以下命令查看示例应用所在节点的GPU显存总量（在节点上执行）。

```
nvidia-smi
```

预期输出：

```
Fri Mar 13 17:36:24 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 T4            On   | 00000000:00:07.0 Off |                    0 |
| N/A   40C    P0    26W /  70W |   3053MiB / 15079MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      8796      C   python3                                 3043MiB |
+-----------------------------------------------------------------------------+
```

可以看到主机上的显存总量为15079 MiB，其中分配给容器的是3053 MiB。

相关文档

- 共享GPU调度概述
- 安装共享GPU组件

- 监控和隔离GPU资源

18.3.2.3. 监控和隔离GPU资源

阿里云容器服务Kubernetes版（ACK）通过托管的Prometheus，可以提升GPU资源管理的可见性。通过共享GPU方案，能够实现多个应用调度到同一张GPU卡上，并对卡上的每个应用实现显存隔离与算力分割。本文以实际示例介绍如何通过托管的Prometheus查看集群的GPU显存使用，以及验证共享GPU方案。

适用场景

本文示例适用于已开通GPU共享调度功能的ACK专有版集群及ACK Pro版集群。

前提条件

- 已创建GPU专有版集群，且Kubernetes版本不低于1.16。具体操作，请参见创建专有GPU集群。
- 已开通ARMS。具体操作，请参见开通和升级ARMS。
- 已开启阿里云Prometheus监控。具体操作，请参见开启阿里云Prometheus监控。
- GPU硬件为Tesla P4、Tesla P100、 Tesla T4或Tesla v100（16 GB）。

背景信息

推动人工智能不断向前的动力来自于强大的算力、海量的数据和优化的算法，而NVIDIA GPU是最流行的异构算力提供者，是高性能深度学习的基石。GPU的价格不菲，从使用率的角度来看，模型预测场景下，应用独占GPU模式会造成计算资源的浪费。共享GPU模式可以提升资源利用率，但需要考虑如何达到成本和QPS平衡的最优，以及如何保障应用的SLA。

通过托管Prometheus监控独享GPU

1. 登录ARMS控制台。
2. 在左侧导航栏中，单击**Prometheus监控**。
3. 在**Prometheus监控**页面中，选择集群所在地域，然后单击目标集群**操作**列的**安装**。
4. 在**确认**对话框中，单击**确认**。

   插件安装过程需要2分钟左右。安装插件完毕后，**已安装大盘**列中将显示全部已安装的插件。
5. 通过命令行部署以下示例应用，详情请参见通过命令管理应用。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: app-3g-v1
 labels:
  app: app-3g-v1
spec:
 replicas: 1
 serviceName: "app-3g-v1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: app-3g-v1
 updateStrategy:
  type: RollingUpdate
 template: # define the pods specifications
  metadata:
   labels:
    app: app-3g-v1
  spec:
   containers:
   - name: app-3g-v1
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/cuda-malloc:3G
    resources:
     limits:
      nvidia.com/gpu: 1
```

部署成功后，执行以下命令，查看应用的状态，可以得出应用的名称是app-3g-v1-0。

```
kubectl get pod
```

预期输出：

```
NAME     READY  STATUS   RESTARTS  AGE
app-3g-v1-0  1/1   Running  1     2m56s
```

6. 单击目标集群，在目标集群的**大盘列表**页面，单击**名称**列的**GPU APP**。
可以看到该应用的GPU显存占用率仅为20%，存在80%的浪费。而它使用的显存稳定在3.4 GB左右，而总显存为16 GB左右。因此一个应用独占一张GPU卡的模式比较浪费，可以考虑通过使用cGPU（container GPU）将多个应用部署在同一个GPU卡上。



实现多容器共享同一张GPU卡

1. 为带有GPU设备的节点打标签。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。

iv. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。

v. 在**节点管理**页面，单击右上角**标签与污点管理**。

vi. 在**标签与污点管理**页面中，批量选择Worker节点，然后单击**添加标签**。

vii. 在**添加**对话框中，填写指定的标签名称和值（标签的名称为*cgpu*，值为*true*），单击**确定**。

> **注意** 如果某个Worker节点设置了标签为*cgpu=true*，那么该节点将不再拥有独享GPU资源nvidia.com/gpu；如果该节点需要关闭GPU共享功能，请设置标签cgpu的值为*false*，同时该节点将重新拥有独享GPU资源nvidia.com/gpu。

2. 安装cGPU相关组件。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，选择**市场 > 应用目录**。

iii. 在**应用目录**页面搜索ack-cgpu，然后单击ack-cgpu。

iv. 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**。

v. 登录Master节点并执行以下命令查看GPU资源。

登录Master节点相关步骤，请参见通过kubectl工具连接集群。

```
kubectl inspect cgpu
```

预期输出：

```
NAME                  IPADDRESS     GPU0(Allocated/Total)  GPU Memory(GiB)
cn-hangzhou.192.168.2.167  192.168.2.167  0/15                   0/15
------------------------------------------------------------------------------
Allocated/Total GPU Memory In Cluster:
0/15 (0%)
```

> **说明** 此时可以发现该节点的GPU资源维度已经从GPU显卡变成GPU显存。

3. 部署共享GPU的工作负载。

i. 修改之前部署应用的*YAML*文件。

- 将实例的副本数从1改为2，这样可以指定部署两个负载。而在原有GPU独享的配置下，单个GPU卡只能调度单个容器；修改配置后，可以部署两个Pod实例。
- 将资源维度从 `nvidia.com/gpu` 变为 `aliyun.com/gpu-mem` ，单位也从个变成了GB。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: app-3g-v1
 labels:
  app: app-3g-v1
spec:
 replicas: 2
 serviceName: "app-3g-v1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: app-3g-v1
 template: # define the pods specifications
  metadata:
   labels:
    app: app-3g-v1
  spec:
   containers:
   - name: app-3g-v1
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/cuda-malloc:3G
    resources:
     limits:
      aliyun.com/gpu-mem: 4  #每个Pod申请4 GB显存，因为replicas值为2，所以该应用总共申请8 GB显存
```

ii. 按照GPU显存维度调度重新创建工作负载。
从运行结果看，两个Pod都运行在同一个GPU设备上。

```
kubectl inspect cgpu -d
```

预期输出：

```
NAME:     cn-hangzhou.192.168.2.167
IPADDRESS: 192.168.2.167
NAME      NAMESPACE  GPU0(Allocated)
app-3g-v1-0 default   4
app-3g-v1-1 default   4
Allocated : 8 (53%)
Total :    15
---------------------------------------------------------
Allocated/Total GPU Memory In Cluster:  8/15 (53%)
```

iii. 执行以下命令，分别登录到两个容器。
可以看到各自GPU显存的上限已经设置为4301 MiB， 也就是在容器内使用的GPU显存不会超过此上限。

- 执行以下命令登录app-3g-v1-0容器。

```
kubectl exec -it app-3g-v1-0 nvidia-smi
```

预期输出：

```
Mon Apr 13 01:33:10 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   37C    P0    57W / 300W |   3193MiB /  4301MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

- 执行以下命令登录app-3g-v1-1容器。

```
kubectl exec -it app-3g-v1-1 nvidia-smi
```

预期输出：

```
Mon Apr 13 01:36:07 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   38C    P0    57W / 300W |   3193MiB /  4301MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

iv. 登录到节点，查看GPU的使用情况。

可以看到该GPU被使用的显存资源为两个容器之和，即6396 MiB，因此cGPU资源已经实现了按容器隔离的效果。如果此时登录到容器内尝试申请更多的GPU资源，会直接报出显存分配失败的错误。

a. 执行以下命令，登录到节点。

```
kubectl exec -it app-3g-v1-1 bash
```

b. 执行以下命令， 查看GPU的使用情况。

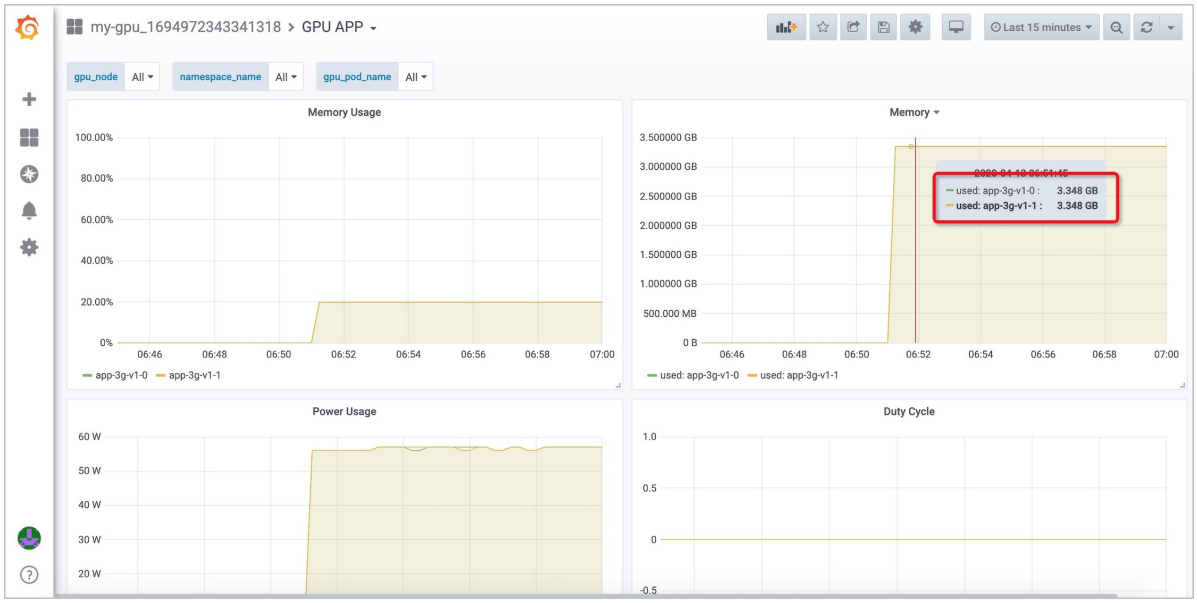
```
cuda_malloc -size=1024
```

预期输出：

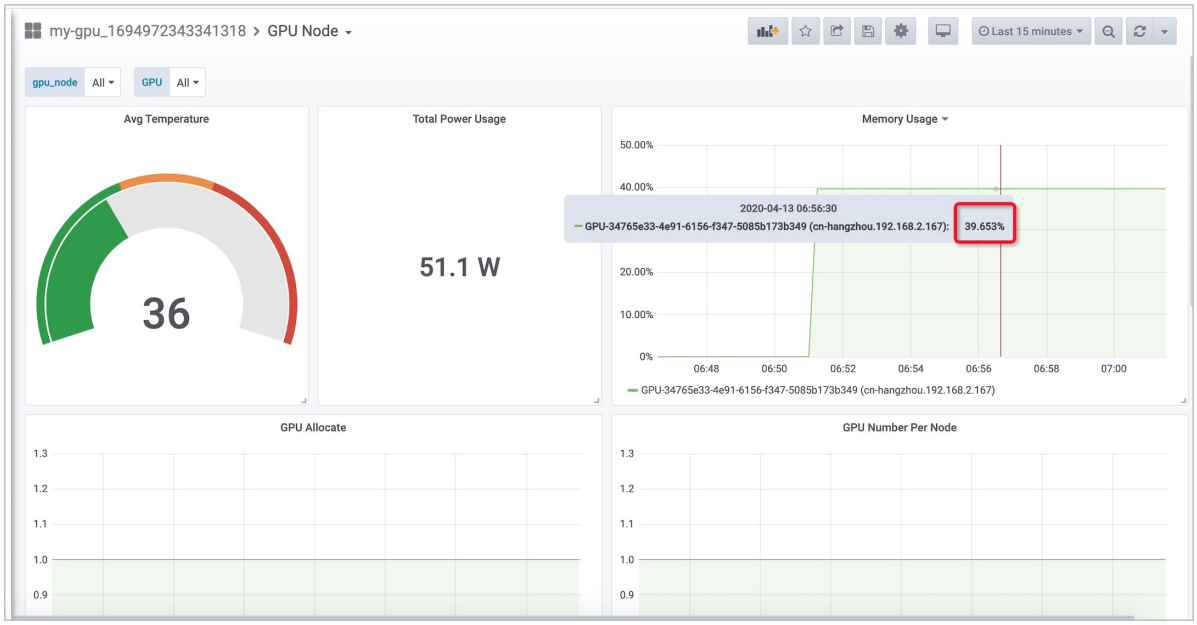
```
gpu_cuda_malloc starting...
Detected 1 CUDA Capable device(s)
Device 0: "Tesla V100-SXM2-16GB"
 CUDA Driver Version / Runtime Version       10.1 / 10.1
 Total amount of global memory:          4301 MBytes (4509925376 bytes)
Try to malloc 1024 MBytes memory on GPU 0
CUDA error at cgpu_cuda_malloc.cu:119 code=2(cudaErrorMemoryAllocation) "cudaMalloc( (void**)&dev_c, malloc_size)"
```

您可以通过ARMS控制台，从应用和节点两个维度来监控GPU的使用量。

- GPU APP：可以查看每个应用的GPU显存用量和占比。



- GPU Node：可以查看GPU卡的显存使用量。



使用托管Prometheus来监控共享GPU

当某个应用声明的GPU显存使用量超过了资源上限后，共享GPU方案中的GPU显存隔离模块可以确保其他应用不受影响。

1. 部署一个新的GPU应用。

   该应用声明使用的GPU显存是4 GB，但是它实际使用的GPU显存为6 GB。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: app-6g-v1
 labels:
  app: app-6g-v1
spec:
 replicas: 1
 serviceName: "app-6g-v1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: app-6g-v1
 template: # define the pods specifications
  metadata:
   labels:
    app: app-6g-v1
  spec:
   containers:
   - name: app-6g-v1
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/cuda-malloc:6G
    resources:
     limits:
      aliyun.com/gpu-mem: 4 #每个Pod申请了4 GB显存，副本数为1，该应用总共申请4 GB显存
```

2. 执行以下命令，查看Pod的状态。

   新应用的pod一直处于**CrashLoopBackOff**，而之前两个Pod还是正常运行的状态。

```
kubectl get pod
```

预期输出：

```
NAME      READY  STATUS       RESTARTS  AGE
app-3g-v1-0  1/1   Running      0      7h35m
app-3g-v1-1  1/1   Running      0      7h35m
app-6g-v1-0  0/1   CrashLoopBackOff  5      3m15s
```

3. 执行以下命令，查看容器的日志报错。

   可以看到报错是由于cudaErrorMemoryAllocation造成的。

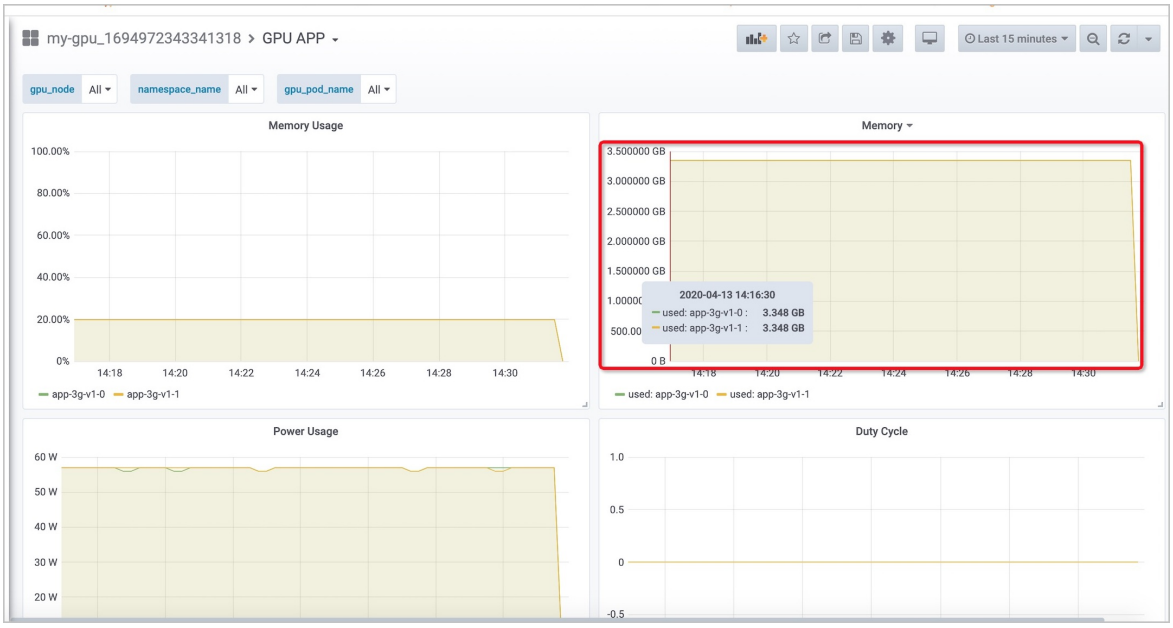
   ```
   kubectl logs app-6g-v1-0
   ```

   预期输出：

   ```
   CUDA error at cgpu_cuda_malloc.cu:119 code=2(cudaErrorMemoryAllocation) "cudaMalloc( (void**)&dev_c, malloc_size)"
   ```

4. 通过托管Prometheus的GPU APP组件来查看容器状态。

   可以看到之前的容器一直处于平稳运行之中，并没有受到新部署应用的影响。



18.3.2.4. 关闭共享GPU显存隔离能力

本文通过部署一个简单的示例应用向您介绍如何在集群中关闭共享GPU显存隔离能力。

适用场景

本文示例适用于已开启共享GPU显存隔离能力的专有版GPU集群及ACK Pro版集群。

前提条件

已安装共享GPU组件。具体操作，请参见安装共享GPU组件或安装并使用共享GPU组件和资源工具。

操作步骤

1. 执行以下命令查询集群的GPU共享能力。

   ```
   kubectl inspect cgpu
   ```

   预期输出：

```
NAME              IPADDRESS     GPU0(Allocated/Total)  GPU Memory(GiB)
cn-beijing.192.16x.x.xx3  192.16x.x.xx3  0/15           0/15
cn-beijing.192.16x.x.xx1  192.16x.x.xx1  0/15           0/15
cn-beijing.192.16x.x.xx2  192.16x.x.xx2  0/15           0/15
------------------------------------------------------------------------------
Allocated/Total GPU Memory In Cluster:
0/45 (0%)
```

> **说明** 如果您想查询GPU共享能力详细信息，请执行命令 `kubectl inspect cgpu -d` 。

2. 使用以下模板创建GPU共享容器，同时该容器不使用共享GPU的隔离能力。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: binpack
 labels:
  app: binpack
spec:
 replicas: 1
 serviceName: "binpack-1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: binpack-1
 template: # define the pods specifications
  metadata:
   labels:
    app: binpack-1
  spec:
   containers:
   - name: binpack-1
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/tensorflow-gpu-mem:10.0-runtime-centos7
    command:
     - python3
     - /app/main.py
    env:
     - name: CGPU_DISABLE #关闭cGPU隔离能力。
       value: "true"
    resources:
     limits:
      # GiB
      aliyun.com/gpu-mem: 3
```

> **说明**
>
> - aliyun.com/gpu-mem：设置GPU显存大小。
> - 关闭共享GPU显存隔离能力：设置容器的环境变量CGPU_DISABLE为true即可关闭cGPU。

3. 执行以下命令，查看共享GPU调度结果。

```
kubectl inspect cgpu
```

预期输出：

```
NAME              IPADDRESS     GPU0(Allocated/Total)  GPU Memory(GiB)
cn-beijing.192.16x.x.xx1  192.16x.x.xx1  0/15           0/15
cn-beijing.192.16x.x.xx2  192.16x.x.xx2  0/15           0/15
cn-beijing.192.16x.x.xx3  192.16x.x.xx3  3/15           3/15
-----------------------------------------------------------------------
Allocated/Total GPU Memory In Cluster:
3/45 (6%)
```

节点cn-beijing.192.16x.x.xx3分配了3 GiB显存给刚刚创建的容器。

执行结果

您可以通过以下两种方式验证部署的共享GPU显存隔离能力是否被关闭：

- 方式一：执行以下命令查看部署应用的日志。

  ```
  kubectl logs binpack-0 --tail=1
  ```

  预期输出：

  ```
  2020-08-25 08:14:54.927965: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1326] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 15024 MB memory) -> physical GPU (device: 0, name: Tesla V100-SXM2-16GB, pci bus id: 0000:00:07.0, compute capability: 7.0)
  ```

  从日志中可以看到，容器中的应用程序能够使用的显存为15024 MiB，可证明共享GPU显存隔离能力被关闭了（共享GPU显存隔离能力开启时，应用程序能够看到的显存为3 GiB）。

- 方式二：执行以下命令登录容器查看容器被分配显存总量。

  ```
  kubectl exec binpack-0 nvidia-smi
  ```

  预期输出：

  ```
  Tue Aug 25 08:23:33 2020
  +-----------------------------------------------------------------------------+
  | NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
  |-------------------------------+----------------------+----------------------+
  | 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 V100-SXM2...  Off  | 00000000:00:07.0 Off |                    0 |
  | N/A   33C    P0    55W / 300W |  15453MiB / 16130MiB |      1%      Default |
  +-------------------------------+----------------------+----------------------+
  +-----------------------------------------------------------------------------+
  | Processes:                                                       GPU Memory |
  |  GPU       PID   Type   Process name                             Usage      |
  |=============================================================================|
  +-----------------------------------------------------------------------------+
  ```

  从输出信息中可以看到宿主机的显存容量为16130 MiB，容器分配到的显存为15453 MiB，可证明共享GPU隔离能力未生效（共享GPU隔离能力开启时，容器分配到的显存为3 GiB）。

相关文档

- 共享GPU调度概述
- 安装共享GPU组件
- 监控和隔离GPU资源

18.3.2.5. 基于节点池管理共享GPU

基于节点池管理共享GPU能够为您提供更加灵活的GPU共享调度和显存隔离策略。本文通过在专有版GPU集群中创建两个含标签的节点池示例，介绍如何基于节点池管理共享GPU的调度能力和显存隔离能力。

适用场景

- 本文所描述的组件，仅支持专有版GPU集群，不支持托管版GPU集群。
- 若您想在ACK Pro版集群安装共享GPU组件，请参考安装并使用共享GPU组件和资源工具。
- 若您想在ACK Pro集群使用基于节点池管理共享GPU，请参考基于节点池管理共享GPU。

前提条件

您已完成以下操作：

- 安装共享GPU组件。
- 规划节点池。
  节点池名称可自定义，本文以cgpu和cgpu-no-isolation为例进行阐述。

节点池名称	共享调度	显存隔离	标签
cgpu	是	是	○ cgpu=true ○ cgpu.disable.isolation=false
cgpu-no-isolation	是	否	○ cgpu=true ○ cgpu.disable.isolation=true

背景信息

在阿里云容器服务Kubernetes版ACK（Container Service for Kubernetes）上使用GPU共享调度时，您可能会遇到以下场景：

- 训练任务A在代码中已经配置该任务可使用的GPU显存大小，此时集群只需要提供GPU共享调度能力即可，不需要提供GPU显存隔离能力。
- 训练任务B在代码中没有配置该任务可使用的GPU显存大小，此时集群不仅需要提供GPU共享调度能力，而且还需要提供GPU显存隔离能力。

那如何在一个集群中同时支持这两种场景呢？

基于节点池管理共享GPU的方法同时支持这两种场景。您只需要创建两个节点池：

- 第一个节点池的节点只提供共享调度能力，不提供显存隔离能力，该节点池只适用于运行训练任务A。
- 第二个节点池的节点同时提供共享调度能力和显存隔离能力，该节点池只适用于运行训练任务B。

注意事项

在使用节点池管理共享GPU的过程中，需要注意以下两点：

- 用节点池管理共享GPU时，如果某个任务没有写nodeSelector，那么该任务的Pod可能被调度到任意一个节点池，容易造成结果错乱。

  > **注意** 强烈建议给任务指定nodeSelector。

- 如果某个节点的特定标签发生变化（例如：cgpu.disable.isolation=false变为cgpu.disable.isolation=true），需要重启该节点的gpushare-device-plugin，显存隔离的配置才能生效。
  重启策略是将旧的gpushare-device-plugin Pod删除后，ACK会自动创建一个新的Pod，具体操作如下：
  i. 执行以下命令，查询集群的gpushare-device-plugin Pod。

```
kubectl get po -n kube-system  -l name=gpushare-device-plugin-ds -o wide
```

预期输出：

```
NAME                    READY  STATUS   RESTARTS  AGE  IP        NODE              NOMINATED NODE  READINES
S GATES
gpushare-device-plugin-ds-6r8gs  1/1   Running  0     18h  192.168.7.157  cn-shanghai.192.168.7.157  <non
e>      <none>
gpushare-device-plugin-ds-pjrvn  1/1   Running  0     15h  192.168.7.158  cn-shanghai.192.168.7.158  <non
e>      <none>
```

ii. 以cn-shanghai.192.168.7.157节点为例，执行以下命令，删除gpushare-device-plugin Pod。

```
kubectl delete po gpushare-device-plugin-ds-6r8gs -n kube-system
```

步骤一：创建节点池

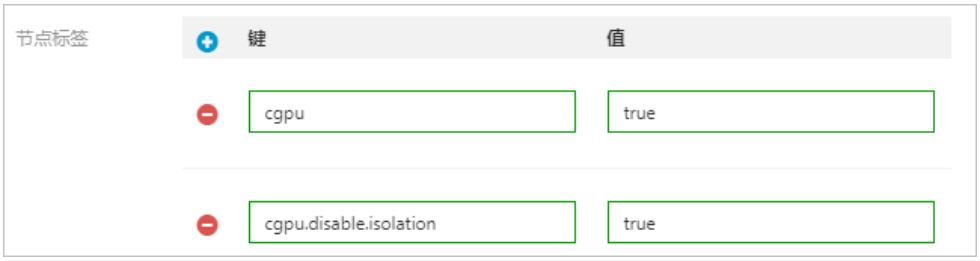
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面右上角，单击**创建节点池**。

   您还可以在**节点池**页面右上角，单击**创建托管节点池**或者**自动弹性伸缩配置**创建托管节点池或者具有弹性功能的节点池。
6. 在**创建节点池**页面，设置节点池的配置项。

   有关配置项的详细说明，请参见创建Kubernetes托管版集群。部分配置说明如下：
   - **数量**：设置节点池初始节点数量。如不需要创建节点，可以填写为0。
   - **操作系统**：选择节点的操作系统，包括CentOS 7.x、Alibaba Cloud Linux 2.x。
   - **节点标签**：您可以为集群节点添加标签。
   - **ECS标签**：您可以为ECS实例添加标签。
   - **自定义资源组**：您可以指定节点池所扩容节点的资源组信息。

   其中，在**节点标签**区域为每个节点池打上特定的标签。
   - 节点池cgpu需要的特定标签：cgpu=true、cgpu.disable.isolation=false
   - 节点池cgpu-no-isolation需要的特定标签：cgpu=true、cgpu.disable.isolation=true

   以节点池cgpu-no-isolation为例：



7. 单击**确认配置**。
   在**节点池**页面，如果节点池**状态**显示**初始化中**，则说明节点池正在创建中。创建完成后，**状态**显示为**已激活**。

> **说明** 如果您需要增加GPU节点，可对该节点进行扩容。具体操作，请参见扩容节点池。

步骤二：提交任务

提交cgpu-test和cgpu-test-no-isolation两个任务，在部署任务的YAML文件中均需要指定nodeSelector。

- cgpu-test：该任务的代码中没有设置可用的GPU显存大小，需要借助共享的显存隔离能力才能正常运行。示例YAML配置如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: cgpu-test
 labels:
  app: cgpu-test
spec:
 replicas: 1
 serviceName: "cgpu-test"
 podManagementPolicy: "Parallel"
 selector:     # define how the deployment finds the pods it manages.
  matchLabels:
   app: cgpu-test
 template:     # define the pods specifications.
  metadata:
   labels:
    app: cgpu-test
  spec:
   nodeSelector:     # 添加nodeSelector，选择节点池cgpu。
    cgpu.disable.isolation: "false"
   containers:
   - name: cgpu-test
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/tensorflow-gpu-mem:10.0-runtime-centos7
    command:
     - python3
     - /app/main.py
    env:
    resources:
     limits:     # 申请的显存单位为GiB，该容器申请了3 GiB显存。
      aliyun.com/gpu-mem: 3
```

> 说明
>
> - nodeSelector：指定节点池为cgpu。
> - cgpu.disable.isolation=false：表示将任务调度到节点池cgpu的节点上运行。
> - aliyun.com/gpu-mem：设置GPU显存大小。

- cgpu-test-no-isolation：该任务的代码中设置了每个GPU可用显存大小，不需要借助cgpu的显存隔离能力。示例YAML配置如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: cgpu-test-no-isolation
 labels:
  app: cgpu-test-no-isolation
spec:
 replicas: 1
 serviceName: "cgpu-test-no-isolation"
 podManagementPolicy: "Parallel"
 selector:       # define how the deployment finds the pods it manages
  matchLabels:
   app: cgpu-test-no-isolation
 template:       # define the pods specifications
  metadata:
   labels:
    app: cgpu-test-no-isolation
  spec:
   nodeSelector:    # 添加nodeSelector，选择节点池cgpu-no-isolation。
    cgpu.disable.isolation: "true"
   containers:
   - name: cgpu-test-no-isolation
    image: cheyang/gpu-player:v2
    resources:
     limits:   # 申请了3 GiB显存。
      aliyun.com/gpu-mem: 3
```

> 说明
>
> - nodeSelector：指定节点池为cgpu-no-isolation。
> - cgpu.disable.isolation=true：表示将任务调度到节点池cgpu-no-isolation的节点上运行。
> - aliyun.com/gpu-mem：设置GPU显存大小。

步骤三：查看执行结果

1. 执行以下命令，查看任务状态。

   ```
   kubectl get po
   ```

   预期输出：

   ```
   NAME                       READY  STATUS   RESTARTS  AGE
   cgpu-test-0                  1/1   Running  0      5m55s
   cgpu-test-no-isolation-0  1/1   Running  0      6m42s
   ```

2. 在Pod cgpu-test-0（需要显存隔离的任务）执行命令 `nvidia-smi` ，查看容器可使用的显存。

   ```
   kubectl exec cgpu-test-0 nvidia-smi
   ```

   预期输出：

```
Mon Nov  2 11:33:10 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   34C    P0    54W / 300W |   3039MiB /  3226MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

容器可使用的显存为3226 MiB（该GPU卡的总显存为16 GiB），说明共享GPU的显存隔离能力已生效。

3. 在Pod cgpu-test-no-isolation-0（不需要显存隔离的任务）执行 `nvidia-smi` ，查看容器可使用的显存。

```
kubectl exec cgpu-test-no-isolation-0 nvidia-smi
```

预期输出：

```
Mon Nov  2 11:39:59 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   37C    P0    56W / 300W |   1929MiB / 16130MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

容器可发现的显存为16130 MiB（该GPU卡的总显存为16 GiB），说明共享GPU的显存隔离能力已被关闭，此种情况下，容器中的应用程序需要通过以下环境变量获取自己能够使用的显存大小。执行以下命令，获取能够使用的显存大小。

```
kubectl exec cgpu-test-no-isolation-0 env | grep ALIYUN
```

预期输出：

```
ALIYUN_COM_GPU_MEM_CONTAINER=3  # 在某张卡上，该容器被允许使用的显存大小，此处代表3 GiB。
ALIYUN_COM_GPU_MEM_DEV=15  # 这张卡的总显存大小。
...
```

4. 对比在Pod cgpu-test-no-isolation-0和cgpu-test-0中执行 `nvidia-smi` 的结果。
cgpu-test-no-isolation-0显示整张GPU卡的显存，而cgpu-test-0只能显示申请的GPU显存大小，说明基于节点池可以有效管理共享GPU。

18.3.3. 专业版

18.3.3.1. 安装并使用共享GPU组件和资源工具

ACK提供了GPU共享调度能力，服务于共享单GPU的模型预测场景，同时也可以通过Nvidia驱动内核态保障GPU显存的隔离。本文介绍如何在GPU节点上安装共享GPU组件和GPU资源查询工具，实现GPU的调度和隔离能力。

前提条件

-
- 创建ACK Pro集群。ACK Pro集群的实例规格类型需要设置为**异构计算GPU/FPGA/NPU**，其他配置请参见创建ACK Pro版集群。

  > **说明** 仅支持在ACK Pro集群安装共享GPU组件，如果您使用的是专有版集群，您可以提交工单申请白名单。

- 通过kubectl工具连接集群。
- 共享GPU组件的安装没有地域限制，但是目前只有以下地域支持GPU显存隔离能力。如果您需要使用显存隔离能力，请确保集群所在地域在此范围内。

地域	地域ID
华北2（北京）	cn-beijing
华东2（上海）	cn-shanghai
华东1（杭州）	cn-hangzhou
华北（张家口）	cn-zhangjiakou
华南1（深圳）	cn-shenzhen
西南1（成都）	cn-chengdu
华南2（河源）	cn-heyuan
中国（香港）	cn-hongkong
印度尼西亚（雅加达）	ap-southeast-5
新加坡	ap-southeast-1
美国（弗吉尼亚）	us-east-1
美国（硅谷）	us-west-1

使用须知

配置	支持版本
Kubernetes	1.18.8及其以上
Helm版本	3.0及以上版本
Nvidia驱动版本	418.87.01及以上版本
Docker版本	19.03.5

配置	支持版本
操作系统	CentOS 7.6、CentOS 7.7、Ubuntu 16.04和Ubuntu 18.04，Alibaba Cloud Linux 2.x
支持显卡	Tesla P4、Tesla P100、 Tesla T4和Tesla v100

步骤一：安装共享GPU组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**应用 > 云原生AI套件（公测）**。
5. 在**云原生AI套件**页面，单击**一键部署**。
6. 在一键部署**云原生AI套件**页面，选中**调度组件（批量任务调度、GPU共享、GPU拓扑感知、NPU调度）**，单击页面下方的**部署云原生AI套件**。



> **说明** 如果您已经部署了云原生AI套件，则在**云原生AI套件**页面的组件列表，单击ack-ai-installer右侧**操作**列下的**部署**。

组件安装成功后，在**云原生AI套件**页面的组件列表中能看到已安装的共享GPU组件**ack-ai-installer**。

步骤二：开启GPU共享调度能力和显存隔离能力

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面单击已安装ack-ai-installer组件集群**操作**列的**节点池**。
4. 在**节点池**页面单击**创建节点池**。
5. 在**创建节点池**页面，设置创建节点池的配置项。

   关于配置项的详细说明，请参见创建ACK Pro版集群。部分配置项说明如下：
   - 数量：设置节点池初始节点数量。如不需要创建节点，可以填写为0。
   - 操作系统：选择节点的操作系统，包括CentOS、Alibaba Cloud Linux 2.x、Windows。
   - ECS标签：您可以为ECS实例添加标签。
   - 自定义资源组：您可以指定节点池所扩容节点的资源组信息。
   - 节点标签：为集群节点添加标签。关于节点标签的详细说明，请参见ACK调度GPU使用的节点标签说明。
     - 开启GPU共享调度能力，同时开启GPU显存隔离能力。

       单击**节点标签**的，设置**键**为ack.node.gpu.schedule，**值**为cgpu。

       > **说明** 如果只需要节点池支持共享调度，不支持显存隔离，您可以设置**节点标签**的**键**为ack.node.gpu.schedule，**值**为share。

- 在节点上使用Binpack算法为Pod选择GPU卡。
  单击**节点标签**的，设置**键**为ack.node.gpu.placement，设置**值**为binpack。

6. 单击**确认配置**。

步骤三：添加GPU节点

完成创建节点池后，您还可以在节点池中添加GPU节点。添加GPU节点时，您需要将实例规格类型设置为**异构计算GPU/FPGA/NPU**。具体操作，请参见添加已有节点或扩容节点池。

> **说明** 如果您添加节点池时已经创建GPU节点，您可以跳过此步骤。

步骤四：安装和使用GPU资源查询工具

1. 配置kubeconfig文件。具体操作，请参见通过kubectl工具连接集群。
2. 下载kubectl-inspect-cgpu。
   - 如果您使用的是Linux系统，您可以通过以下命令下载kubectl-inspect-cgpu。

     ```
     wget http://aliacs-k8s-cn-beijing.oss-cn-beijing.aliyuncs.com/gpushare/kubectl-inspect-cgpu-linux -O /usr/local/bin/kubectl-inspect-cgpu
     ```

   - 如果您使用的是macOS系统，您可以通过以下命令下载kubectl-inspect-cgpu。

     ```
     wget http://aliacs-k8s-cn-beijing.oss-cn-beijing.aliyuncs.com/gpushare/kubectl-inspect-cgpu-darwin -O /usr/local/bin/kubectl-inspect-cgpu
     ```

3. 为kubectl-inspect-cgpu添加执行权限。

   ```
   chmod +x /usr/local/bin/kubectl-inspect-cgpu
   ```

4. 查看集群GPU使用情况。

   ```
   kubectl inspect cgpu
   ```

   预期输出：

   ```
   NAME                IPADDRESS     GPU0(Allocated/Total)  GPU Memory(GiB)
   cn-shanghai.192.168.6.104  192.168.6.104  0/15                   0/15
   ------------------------------------------------------------------------------
   Allocated/Total GPU Memory In Cluster:
   0/15 (0%)
   ```

18.3.3.2. 运行共享GPU示例

本文介绍通过部署创建GPU共享容器的YAML文件说明如何使用cGPU显存隔离能力，高效利用GPU设备资源。

前提条件

安装并使用共享GPU组件和资源工具

操作步骤

1. 执行以下命令查询集群的GPU共享能力。

   ```
   kubectl inspect cgpu
   ```

```
NAME             IPADDRESS   GPU0(Allocated/Total)  GPU1(Allocated/Total)  GPU Memory(GiB)
cn-shanghai.192.168.0.4  192.168.0.4  0/7              0/7              0/14
------------------------------------------------------------------------
Allocated/Total GPU Memory In Cluster:
0/14 (0%)
```

> **说明** 如果想查询GPU共享能力详细信息，执行命令kubectl inspect cgpu -d。

2. 部署共享GPU示例应用，该应用申请3 GiB显存。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: binpack
 labels:
  app: binpack
spec:
 replicas: 1
 serviceName: "binpack-1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: binpack-1
 template: # define the pods specifications
  metadata:
   labels:
    app: binpack-1
  spec:
   containers:
   - name: binpack-1
    image: registry.cn-beijing.aliyuncs.com/ai-samples/gpushare-sample:tensorflow-1.5
    command:
     - python
     - cgpu/main.py
    resources:
     limits:
      # GiB
      aliyun.com/gpu-mem: 3
```

> **说明** aliyun.com/gpu-mem：设置GPU显存大小。

3. 执行以下命令，查看集群GPU显存使用情况。

```
kubectl inspect cgpu
```

预期输出：

```
NAME             IPADDRESS    GPU0(Allocated/Total)  GPU Memory(GiB)
cn-beijing.192.168.1.105  192.168.1.105  3/14             3/14
------------------------------------------------------------------------
Allocated/Total GPU Memory In Cluster:
3/14 (21%)
```

可以看到节点cn-beijing.192.168.1.105总共有14 GiB显存，已分配3 GiB显存。

执行结果

您可以通过以下方式验证节点GPU显存隔离能力是否生效。

- 执行以下命令查看步骤2中部署应用的日志。
  通过查看日志，您可以确定cGPU显存隔离是否成功部署。

  ```
  kubectl logs binpack-0 --tail=1
  ```

  预期输出：

  ```
  2020-03-13 09:14:13.931003: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1326] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 2832 MB memory) -> physical GPU (device: 0, name: Tesla T4, pci bus id: 0000:00:07.0, compute capability: 7.5)
  ```

  可以看到容器申请的显存为2832 MiB。
- 执行以下命令登录容器查看容器被分配显存总量。

  ```
  kubectl exec -it binpack-0 nvidia-smi
  ```

  预期输出：

  ```
  Fri Mar 13 09:32:18 2020
  +-----------------------------------------------------------------------------+
  | NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
  |-------------------------------+----------------------+----------------------+
  | 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 T4            On   | 00000000:00:07.0 Off |                    0 |
  | N/A   41C    P0    26W /  70W |   3043MiB /  3231MiB |      0%      Default |
  +-------------------------------+----------------------+----------------------+

  +-----------------------------------------------------------------------------+
  | Processes:                                                       GPU Memory |
  |  GPU       PID   Type   Process name                             Usage      |
  |=============================================================================|
  +-----------------------------------------------------------------------------+
  ```

  可以看到该容器被分配显存总量为3231 MiB 。
- 执行以下命令查看示例应用所在节点的GPU显存总量（在节点上执行）。

  ```
  nvidia-smi
  ```

  预期输出：

```
Fri Mar 13 17:36:24 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01   Driver Version: 418.87.01   CUDA Version: 10.1    |
|-------------------------------+----------------------+----------------------+
| 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 T4        On   | 00000000:00:07.0 Off |                    0 |
| N/A  40C   P0   26W /  70W |  3053MiB / 15079MiB |    0%    Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
| GPU     PID   Type   Process name                             Usage      |
|=============================================================================|
|  0    8796    C  python3                                    3043MiB |
+-----------------------------------------------------------------------------+
```

可以看到主机上的显存总量为15079 MiB，其中分配给容器的是3053 MiB。

18.3.3.3. 通过共享GPU实现多卡共享策略

ACK Pro版集群支持共享GPU，共享GPU能够在Kubernetes上实现GPU共享调度和显存隔离。本文介绍如何设置共享GPU的多卡共享策略。

前提条件

已安装共享GPU组件和资源工具。具体操作，请参见安装并使用共享GPU组件和资源工具。

背景信息

多卡共享策略指的是某个应用申请了N个GiB的显存，并指定了这N个GiB的显存由M块GPU卡分配，每块GPU卡分配的显存为N/M（目前N/M必须为整数，并且这M张GPU卡必须在同一个Kubernetes节点上）。例如，某个应用申请了8 GiB显存，并指定了GPU卡个数为4，那么某个节点需分配4块GPU卡给该应用，每块GPU卡分配2 GiB显存。

操作步骤

1. 创建*binpack-1.yaml*。

   使用YAML文件创建一个资源类型为StatefulSet的binpack-1应用，并在YAML文件中设置以下配置。

   - 设置Replicas为1。
   - 申明每个Pod使用4张GPU卡，每张GPU卡使用2 GiB显存。
     - 给Pod添加标签 `aliyun.com/gpu-count=4` 。

       > 注意 将标签添加在 `spec.template.metadata.labels` 下。

     - 指定Pod的扩展资源 `aliyun.com/gpu-mem` 数量为8。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: binpack-1
 labels:
  app: binpack-1
spec:
 # 副本数设置1，便于观察。
 replicas: 1
 serviceName: "binpack-1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: binpack-1
 template: # define the pods specifications
  metadata:
   labels:
    app: binpack-1
    # 申明8 GiB显存由4块GPU卡提供，每块提供2 GiB显存。
    aliyun.com/gpu-count: "4"
  spec:
   containers:
   - name: binpack-1
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/tensorflow-gpu-mem:10.0-runtime-centos7
    resources:
     limits:
      # 单位为GiB，该Pod总共申请了8 GiB显存。
      aliyun.com/gpu-mem: 8
```

2. 部署binpack-1应用。

```
kubectl apply -f binpack-1.yaml
```

验证多卡共享策略

1. 查看Pod运行状态。

```
kubectl get pod
```

预期输出：

```
NAME        READY  STATUS   RESTARTS  AGE
binpack-1-0    1/1   Running  0     9m34s
```

2. 查看GPU数量和GPU的显存。

```
kubectl exec binpack-1-0 -- nvidia-smi
```

预期输出：

```
Tue Nov  3 09:05:05 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A  31C   P0    54W / 300W |   2084MiB /  2150MiB |     1%      Default |
+-------------------------------+----------------------+----------------------+
|  1  Tesla V100-SXM2...  On   | 00000000:00:08.0 Off |                    0 |
| N/A  32C   P0    55W / 300W |   2084MiB /  2150MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|  2  Tesla V100-SXM2...  On   | 00000000:00:09.0 Off |                    0 |
| N/A  32C   P0    54W / 300W |   2084MiB /  2150MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|  3  Tesla V100-SXM2...  On   | 00000000:00:0A.0 Off |                    0 |
| N/A  34C   P0    54W / 300W |   2084MiB /  2150MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

可以看到使用了4张GPU卡，且每张GPU卡的显存大小与所申请的显卡大小基本一致，GPU卡的显存大小为2 GiB。

> **说明** 每张GPU卡的可使用显存大小与所申请的显卡大小有可能存在几十到几百兆的误差。

3. （可选）查看容器使用的GPU卡。

```
kubectl exec binpack-1-0 -- env | grep NVIDIA_VISIBLE_DEVICES
```

预期输出：

```
NVIDIA_VISIBLE_DEVICES=GPU-c09d11f5-a565-1f83-903c-a65a7ba138af,GPU-959873c5-b57b-5d24-f3aa-e700c5d431cb,GPU-abdcd4da-7505-c267-fb4d-30478a360e07,GPU-ba997778-f8ce-e433-3c84-6d15392a06a1
```

4. （可选）通过环境变量 `ALIYUN_COM_GPU_MEM_CONTAINER` 查看容器在每张GPU卡上可使用的显存大小。

```
kubectl exec binpack-1-0 -- env | grep ALIYUN
```

预期输出：

```
ALIYUN_COM_GPU_MEM_CONTAINER=2 # 该值表示每张卡上该容器可使用的显存为2 GiB，容器内的应用程序可以读取该值设置显存限制。
ALIYUN_COM_GPU_MEM_DEV=15 # 每张卡的总显存为15 GiB。
```

18.3.3.4. 基于节点池管理共享GPU

ACK Pro集群支持通过节点池管理共享GPU的调度能力和显存隔离能力。本文介绍如何基于节点池管理共享GPU。

前提条件

- Helm版本≥3.0.0。
- 创建ACK Pro集群。
  创建ACK Pro集群时，需要按以下要求设置Kubernetes版本和实例规格类型，其他配置请参见创建ACK Pro版集群。

> **说明** 仅支持在ACK Pro集群安装共享GPU组件，如果您使用的是专有版集群，您可以提交工单申请白名单。

- Kubernetes版本≥1.18.8。
- 设置实例规格类型为**异构计算GPU/FPGA/NPU**。

- 目前只有以下地域支持GPU显存隔离功能。如果您需要使用该功能，请确保集群所在地域在此范围内。

地域	地域ID
华北2（北京）	cn-beijing
华东2（上海）	cn-shanghai
华东1（杭州）	cn-hangzhou
华北（张家口）	cn-zhangjiakou
华南1（深圳）	cn-shenzhen
西南1（成都）	cn-chengdu
华南2（河源）	cn-heyuan
中国（香港）	cn-hongkong
印度尼西亚（雅加达）	ap-southeast-5
新加坡	ap-southeast-1
美国（弗吉尼亚）	us-east-1
美国（硅谷）	us-west-1

- 规划节点池的名称、节点标签、共享调度和显存隔离能力。
  节点池名称可自定义，本文以cgpu和cgpu-no-isolation为例进行阐述。关于节点标签的详细说明请参见ACK调度GPU使用的节点标签说明。

节点池名称	共享调度	显存隔离	节点标签
cgpu-no-isolation	是	否	ack.node.gpu.schedule=share ack.node.gpu.placement=binpack
cgpu	是	是	ack.node.gpu.schedule=cgpu ack.node.gpu.placement=binpack

适用场景

在ACK上使用GPU共享调度时，您可能遇到以下场景：

- 训练任务A在代码中已配置该任务可使用的GPU显存大小，此时只需要提供GPU共享调度能力，不需要提供GPU显存隔离能力。
- 训练任务B在代码中没有配置该任务可使用的GPU显存大小，此时不仅需要提供GPU共享调度能力，还需要提供GPU显存隔离能力。

基于节点池管理共享GPU能够在一个集群支持以上两种场景。您只需要在集群中创建以下两个节点池：

- 第一个节点池的节点只提供共享调度能力，不提供显存隔离能力，该节点池只适用于运行训练任务A。
- 第二个节点池的节点同时提供共享调度能力和显存隔离能力，该节点池只适用于运行训练任务B。

注意事项

在使用节点池管理共享GPU的过程中，需要注意以下两点：

- 建议给任务设置nodeSelector。
  使用节点池管理共享GPU时，如果某个任务没有设置nodeSelector，那么该任务的Pod可能被调度到任意一个节点池。例如，某些Pod不需要显存隔离，结果被调度到具有显存隔离的节点。
- 如果您需要关闭显存隔离能力，需要在节点卸载cGPU的Kernel Module并重启机器后才能生效。
  - 如果特定标签由ack.node.gpu.schedule=cgpu变成ack.node.gpu.schedule=share，该节点不会关闭显存隔离能力，需要在节点卸载cGPU的Kernel Module并重启机器后才能生效。有关如何卸载cGPU请参见通过Docker安装并使用cGPU服务。
  - 如果特定标签ack.node.gpu.schedule=share变成ack.node.gpu.schedule=cgpu，该节点会直接由不支持显存隔离变成支持显存隔离。

步骤一：安装共享GPU组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的搜索框中搜索ack-ai-installer，找到ack-ai-installer，单击**ack-ai-installer**。
4. 在**应用目录-ack-ai-installer**页面创建区域选择目标集群，然后单击**创建**。
   安装ack-ai-installer成功后，将会跳转到ack-ai-installer详情页面，您可以看到ack-ai-installer下的各个组件。

步骤二：创建节点池

创建两个不同能力的节点池，一个节点池支持共享调度和显存隔离，另一个节点池仅支持共享调度，不支持显存隔离。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面单击已安装共享GPU组件集群**操作**列的**节点池**。
4. 在**节点池**页面单击右上角的**创建节点池**。
5. 在**创建节点池**页面，设置创建节点池的配置项。

   有关配置项的详细说明，请参见创建ACK Pro版集群。部分配置项说明如下：
   - 节点池名称：设置**节点池名称**为cgpu。
   - 数量：设置节点池初始节点数量。如不需要创建节点，可以填写为0。
   - 操作系统：选择节点的操作系统，包括CentOS 7.x、Alibaba Cloud Linux 2.x。
   - ECS标签：您可以为ECS实例添加标签。
   - 自定义资源组：您可以指定节点池所扩容节点的资源组信息。
   - 节点标签：为集群节点添加标签。
     - 开启GPU共享调度能力，同时开启GPU显存隔离能力。
       单击**节点标签**的，设置**键**为ack.node.gpu.schedule，**值**为cgpu。

- 在节点上使用Binpack算法为Pod选择GPU卡。
  单击**节点标签**的，设置**键**为ack.node.gpu.placement，设置**值**为binpack。

6. 重复步骤和，创建cgpu-no-isolation节点池。

   创建cgpu-no-isolation节点池时，设置节点池名称为cgpu-no-isolation，并添加ack.node.gpu.schedule=share和ack.node.gpu.placement=binpack标签。

步骤三：添加GPU节点

完成创建节点池后，您还可以在节点池中添加GPU节点。添加GPU节点时，您需要将实例规格类型设置为**异构计算GPU/FPGA/NPU**。具体操作，请参见添加已有节点或扩容节点池。

> **说明** 如果您添加节点池时已经创建GPU节点，您可以跳过此步骤。

步骤四：提交任务

为了验证创建的节点池是否具有GPU共享调度和显存隔离的能力，需要提交两个任务：

- cgpu-test：该任务没有设置可用的GPU显存大小，需要借助cgpu的显存隔离能力才能正常运行。
- cgpu-test-no-isolation：该任务设置了每个GPU可用显存大小，不需要借助cgpu的显存隔离能力。

提交cgpu-test-no-isolation任务

1. 创建*cgpu-test-no-isolation.yaml*。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: cgpu-test-no-isolation
 labels:
  app: cgpu-test-no-isolation
spec:
 replicas: 1
 serviceName: "cgpu-test-no-isolation"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: cgpu-test-no-isolation
 template: # define the pods specifications
  metadata:
   labels:
    app: cgpu-test-no-isolation
  spec:
   nodeSelector:
    # 需指定该任务调度到哪些节点上。
    # 由带有标签ack.node.gpu.schedule=share的节点组成了节点池cgpu-no-isolation。
    ack.node.gpu.schedule: "share"
   containers:
   - name: cgpu-test-no-isolation
    image: cheyang/gpu-player:v2
    resources:
     limits:
      # 申请了3 GiB显存。
      aliyun.com/gpu-mem: 3
```

   - aliyun.com/gpu-mem：设置GPU显存大小。
   - nodeSelector：指定节点池为cgpu-no-isolation。

2. 提交cgpu-test-no-isolation任务。

```
kubectl apply -f cgpu-test-no-isolation.yaml
```

提交cgpu-test任务

1. 创建*cgpu-test.yaml*。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: cgpu-test
 labels:
  app: cgpu-test
spec:
 replicas: 1
 serviceName: "cgpu-test"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: cgpu-test
 template: # define the pods specifications
  metadata:
   labels:
    app: cgpu-test
  spec:
   nodeSelector:
    # 加node selector，选择节点池cgpu。
    ack.node.gpu.schedule: "cgpu"
   containers:
   - name: cgpu-test
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/tensorflow-gpu-mem:10.0-runtime-centos7
    command:
     - python3
     - /app/main.py
    env:
    resources:
     limits:
      # 申请的显存单位为GiB，该容器申请了3 GiB显存。
      aliyun.com/gpu-mem: 3
```

   - aliyun.com/gpu-mem：设置GPU显存大小。
   - nodeSelector：指定节点池为cgpu。

2. 提交cgpu-test任务。

```
kubectl apply -f cgpu-test.yaml
```

步骤五：结果验证

如果设置了显存隔离，那么容器只能看到GPU卡分配给容器的显存。如果没有设置显存隔离，那么容器可以看到整个GPU卡的显存。执行以下命令，查看容器能看到的GPU卡的显存。

- 查看任务状态。

```
kubectl get po
```

预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE
cgpu-test-0           1/1   Running  0       5m55s
cgpu-test-no-isolation-0  1/1   Running  0       6m42s
```

- 查看cgpu-test-0容器可以看到的显存。

```
kubectl exec cgpu-test-0 nvidia-smi
```

预期输出：

```
Mon Nov  2 11:33:10 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01   Driver Version: 418.87.01   CUDA Version: 10.1    |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |              0 |
| N/A   34C    P0    54W / 300W |   3039MiB /  3226MiB |     1%     Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU      PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

cgpu-test-0容器可以看到的显存为3226 MiB（该GPU卡的总显存为16 GiB），说明cGPU的显存隔离能力生效了。

- 查看cgpu-test-no-isolation-0容器可以看到的显存。

```
kubectl exec cgpu-test-no-isolation-0 nvidia-smi
```

预期输出：

```
Mon Nov  2 11:39:59 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01   Driver Version: 418.87.01   CUDA Version: 10.1    |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |              0 |
| N/A   37C    P0    56W / 300W |   1929MiB / 16130MiB |     1%     Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU      PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

cgpu-test-no-isolation-0容器可以看到的显存为16130 MiB（该GPU卡的总显存为16 GiB），说明没有设置显存隔离。
对比cgpu-test-no-isolation-0和cgpu-test-0容器。cgpu-test-no-isolation-0能够看到整张GPU卡的显存，而cgpu-test-0只能看到申请的GPU显存大小，说明基于节点池管理共享GPU是成功的。

18.3.3.5. 在ACK Pro版集群中将共享GPU基础版升级为共享GPU专业版

ACK专有版集群安装的共享GPU基础版在ACK Pro版集群中是无法正常使用的，ACK Pro集群仅支持共享GPU专业版。所以当集群由ACK专有版集群迁移至ACK Pro版集群后，共享GPU组件也需要升级。本文介绍如何在ACK Pro版集群将共享GPU基础版升级为共享GPU专业版。

前提条件

已将ACK专有版集群迁移至ACK Pro托管版集群中，且迁移前的ACK专有版集群已安装共享GPU基础版。具体操作，请参见热迁移ACK专有版集群至Pro托管集群。

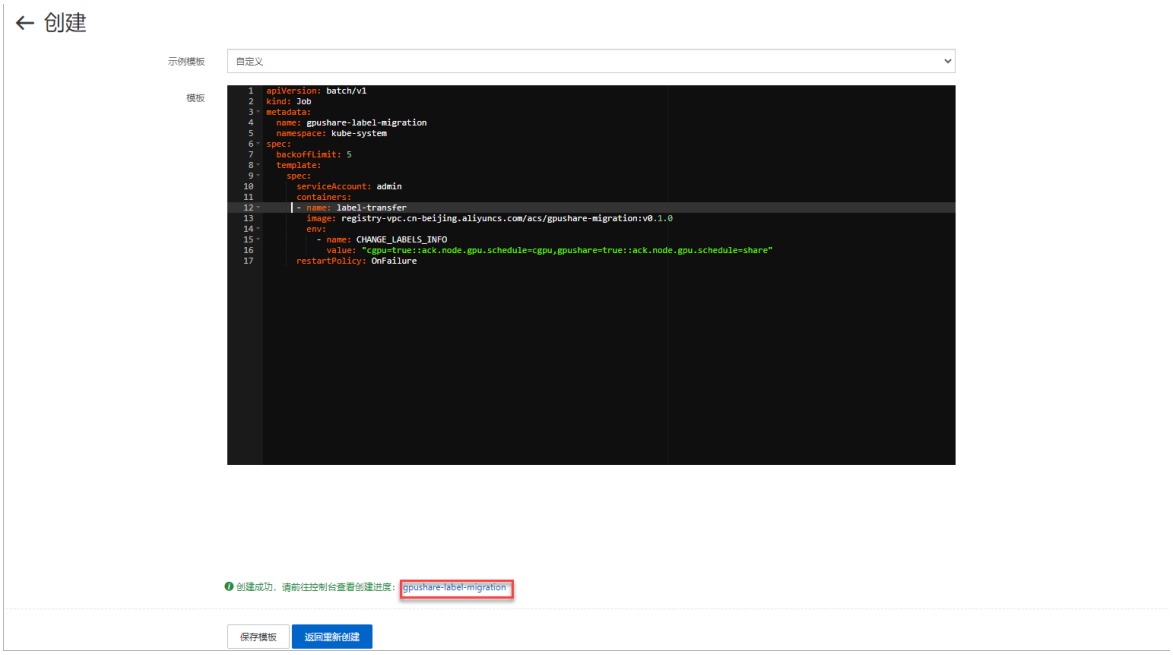
操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 任务**。
5. 在**任务**页面，单击右上角的**使用YAML创建资源**。
6. 在**创建**页面，选择**自定义**的**示例模板**，将以下YAML示例复制到**模板**。

   该任务用于卸载旧版共享GPU组件及转换共享GPU节点标签。

```
apiVersion: batch/v1
kind: Job
metadata:
 name: gpushare-label-migration
 namespace: kube-system
spec:
 backoffLimit: 5
 template:
  spec:
   serviceAccount: admin
   containers:
   - name: label-transfer
    image: registry-vpc.cn-beijing.aliyuncs.com/acs/gpushare-migration:v0.1.0
    env:
     - name: CHANGE_LABELS_INFO
       value: "cgpu=true::ack.node.gpu.schedule=cgpu,gpushare=true::ack.node.gpu.schedule=share"
   restartPolicy: OnFailure
```

7. 单击**创建**，单击任务gpushare-label-migration的链接查看创建进度。



在任务**gpushare-label-migration**详情页面的**容器组**页签下，若任务的状态为**Completed**，表示任务运行成功。

8. 安装共享GPU专业版。具体操作，请参见步骤一：安装共享GPU组件。
9. 安装查询集群GPU显存使用情况的工具。具体操作，请参见步骤四：安装和使用GPU资源查询工具。

后续步骤

关于如何验证共享GPU专业版的共享调度及显存隔离能力，请参见运行共享GPU示例。

18.4. GPU监控

18.4.1. 使用Kubernetes事件中心监控GPU异常

本文介绍如何通过Kubernetes事件中心对GPU Xid错误进行监控告警，帮助您实时了解和诊断NVIDIA驱动程序的错误报告。

前提条件

- 创建托管GPU集群或者创建专有GPU集群
- 创建并使用Kubernetes事件中心

背景信息

Xid消息是来自NVIDIA驱动程序的错误报告，该报告会打印到操作系统的内核日志或事件日志中。Xid消息表明发生了一般的GPU错误，通常是由于驱动程序对GPU的编程不正确或发送给GPU的命令损坏所致。这些消息可能表示硬件问题、NVIDIA软件问题或用户应用程序问题。

GPU设备在使用中，容易发生一些Xid错误，可以配合Kubernetes事件中心，对这些Xid错误进行监控告警，及时发现并定位故障原因。

操作步骤

1. 进入日志服务**K8s事件中心**页面。

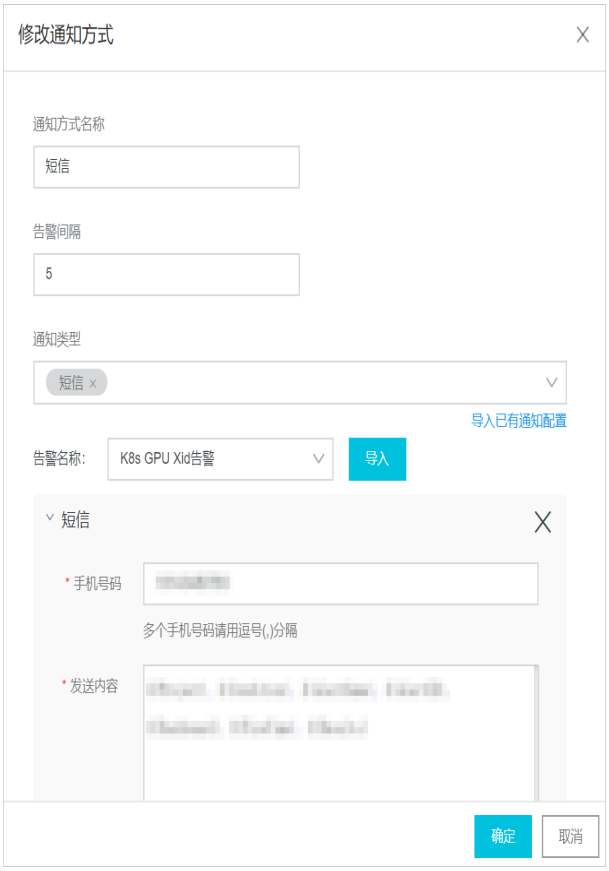
   具体操作步骤，请参见创建并使用Kubernetes事件中心。

2. 在**K8s事件中心**左侧导航栏中，单击目标集群，然后单击**事件总览**。
   在**事件总览**页面查看GPU Xid告警统计信息及Xid错误的详细内容。



3. 在目标集群下拉菜单中，单击**告警配置**。
4. 单击**添加通知方式**，在**添加通知方式**面板，配置通知方式，然后单击**确定**。

   可以根据需要选择短信、邮件、钉钉等接收通知，再自定义通知内容，下图以短信为例。



5. 通知方式配置完成后，在**全部告警事件**页面右上角单击**修改**，选择**K8s GPU Xid告警**，并在**K8s GPU Xid告警**下拉框中选择**短信**。



6. 在**全部告警事件**页面，单击**保存**。
   触发告警后，您会收到阿里云发出的告警短信。

18.4.2. 使用ARMS Prometheus监控集群GPU资源

本文介绍通过阿里云ARMS Prometheus对GPU资源进行监控，查看GPU各项指标。

前提条件

您已完成以下操作：

- 创建托管GPU集群或创建专有GPU集群。
- 开通阿里云ARMS Prometheus服务。

安装阿里云ARMS Prometheus监控

在使用阿里云ARMS Prometheus监控对GPU进行监控前，您需要安装阿里云ARMS Prometheus监控。

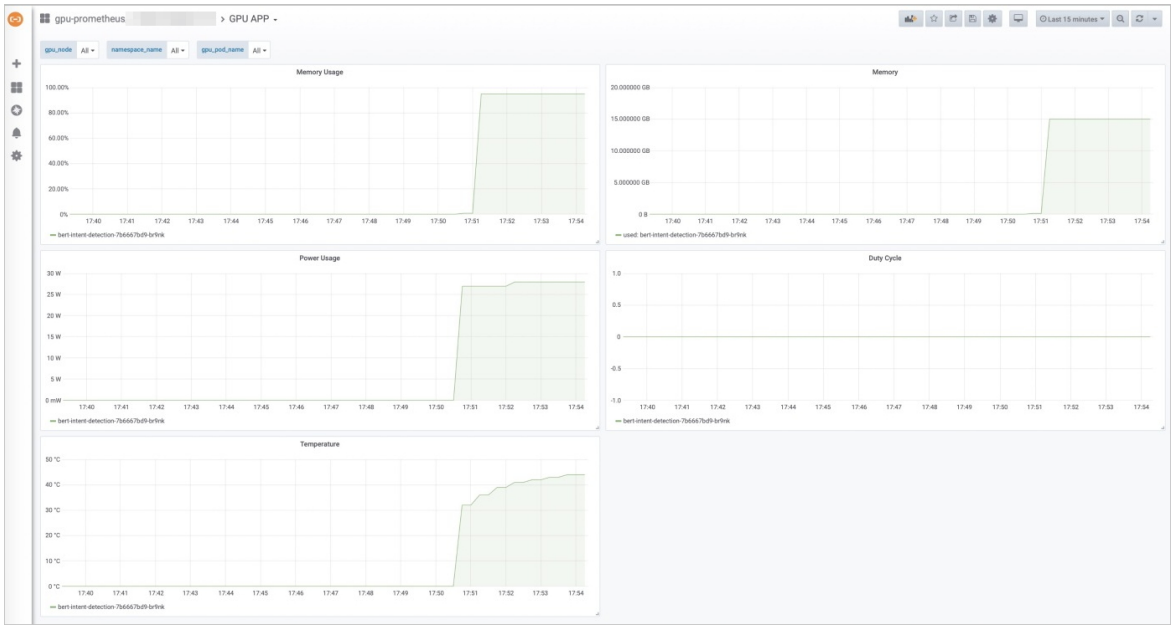
您有三种安装阿里云ARMS Prometheus监控的方式。有关具体操作，请参见阿里云ARMS Prometheus监控。

使用阿里云ARMS Prometheus进行GPU监控

1. 登录容器服务管理控制台。
2. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
3. 在集群管理页左侧导航栏中，选择**运维管理 > Prometheus监控**。
4. 在**Prometheus监控**大盘列表页面，单击**GPU APP**和**GPU Node**页签，您分别可以看到**GPU APP**和**GPU Node**两个监控大盘。
   - **GPU APP**用于监控Pod的GPU使用情况。
   - **GPU Node**用于监控集群节点的GPU使用情况。
5. 使用以下YAML文件在GPU节点上部署一个服务，测试监控效果。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: bert-intent-detection
spec:
 replicas: 1
 template:
  metadata:
   labels:
    app: bert-intent-detection
  spec:
   containers:
   - name: bert-container
    image: registry.cn-beijing.aliyuncs.com/ai-samples/bert-intent-detection:1.0.1
    ports:
    - containerPort: 80
    resources:
     limits:
      nvidia.com/gpu: 1
---
apiVersion: v1
kind: Service
metadata:
 labels:
  run: bert-intent-detection
 name: bert-intent-detection-svc
spec:
 ports:
 - port: 8500
  targetPort: 80
 selector:
  app: bert-intent-detection
 type: LoadBalancer
```

6. 在**Prometheus监控**大盘列表页面，单击**GPU APP**页签。
   在**GPU APP**监控页面，您可以看到GPU显存、使用率、电量、稳定性几项指标，以及部署在GPU节点上的应用。



7. 压测部署在GPU节点上的应用，查看监控状态的变化。

   i. 执行以下命令查看推理服务并获取IP地址。

   ```
   kubectl get svc bert-intent-detection-svc
   ```

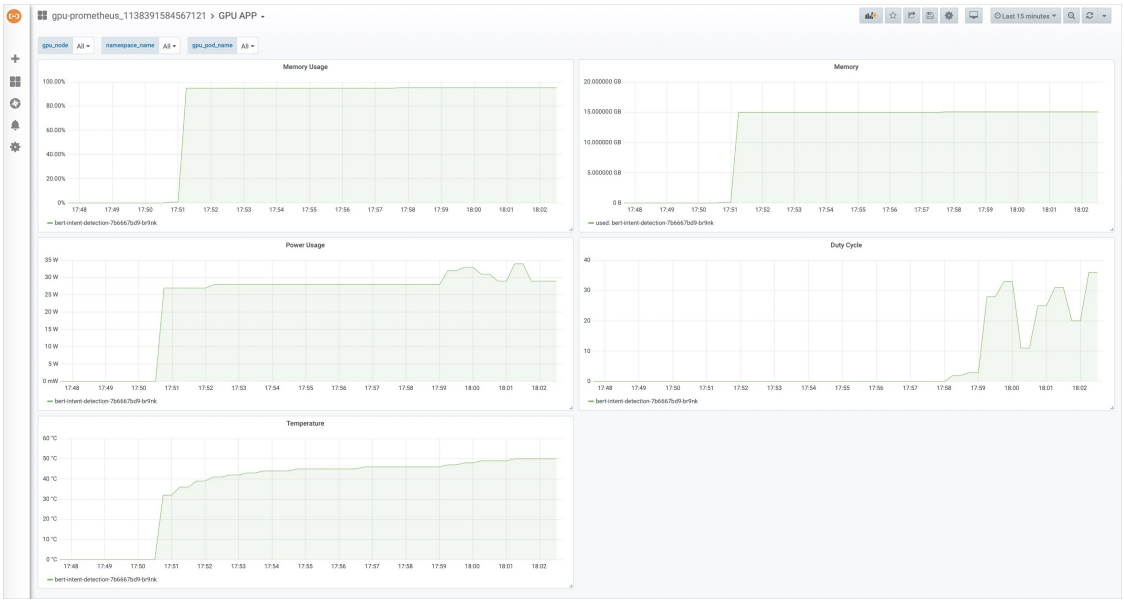
   预期输出：

   ```
   NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
   bert-intent-detection-svc  LoadBalancer  172.23.5.253  123.56.xx.xx  8500:32451/TCP  14m
   ```

   ii. 执行以下命令进行压测。

   ```
   hey -z 10m -c 100 "http://123.56.xx.xx:8500/predict?query=music"
   ```

   下图可以看出压测时，GPU利用率有了明显的变化。



18.5. GPU弹性伸缩

18.5.1. 基于GPU指标实现弹性伸缩

Kubernetes提供了External Metrics机制，该机制可以对接阿里云Prometheus监控来采集GPU指标。本文介绍如何部署阿里云Prometheus监控，然后举例说明如何通过阿里云Prometheus监控观测GPU指标，实现容器的弹性伸缩。

前提条件

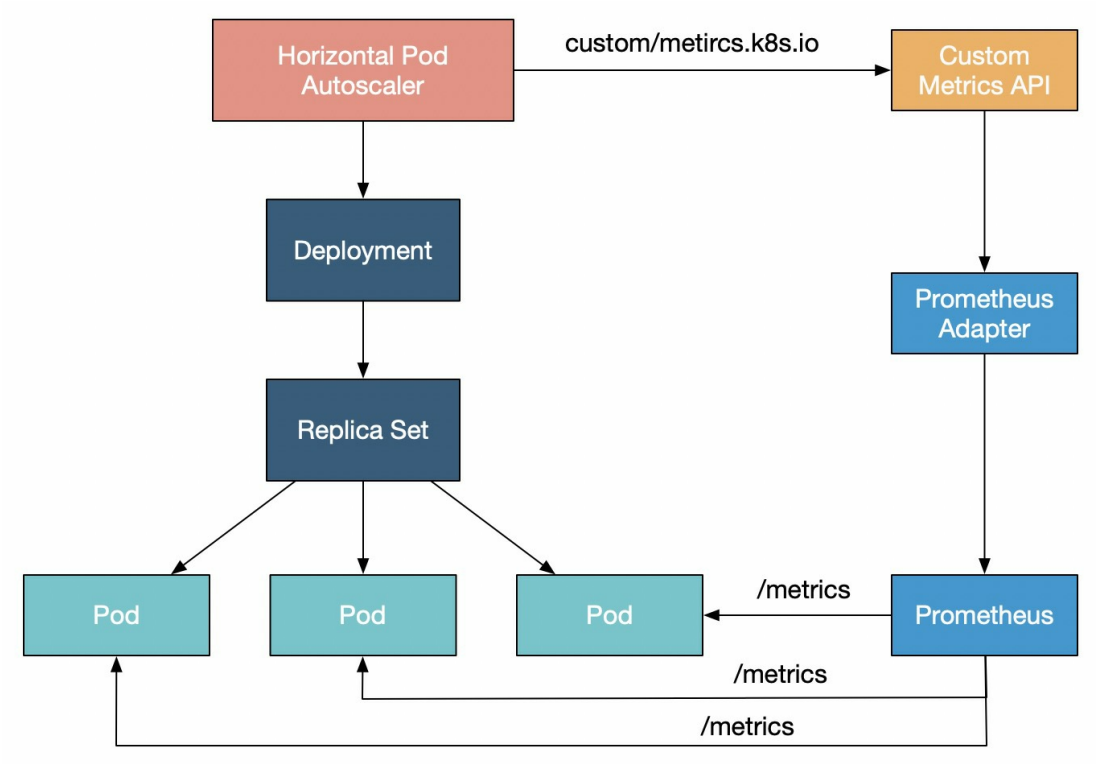
您已完成创建以下任一类型集群：

- 创建托管GPU集群
- 创建专有GPU集群

背景信息

在高性能计算领域，比如深度学习模型训练、推理等场景，通常需要使用GPU来做计算加速。为了节省成本，您往往需要根据GPU指标（利用率、显存）来做弹性伸缩。虽然默认的HPA（Horizontal Pod Autoscaler）组件并不支持根据GPU指标实现弹性伸缩，但Kubernetes提供的External Metrics机制可以通过GPU指标实现容器弹性伸缩。

GPU弹性伸缩原理图



步骤一：部署阿里云Prometheus

1. 安装阿里云Prometheus监控。

   您有三种安装阿里云Prometheus监控的方式。有关具体操作，请参见开启阿里云Prometheus监控。

2. 配置ack-alibaba-cloud-metrics-adapter。

   i. 登录阿里云Prometheus控制台。

   ii. 在控制台左侧导航栏中，单击**监控列表**，然后在**Prometheus监控**页面单击目标集群名称。

   iii. 在左侧导航栏，单击**设置**。

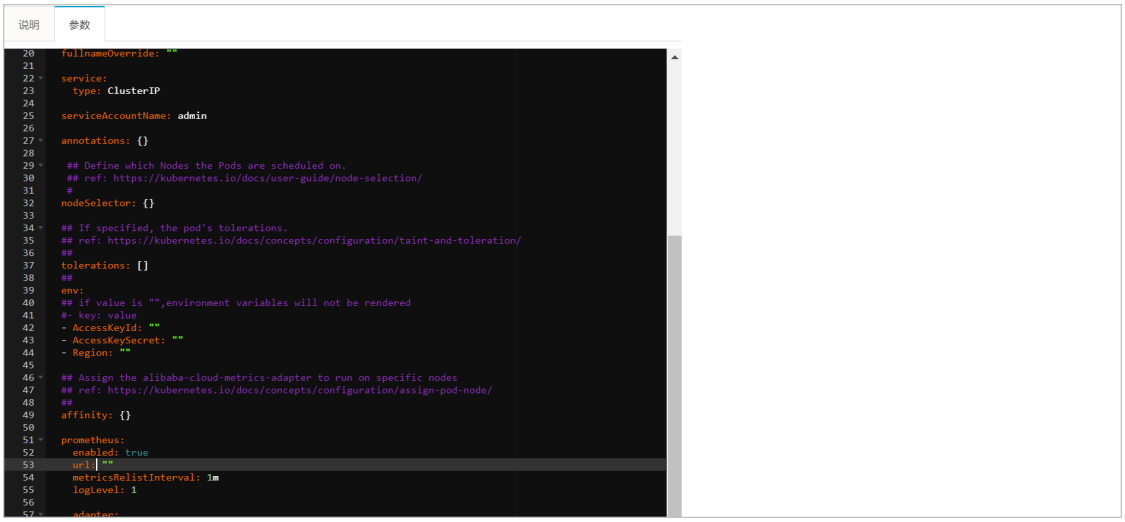
iv. 单击**Agent设置**页签，在**健康检查结果**区域中找到**步骤2：api接口地址**。



v. 在容器服务管理控制台左侧导航栏中，单击**市场 > 应用目录**。

vi. 在**应用目录**页面，找到并单击**ack-alibaba-cloud-metrics-adapter**。

vii. 在**应用目录- ack-alibaba-cloud-metrics-adapter**页面，单击**参数**页签，然后将上述获取的api接口地址设置为**url**的值。



viii. 在**创建**面板选择目标集群，单击**创建**。

步骤二：基于GPU指标实现弹性伸缩

本文通过在GPU上部署一个模型推理服务，然后对其进行压测。根据GPU利用率测试弹性伸缩。

1. 部署推理服务。

i. 使用以下YAML文件内容部署推理服务。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: bert-intent-detection
spec:
 replicas: 1
 selector:
  matchLabels:
   app: bert-intent-detection
 template:
  metadata:
   labels:
    app: bert-intent-detection
  spec:
   containers:
   - name: bert-container
    image: registry.cn-hangzhou.aliyuncs.com/ai-samples/bert-intent-detection:1.0.1
    ports:
    - containerPort: 80
    resources:
     limits:
      nvidia.com/gpu: 1
```

ii. 执行以下命令查看Pod和Service。

查看Pod命令：

```
kubectl get pods -o wide
```

预期输出：

```
NAME                    READY  STATUS   RESTARTS  AGE   IP       NODE           NOMINATED NODE  REA
DINESS GATES
bert-intent-detection-7b6667bd9-kgx2g  1/1   Running  0     8m58s  172.20.0.207  cn-beijing.192.168.0.1
14  <none>      <none>
```

上述输出可以看到当前只有一个Pod部署在192.168.0.115这个GPU节点上。
查看Service命令：

```
kubectl get svc bert-intent-detection-svc
```

预期输出：

```
NAME             TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
bert-intent-detection-svc  LoadBalancer  172.21.5.80  101.200.xx.xx 8500:32599/TCP  9m16s
```

iii. 通过SSH登录节点192.168.0.114后，执行以下命令查看GPU使用情况。

```
nvidia-smi
```

预期输出：

```
Thu Oct 15 15:41:50 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 T4       On   | 00000000:00:07.0 Off |                    0 |
| N/A   45C    P0    28W /  70W |  14315MiB / 15079MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0   1800353      C   python                                  14305MiB |
+-----------------------------------------------------------------------------+
```

推理服务进程已经运行在GPU上，当前GPU利用率为0。

iv. 执行以下命令调用推理服务，测试是否部署成功。

```
curl -v "http://101.200.xx.xx:8500/predict?query=music"
```

预期输出：

```
*   Trying 101.200.xx.xx...
* TCP_NODELAY set
* Connected to 101.200.xx.xx (101.200.xx.xx) port 8500 (#0)
> GET /predict?query=music HTTP/1.1
> Host: 101.200.xx.xx:8500
> User-Agent: curl/7.64.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 9
< Server: Werkzeug/1.0.1 Python/3.6.9
< Date: Thu, 15 Oct 2020 07:42:43 GMT
<
* Closing connection 0
PlayMusic #意图识别结果
```

当HTTP请求返回状态码200时，说明成功部署推理服务。

2. 配置HPA。

i. 使用以下YAML信息部署HPA。

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: gpu-hpa
spec:
 scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: bert-intent-detection
 minReplicas: 1
 maxReplicas: 10
 metrics:
 - type: Pods
  pods:
   metricName: duty_cycle_current # 指标为Pod的平均GPU使用率。
   targetAverageValue: 20
```

ii. 执行以下命令查看HPA。

```
kubectl get hpa
```

预期输出：

```
NAME    REFERENCE            TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
gpu-hpa  Deployment/bert-intent-detection  0/20   1     10     1     11m
```

从TARGETS中可以看到，当前GPU利用率是0，当利用率为20时触发弹性伸缩。

3. 执行以下命令进行压测。

```
hey -n 10000 -c 100 "http://101.200.88.51:8500/predict?query=music"
```

i. 压测过程中，执行以下命令观察HPA的状态。

```
kubectl get hpa
```

预期输出：

```
NAME    REFERENCE            TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
gpu-hpa  Deployment/bert-intent-detection  22/20   1     10     1     13m
```

ii. 当TARGETS中当前GPU利用率超过20时，触发弹性伸缩，此时ACK集群开始扩容。执行以下命令查看伸缩情况。

```
kubectl get pods
```

预期输出：

```
NAME                READY  STATUS       RESTARTS  AGE
bert-intent-detection-7b6667bd9-kgx2g  1/1   Running       0     13m
bert-intent-detection-7b6667bd9-zrvvr  0/1   ContainerCreating  0     21s
```

> **说明** 当压测停止，GPU利用率降低且低于20后，就会进行缩容。

常见问题

问：如果运行 `kubectl get hpa` 后，发现target一栏为UNKNOWN怎么办？

答：请确定HorizontalPodAutoscaler中指标名字正确。如果正确可以执行 `kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1"` 查看返回结果中有没有对应的指标。

问：目前GPU HPA都支持哪些指标？

答：GPU HPA支持的指标如下表。

指标名称	说明
duty_cycle_current	GPU利用率
memory_used_bytes_current	显存利用率
temperature_celsius_current	温度
power_usage_milliwatts_current	电量

18.6. GPU运维管理

18.6.1. 升级GPU节点的Docker

共享GPU的隔离能力依赖Docker 19.03.5以及与其对应的Nvidia-container-runtime版本，如果Kubernetes集群节点安装的Docker版本低于19.03.5，您需要将其升级至19.03.5。本文介绍如何升级Docker以及与其对应的Nvidia-container-runtime，从而使节点支持共享GPU。

背景信息

Nvidia-container-runtime允许用户构建和运行GPU加速的Docker容器，能够自动对容器进行配置，以达到容器使用Nvidia GPU的目的。

操作步骤

> **注意** 本文操作步骤仅适用于CentOS和Alibaba Cloud Linux 2操作系统。

在执行以下操作前，您需要使用命令行工具连接您的Kubernetes集群。详情请参见通过kubectl工具连接集群。

1. 在Master节点执行以下命令，下线节点。

   为避免在Docker升级期间，有Pod调度到该节点，您需要将节点标记为不可调度。

   ```
   kubectl cordon <NODE_NAME>
   ```

   > **说明** <NODE_NAME>为待升级Docker版本的节点名称。执行命令kubectl get nodes查询节点名称。

2. 在Master节点上执行以下命令，迁移节点中的Pod。

   节点下线以后，需要将该节点上的Pod迁移至其他节点。

   ```
   kubectl drain <NODE_NAME> --ignore-daemonsets --delete-local-data --force
   ```

   > **说明** <NODE_NAME>为待升级Docker版本的节点。

3. 执行以下命令，暂停kubelet和Docker服务。

   在待升级Docker版本的节点上停止kubelet和Docker服务。

```
service kubelet stop
docker rm -f $(docker ps -aq)
service docker stop
```

4. 执行以下命令，卸载Docker和Nvidia-container-runtime。

   在待升级Docker版本的节点上卸载旧版Docker和Nvidia-container-runtime。

```
yum remove -y docker-ce docker-ce-cli containerd
yum remove -y nvidia-container-runtime*  libnvidia-container*
```

5. 执行以下命令，备份并移除*daemon.json*文件。

```
cat /etc/docker/daemon.json
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  },
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "10"
  },
  "bip": "169.254.123.1/24",
  "oom-score-adjust": -1000,
  "storage-driver": "overlay2",
  "storage-opts":["overlay2.override_kernel_check=true"],
  "live-restore": true
}
mv /etc/docker/daemon.json /tmp
```

6. 执行以下命令，安装Docker。

   在待升级Docker版本的节点上下载Docker安装包。

```
VERSION=19.03.5
URL=http://aliacs-k8s-cn-beijing.oss-cn-beijing.aliyuncs.com/public/pkg/docker/docker-${VERSION}.tar.gz
curl -ssL $URL -o /tmp/docker-${VERSION}.tar.gz
cd /tmp
tar -xf docker-${VERSION}.tar.gz
cd /tmp/pkg/docker/${VERSION}/rpm
yum localinstall -y $(ls .)
```

7. 执行以下命令，在节点上安装Nvidia-container-runtime。

```
cd /tmp
yum install -y unzip
wget http://kubeflow.oss-cn-beijing.aliyuncs.com/nvidia.zip
unzip nvidia.zip
yum -y -q --nogpgcheck localinstall /tmp/nvidia/*
```

8. 执行以下命令，配置*daemon.json*。

   将上述的*daemon.json*覆盖*/etc/docker/daemon.json*，使原有配置生效。

```
mv /tmp/daemon.json  /etc/docker/daemon.json
cat /etc/docker/daemon.json
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  },
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "10"
  },
  "bip": "169.254.123.1/24",
  "oom-score-adjust": -1000,
  "storage-driver": "overlay2",
  "storage-opts":["overlay2.override_kernel_check=true"],
  "live-restore": true
}
```

9. 执行以下命令，重启Docker和kubelet服务。

```
service docker start
service kubelet start
```

10. 执行以下命令，上线节点。

    节点Docker升级完成后，使节点在集群中的状态变成可调度。

```
kubectl uncordon <NODE_NAME>
```

> **说明** <NODE_NAME>为升级完Docker版本的节点名称。

11. 执行以下命令，在该GPU节点重新启动GPU安装程序。

```
docker ps |grep cgpu-installer | awk '{print $1}' | xargs docker rm -f
```

相关文档

- 共享GPU调度概述
- 安装共享GPU组件

18.6.2. 升级节点cGPU版本

ACK集群支持GPU共享调度时节点需要安装cGPU模块。本文介绍如何通过命令及控制台升级节点上的cGPU模块。

前提条件

- 您已通过kubectl连接集群。具体操作，请参见通过kubectl连接Kubernetes集群。
- 集群已安装ack-cgpu组件。具体操作，请参见安装ack-cgpu组件。
- 升级节点上无正在运行的业务。

通过命令升级节点cGPU版本

cgpu-installer的DaemonSet用于在节点上安装cGPU驱动模块。在升级cGPU时，需要将cgpu-installer的镜像版本修改为待升级的版本。

目前支持的cGPU镜像版本如下：

- v0.8.10
- v0.8.12
- v0.8.13

> **说明** 升级节点cGPU版本过程中会重启节点，所以在升级节点cGPU版本之前，请确认节点上无正在运行的业务。

1. 执行以下命令修改cgpu-installer的镜像，升级节点的cGPU模块版本。

   ```
   kubectl edit ds cgpu-installer -n kube-system
   ```

   修改镜像的版本，以其他版本升级到v0.8.10版本为例：

   ```
   spec:
     containers:
     - env:
       - name: CHECK_REGIONS
         value: "true"
       - name: SUPPORT_REGIONS
         value: cn-hangzhou,cn-beijing,cn-shanghai,cn-zhangjiakou,cn-shenzhen,cn-chengdu,us-east-1,cn-heyuan,ap-southeast-1,ap-south

       image: registry-vpc.cn-beijing.aliyuncs.com/acs/cgpu-installer:v0.8.10
       imagePullPolicy: IfNotPresent
       name: cgpu-installer
       resources: {}
       securityContext:
   ```

2. 卸载节点上旧版本cGPU。
   i. 登录节点。关于登录方式，请参见通过密码认证登录Linux实例。
   ii. 执行以下命令停止Docker服务。

   ```
   systemctl stop docker
   ```

   iii. 执行以下命令卸载cGPU。

   ```
   bash /usr/local/cgpu-installer/uninstall.sh
   ```

   > **说明**
   > 若*/usr/local/cgpu-installer/uninstall.sh*不存在，请执行以下命令卸载旧版本cGPU。
   >
   > ```
   > wget http://aliacs-k8s-cn-beijing.oss-cn-beijing.aliyuncs.com/gpushare/cgpu-uninstall.sh -O /usr/local/cgpu-installer/uninstall.sh
   > ```

3. 重启卸载旧版本cGPU的节点。具体操作，请参见重启实例。

结果验证

当节点重启后，登录该节点，执行以下命令查询cGPU版本。

```
cat /proc/cgpu_km/version
```

预期输出：

```
0.8.10
```

从预期输出可以得出，节点的cGPU版本已经成功升级到v0.8.10版本。

通过控制台升级节点cGPU版本

cgpu-installer的DaemonSet用于在节点上安装cGPU驱动模块。在升级cGPU时，需要将cgpu-installer的镜像版本修改为待升级的版本。

目前支持的cGPU镜像版本如下：

- v0.8.10
- v0.8.12
- v0.8.13

1. 执行以下命令修改cgpu-installer的镜像，升级节点的cGPU模块版本。

   ```
   kubectl edit ds cgpu-installer -n kube-system
   ```

   修改镜像的版本，以其他版本升级到v0.8.10版本为例：

   ```
   spec:
     containers:
     - env:
       - name: CHECK_REGIONS
         value: "true"
       - name: SUPPORT_REGIONS
         value: cn-hangzhou,cn-beijing,cn-shanghai,cn-zhangjiakou,cn-shenzhen,cn-chengdu,us-east-1,cn-heyuan,ap-southeast-1,ap-south

       image: registry-vpc.cn-beijing.aliyuncs.com/acs/cgpu-installer:v0.8.10
       imagePullPolicy: IfNotPresent
       name: cgpu-installer
       resources: {}
       securityContext:
   ```

2. 移除待升级cGPU版本的节点。

   具体操作，请参见移除节点。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
   v. 在**节点**页面选中待升级节点，单击**批量移除**。
   vi. 在**移除节点**对话框选中**自动排空节点（drain）**。
   vii. 单击**确定**。

3. 创建节点池。

   创建一个新的节点池用于添加上面移除的安装有cGPU模块的节点。
   具体操作，请参见管理节点池。

   i. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
   ii. 在**节点池**页面右上角，单击**创建节点池**。
   iii. 在**创建节点池**页面，设置创建节点池的配置项。

   有关配置项的详细说明，请参见创建Kubernetes专有版集群。部分配置项说明如下：

参数	描述	配置值
数量	设置节点池初始节点数量。	需要填写为0。
节点标签	您可以为集群节点添加标签。	■ 若集群中安装了ack-ai-installer组件，则**键**中输入*ack.node.gpu.schedule*，**值**中输入*cgpu*。 ■ 若集群中安装了ack-cgpu组件，则**键**中输入*cgpu*，**值**中输入*true*。

iv. 单击**确认配置**，完成节点池的创建。

4. 添加节点到创建的节点池。

   节点池创建完成后，需要将步骤2中移除的节点添加到步骤3创建的节点池。具体操作，请参见添加已有节点。

结果验证

当节点添加完成以后，验证节点的cGPU版本是否已升级。

1. 执行以下命令查询新添加节点所对应的cgpu-installer。

```
kubectl get po -l name=cgpu-installer -n kube-system -o wide
```

   预期输出：

```
NAME                READY  STATUS   RESTARTS  AGE   IP          NODE              NOMINATED NODE  READINESS GATE
S
cgpu-installer-kkmp6  1/1   Running  0     4d2h  192.168.XXX.XX1  cn-beijing.192.168.XXX.XX1  <none>      <n
one>
cgpu-installer-**2   1/1   Running  0     4d2h  192.168.XXX.XX2  cn-beijing.192.168.XXX.XX2  <none>      <non
e>
cgpu-installer-**3   1/1   Running  0     4d2h  192.168.XXX.XX3  cn-beijing.192.168.XXX.XX3  <none>      <non
e>
```

2. 执行以下命令进入 `cgpu-installer-kkmp6` 的Pod。

```
kubectl exec -ti cgpu-installer-kkmp6 -n kube-system -- bash
```

3. 执行以下命令查询cGPU当前版本。

```
nsenter -t 1 -i -p -n -u -m -- cat /proc/cgpu_km/version
```

   预期输出：

```
0.8.10
```

   从预期输出可以得出节点的cGPU版本已经为v0.8.10。

18.6.3. 排查GPU监控常见问题

当GPU监控大盘异常或无数据时，您可以按照本文描述的操作步骤排查GPU监控常见问题。

操作步骤

步骤一：查看集群中是否有GPU节点

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点**页面，查看目标集群中是否有GPU节点。

   > **说明** 在节点页面的配置列，如果配置名称包含****ecs.gn****，则说明该集群中有GPU节点。

步骤二：查看ack-arms-prometheus是否正确安装

1. 查看目标集群是否安装ack-arms-prometheus。具体操作，请参见开启阿里云Prometheus监控。
2. 如果已安装ack-arms-prometheus，执行以下命令查看ack-arms-prometheus的Pod状态。

```
kubectl get pods -n arms-prom
```

预期输出：

```
NAME                                        READY  STATUS   RESTARTS  AGE
arms-prom-ack-arms-prometheus-866cfd9f8f-x8jxl  1/1    Running  0     26d
```

如果是Running状态，说明Pod运行正常。如果Pod的状态不是Running，则执行 `kubectl describe pod` 命令，查看Pod状态不正常原因。

步骤三：检查ack-prometheus-gpu-exporter是否成功部署

执行以下命令，查看Pod的运行状态和数量。

```
kubectl get pods -n arms-prom
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE
ack-prometheus-gpu-exporter-6kpj7          1/1    Running  0     7d19h
ack-prometheus-gpu-exporter-bkbf8          1/1    Running  0     18h
ack-prometheus-gpu-exporter-blbnq          1/1    Running  0     18h
```

从上述输出信息，可以知道Pod数量和GPU节点数一致，且Pod的状态是Running，说明ack-prometheus-gpu-exporter已在相应GPU节点成功部署。如果Pod的状态不是Running，则执行 `kubectl describe pod` 命令，查看Pod状态不正常原因。

步骤四：检查ack-prometheus-gpu-exporter是否成功采集到数据

1. 执行以下命令，SSH登录到目标集群节点。

   ```
   ssh root@127.0.XX.XX
   ```

   - root：用户自定义用户名。
   - 127.0.XX.XX：目标集群的公网IP访问地址。

2. 执行以下命令，查看Pod的内网IP。

   ```
   kubectl get pods -n arms-prom -o wide
   ```

   预期输出：

   ```
   NAME                                READY  STATUS   RESTARTS  AGE   IP          NODE            NOMINATED NODE  RE
   ADINESS GATES
   ack-prometheus-gpu-exporter-4rdtl          1/1    Running  0     7h6m  172.21.XX.XX  cn-beijing.192.168.0.22
   <none>      <none>
   ack-prometheus-gpu-exporter-vdkqf          1/1    Running  0     6d16h  172.21.XX.XX  cn-beijing.192.168.94.
   7  <none>      <none>
   ack-prometheus-gpu-exporter-x7v48          1/1    Running  0     7h6m  172.21.XX.XX  cn-beijing.192.168.0.2
   3  <none>      <none>
   ```

3. 执行以下命令，调用 `gpu exporter` 服务，获取GPU指标信息。

   > **说明** ack-prometheus-gpu-exporter的默认端口是9445。

   ```
   curl 172.21.XX.XX:9445 | grep "nvidia_gpu"
   ```

   预期输出：

```
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  7518  100  7518    0     0   101k      0 --:--:-- --:--:-- --:--:--  101k
# HELP nvidia_gpu_duty_cycle Percent of time over the past sample period during which one or more kernels were executing on the GPU device
# TYPE nvidia_gpu_duty_cycle gauge
nvidia_gpu_duty_cycle{allocate_mode="exclusive",container_name="tfserving-gpu",minor_number="0",name="Tesla T4",namespace_name="default",node_name="cn-beijing.192.168.0.22",pod_name="fashion-mnist-eci-2-predictor-0-tfserving-proxy-tfserving-v789b",uuid="GPU-293f6608-281a-cc66-fcb3-0d366f32a31d"} 0
# HELP nvidia_gpu_memory_total_bytes Total memory of the GPU device
# TYPE nvidia_gpu_memory_total_bytes gauge
nvidia_gpu_memory_total_bytes{allocate_mode="exclusive",container_name="tfserving-gpu",minor_number="0",name="Tesla T4",namespace_name="default",node_name="cn-beijing.192.168.0.22",pod_name="fashion-mnist-eci-2-predictor-0-tfserving-proxy-tfserving-v789b",uuid="GPU-293f6608-281a-cc66-fcb3-0d366f32a31d"} 1.5811477504e+10
# HELP nvidia_gpu_memory_used_bytes Memory used by the GPU device
# TYPE nvidia_gpu_memory_used_bytes gauge
nvidia_gpu_memory_used_bytes{allocate_mode="exclusive",container_name="tfserving-gpu",minor_number="0",name="Tesla T4",namespace_name="default",node_name="cn-beijing.192.168.0.22",pod_name="fashion-mnist-eci-2-predictor-0-tfserving-proxy-tfserving-v789b",uuid="GPU-293f6608-281a-cc66-fcb3-0d366f32a31d"} 1.488453632e+10
# HELP nvidia_gpu_num_devices Number of GPU devices
# TYPE nvidia_gpu_num_devices gauge
nvidia_gpu_num_devices{node_name="cn-beijing.192.168.0.22"} 1
# HELP nvidia_gpu_power_usage_milliwatts Power usage of the GPU device in watts
# TYPE nvidia_gpu_power_usage_milliwatts gauge
nvidia_gpu_power_usage_milliwatts{allocate_mode="exclusive",container_name="tfserving-gpu",minor_number="0",name="Tesla T4",namespace_name="default",node_name="cn-beijing.192.168.0.22",pod_name="fashion-mnist-eci-2-predictor-0-tfserving-proxy-tfserving-v789b",uuid="GPU-293f6608-281a-cc66-fcb3-0d366f32a31d"} 27000
# HELP nvidia_gpu_temperature_celsius Temperature of the GPU device in celsius
# TYPE nvidia_gpu_temperature_celsius gauge
nvidia_gpu_temperature_celsius{allocate_mode="exclusive",container_name="tfserving-gpu",minor_number="0",name="Tesla T4",namespace_name="default",node_name="cn-beijing.192.168.0.22",pod_name="fashion-mnist-eci-2-predictor-0-tfserving-proxy-tfserving-v789b",uuid="GPU-293f6608-281a-cc66-fcb3-0d366f32a31d"} 44
```

如果输出的数据中有 `nvidia_gpu` 开头的指标信息，说明ack-prometheus-gpu-exporter可以成功采集数据。

相关文档

- 使用ARMS Prometheus监控集群GPU资源

18.6.4. 通过节点池创建新NVIDIA驱动版本的节点

ACK集群默认安装的NVIDIA驱动版本为418.87.01，如果您使用的CUDA库需要匹配更高版本的NVIDIA驱动，需要自定义安装节点的NVIDIA驱动。本文介绍在不使用现有节点的场景下，如何通过节点池创建新NVIDIA驱动版本的节点。

基于节点池安装节点NVIDIA驱动的优势

可以实现对集群中不同节点上的NVIDIA驱动的批量管理功能。以下是两个使用示例：

- 将集群中需要安装418.181.07版本的节点用节点池A管理，如果需要将任务调度到NVIDIA驱动版本为418.181.07的节点上，只需要将任务的selector指定为节点池A的标签即可。
- 将集群中的一部分节点安装418.181.07的驱动，而另一部分的节点安装版本为450.102.0的驱动。此时，可以将一部分节点添加到节点池A，另一部分节点添加到节点池B。

步骤一：确定NVIDIA驱动版本

您所使用的CUDA库与哪些NVIDIA驱动兼容是NVIDIA驱动版本选定的最重要依据。以下是CUDA库与NVIDIA驱动的兼容关系。

CUDA Toolkit	Linux x86_64 Driver Version
CUDA 11.1 (11.1.0)	>= 450.80.02
CUDA 11.0 (11.0.3)	>= 450.36.06
CUDA 10.2 (10.2.89)	>= 440.33
CUDA 10.1 (10.1.105)	>= 418.39
CUDA 10.0 (10.0.130)	>= 410.48
CUDA 9.2 (9.2.88)	>= 396.26
CUDA 9.1 (9.1.85)	>= 390.46
CUDA 9.0 (9.0.76)	>= 384.81
CUDA 8.0 (8.0.61 GA2)	>= 375.26
CUDA 8.0 (8.0.44)	>= 367.48
CUDA 7.5 (7.5.16)	>= 352.31
CUDA 7.0 (7.0.28)	>= 346.46

步骤二：创建节点池并指定驱动版本

方式一：从容器服务提供的可用驱动版本列表中选择驱动创建节点池

> 说明 选择此种方式安装驱动比较简单，只需要给节点池的节点打上标签 `ack.aliyun.com/nvidia-driver-version=<驱动版本>`，然后将前面步骤中集群移除的节点添加到该节点池即可。

本文以418.181.07版本为例介绍整个操作流程：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 单击右上角**创建节点池**。
6. 在**创建节点池**对话框中配置相关参数。有关配置项的详细说明，请参见创建Kubernetes专有版集群。部分配置项说明如下：
   i. 单击**显示高级选项**。

ii. 在**节点标签**参数栏添加标签，单击图标，在**键**文本框中输入 `ack.aliyun.com/nvidia-driver-version`，在**值**文本框中输入 `418.181.07`。

容器服务提供可用NVIDIA驱动版本如下：

- 418.181.07
- 450.102.04
- 460.32.03
- 470.57.02

iii. 参数配置完成后，单击**确认配置**。

方式二：自定义驱动版本创建节点池

自定义驱动版本

以下以上传*NVIDIA-Linux-x86_64-460.32.03.run*驱动文件为例介绍如何自定义驱动版本。驱动文件可以在NVIDIA官方网站下载。

> **说明** *NVIDIA-Linux-x86_64-460.32.03.run*需要放在OSS Bucket的根目录下。

1. 在对象存储OSS管理控制台中创建Bucket。具体操作，请参见创建存储空间。
2. 将*NVIDIA-Linux-x86_64-460.32.03.run*驱动文件上传至OSS。具体操作，请参见上传文件。
3. 文件上传成功后，在对象存储管理页左侧导航栏中，单击**文件管理**。
4. 在**文件管理**页面，单击已上传驱动文件右侧**操作**列下的**详情**。
5. 在**详情**面板，关闭**使用HTTPS**开关，取消使用HTTPS的配置。

   > **说明** ACK创建集群时会通过URL拉取驱动文件，URL中使用的协议是HTTP协议。OSS默认使用的是HTTPS协议，您需要关闭**使用HTTPS**开关，取消使用HTTPS的配置。

6. 确认驱动文件配置信息。记录该文件的URL，且将URL拆成两部分，即endpoint和runfile。以 `http://nvidia-XXX-XXX-cn-beijing.aliyuncs.com/NVIDIA-Linux-x86_64-460.32.03.run` 为例，将其分成以下两部分：
   - endpoint：nvidia-XXX-XXX-cn-beijing.aliyuncs.com
   - runfile：NVIDIA-Linux-x86_64-460.32.03.run

通过自定义驱动版本后创建节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 单击右上角**创建节点池**。
6. 在**创建节点池**对话框中配置相关参数。有关配置项的详细说明，请参见创建Kubernetes专有版集群。部分配置项说明如下：

   i. 单击**显示高级选项**。

ii. 在**节点标签**参数栏添加两个标签，单击  图标：

- 输入第一个标签的**键**为 `ack.aliyun.com/nvidia-driver-oss-endpoint` ，**值**为 `nvidia-XXX-XXX-cn-beijing.aliyuncs.com` 。
- 输入第二个标签的**键**为 `ack.aliyun.com/nvidia-driver-runfile` ，**值**为 `NVIDIA-Linux-x86_64-460.32.03.run` 。

iii. 参数配置完成后，单击**确认配置**。

方式三：在OpenAPI中使用指定版本创建节点池

通过在OpenAPI中调用CreateClusterNodePool，配置相关请求参数，可为集群创建指定NVIDIA驱动版本的节点池。具体操作，请参见创建节点池。

方式一：在OpenAPI中使用指定驱动版本创建节点池

如果使用OpenAPI方式创建或者扩容集群，那么只需要在上传的 `body` 中的 `tags` 域添加特定标签即可，代码示例如下所示。

```
{
 // 其他部分省略
 ......
  "tags": [
    {
      "key": "ack.aliyun.com/nvidia-driver-version",
      "value": "410.104"
    }
  ],
 // 其他部分省略
 ......
}
```

方式二：在OpenAPI中使用指定驱动版本的文件下载地址创建节点池

如果需要指定NVIDIA驱动下载地址，那么需要指定OSS产品 `endpoint` 和 `runfile` 名称，代码示例如下所示。关于如何获取OSS产品 `endpoint` 和 `runfile` 名称，请参见确认驱动文件配置信息。

```
{
 // 其他部分省略
 ......
  "tags": [
   {
       "key": "ack.aliyun.com/nvidia-driver-oss-endpoint",
       "value": "nvidia-XXX-XXX-cn-beijing.aliyuncs.com"
   },
   {
    "key": "ack.aliyun.com/nvidia-driver-runfile",
    "value": "NVIDIA-Linux-x86_64-410.104.run"
   }
  ],
 // 其他部分省略
 ......
}
```

步骤三：验证通过节点池自定义安装NVIDIA驱动是否成功

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，单击**集群**。
3. 在目标集群右侧**操作**列选择**更多 > 通过CloudShell管理集群**。
4. 执行以下命令，查看带有 `component: nvidia-device-plugin` 标签的Pod。

```
kubectl get po -n kube-system -l component=nvidia-device-plugin -o wide
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE  IP        NODE              NOMINATED NODE  READI
NESS GATES
nvidia-device-plugin-cn-beijing.192.168.1.127  1/1   Running  0      6d   192.168.1.127  cn-beijing.192.168.1.127
<none>       <none>
nvidia-device-plugin-cn-beijing.192.168.1.128  1/1   Running  0      17m  192.168.1.128  cn-beijing.192.168.1.12
8  <none>       <none>
nvidia-device-plugin-cn-beijing.192.168.8.12   1/1   Running  0      9d   192.168.8.12   cn-beijing.192.168.8.12
<none>       <none>
nvidia-device-plugin-cn-beijing.192.168.8.13   1/1   Running  0      9d   192.168.8.13   cn-beijing.192.168.8.13
<none>       <none>
nvidia-device-plugin-cn-beijing.192.168.8.14   1/1   Running  0      9d   192.168.8.14   cn-beijing.192.168.8.14
<none>       <none>
```

通过NODE列找到集群中刚添加节点所对应的Pod名称为 `nvidia-device-plugin-cn-beijing.192.168.1.128` 。

5. 执行以下命令查看节点的驱动版本是否符合预期。

```
kubectl exec -ti nvidia-device-plugin-cn-beijing.192.168.1.128 -n kube-system -- nvidia-smi
```

预期输出：

```
Sun Feb  7 04:09:01 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.181.07   Driver Version: 418.181.07   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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   27C    P0    40W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:08.0 Off |                    0 |
| N/A   27C    P0    40W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:09.0 Off |                    0 |
| N/A   31C    P0    39W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:0A.0 Off |                    0 |
| N/A   27C    P0    41W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

从以上输出的信息，可以知道驱动版本为418.181.07，通过节点池自定义安装NVIDIA驱动成功。

相关文档

- 通过节点池升级已有节点的NVIDIA驱动

18.6.5. 通过节点池升级已有节点的NVIDIA驱动

如果您使用的CUDA库需要匹配更高版本的NVIDIA驱动，需要升级节点的NVIDIA驱动。本文升级方案是将节点从集群中移除，然后再将该节点添加到集群中，在添加的过程中节点会重装系统、安装指定版本的NVIDIA驱动。本文介绍在使用现有节点的场景下，如何通过节点池升级节点NVIDIA驱动。

背景信息

ACK目前不支持在不移除节点的情况下仅升级节点NVIDIA驱动的场景，且您原有节点池也许还存在其它节点，故不做升级原节点池的操作。

基于节点池升级节点NVIDIA驱动的优势

实现了对集群中不同节点上的NVIDIA驱动的批量管理功能。以下是两个使用示例：

- 将集群中的一部分节点的驱动升级到418.181.07版本，而另一部分的节点升级到450.102.0版本。此时，可以将一部分节点添加到节点池A，另一部分节点添加到节点池B。
- 将集群中需要升级到418.181.07版本的节点用节点池A管理，如果需要将任务调度到NVIDIA驱动版本为418.181.07的节点上，只需要将任务的selector指定为节点池A的标签即可。

> **说明**
>
> - 升级NVIDIA驱动的方式本质上是将节点从集群中移除，然后再将该节点添加到集群中，在添加的过程中会重装系统、安装指定NVIDIA驱动等，所以在执行升级操作前，确认该节点没有正在运行的任务以及重要的数据。
> - 为了降低风险，强烈建议您先升级一个节点，如果整个升级过程没有问题，再进行批量升级操作。

步骤一：确定NVIDIA驱动版本

您所使用的CUDA库与哪些NVIDIA驱动版本兼容是NVIDIA驱动版本选定的最重要依据。以下是CUDA库与NVIDIA驱动的兼容关系。

> **说明** ACK集群默认安装的NVIDIA驱动版本为418.87.01。

CUDA Toolkit	Linux x86_64 Driver Version
CUDA 11.1 (11.1.0)	>= 450.80.02
CUDA 11.0 (11.0.3)	>= 450.36.06
CUDA 10.2 (10.2.89)	>= 440.33
CUDA 10.1 (10.1.105)	>= 418.39
CUDA 10.0 (10.0.130)	>= 410.48
CUDA 9.2 (9.2.88)	>= 396.26
CUDA 9.1 (9.1.85)	>= 390.46
CUDA 9.0 (9.0.76)	>= 384.81
CUDA 8.0 (8.0.61 GA2)	>= 375.26
CUDA 8.0 (8.0.44)	>= 367.48
CUDA 7.5 (7.5.16)	>= 352.31
CUDA 7.0 (7.0.28)	>= 346.46

更多信息，请参见cuda-compatibility。

步骤二：移除节点

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 选中待升级驱动的节点，单击**批量移除**。
6. 在**移除节点**对话框，选中**自动排空节点（drain）**，然后单击**确定**。

步骤三：创建节点池并指定驱动版本

方式一：从容器服务提供的可用驱动版本列表中选择驱动创建节点池

> **说明** 选择此种方式安装驱动比较简单，只需要给节点池的节点打上标签 `ack.aliyun.com/nvidia-driver-version=<驱动版本>`，然后将前面步骤中集群移除的节点添加到该节点池即可。

本文以418.181.07版本为例介绍整个操作流程：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 单击右上角**创建节点池**。
6. 在**创建节点池**对话框中配置相关参数。有关配置项的详细说明，请参见创建Kubernetes专有版集群。部分配置项说明如下：

参数	描述
虚拟交换机	虚拟交换机和实例规格参数的取值任意，因为这些信息在扩容节点时才会用到，升级驱动属于添加已有节点，不会用到这些信息。
实例规格	
操作系统	节点重新加入集群后所使用的操作系统。
数量	实例数量填写为0，如果实例数量不为0，ACK将会创建新的实例。 

   i. 单击**显示高级选项**。

   ii. 在**节点标签**参数栏添加标签，单击图标，在**键**文本框中输入 `ack.aliyun.com/nvidia-driver-version`，在**值**文本框中输入 `418.181.07`。

   容器服务提供可用NVIDIA驱动版本如下：

   - 418.181.07
   - 450.102.04
   - 460.32.03

   iii. 参数配置完成后，单击**确认配置**。

方式二：自定义驱动版本创建节点池

自定义驱动版本

以下以上传*NVIDIA-Linux-x86_64-460.32.03.run*驱动文件为例介绍如何自定义驱动版本。驱动文件可以在NVIDIA官方网站下载。

> **说明** *NVIDIA-Linux-x86_64-460.32.03.run*需要放在OSS Bucket的根目录下。

1. 在对象存储OSS管理控制台中创建Bucket。具体操作，请参见创建存储空间。
2. 将*NVIDIA-Linux-x86_64-460.32.03.run*驱动文件上传至OSS。具体操作，请参见上传文件。
3. 文件上传成功后，在对象存储管理页左侧导航栏中，单击**文件管理**。
4. 在**文件管理**页面，单击已上传驱动文件右侧**操作**列下的**详情**。

5. 在**详情**面板，关闭**使用HTTPS**开关，取消使用HTTPS的配置。

   > **说明** ACK创建集群时会通过URL拉取驱动文件，URL中使用的协议是HTTP协议。OSS默认使用的是HTTPS协议，您需要单击**使用HTTPS**按钮，取消使用HTTPS的配置。

6. 确认驱动文件配置信息。记录该文件的URL，且将URL拆成两部分，即endpoint和runfile。以 `http://nvidia-XXX-XXX-cn-beijing.aliyuncs.com/NVIDIA-Linux-x86_64-460.32.03.run` 为例，将其分成以下两部分：
   - endpoint：nvidia-XXX-XXX-cn-beijing.aliyuncs.com
   - runfile：NVIDIA-Linux-x86_64-460.32.03.run

通过自定义驱动版本后创建节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 单击右上角**创建节点池**。
6. 在**创建节点池**对话框中配置相关参数。有关配置项的详细说明，请参见创建Kubernetes专有版集群。部分配置项说明如下：

参数	描述
虚拟交换机	虚拟交换机和实例规格参数的取值任意，因为这些信息在扩容节点时才会用到，升级驱动属于添加已有节点，不会用到这些信息。
实例规格	
操作系统	节点重新加入集群后所使用的操作系统。
数量	实例数量填写为0，如果实例数量不为0，ACK将会创建新的实例。 

   i. 单击**显示高级选项**。

   ii. 在**节点标签**参数栏添加两个标签，单击 图标：

   - 输入第一个标签的**键**为 `ack.aliyun.com/nvidia-driver-oss-endpoint`，**值**为 `nvidia-XXX-XXX-cn-beijing.aliyuncs.com`。
   - 输入第二个标签的**键**为 `ack.aliyun.com/nvidia-driver-runfile`，**值**为 `NVIDIA-Linux-x86_64-460.32.03.run`。

   iii. 参数配置完成后，单击**确认配置**。

步骤四：添加节点到节点池

节点池创建完成以后，需要将先前移除的节点添加到该节点池。具体操作，请参见添加已有节点。

步骤五：验证通过节点池升级节点NVIDIA驱动是否成功

1. 在目标集群右侧**操作**列选择**更多 > 通过CloudShell管理集群**。
2. 执行以下命令，查看带有 `component: nvidia-device-plugin` 标签的Pod。

```
kubectl get po -n kube-system -l component=nvidia-device-plugin -o wide
```

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE  IP        NODE              NOMINATED NODE  READI
NESS GATES
nvidia-device-plugin-cn-beijing.192.168.1.127  1/1   Running  0      6d   192.168.1.127  cn-beijing.192.168.1.127
<none>       <none>
nvidia-device-plugin-cn-beijing.192.168.1.128  1/1   Running  0      17m  192.168.1.128  cn-beijing.192.168.1.12
8  <none>       <none>
nvidia-device-plugin-cn-beijing.192.168.8.12   1/1   Running  0      9d   192.168.8.12   cn-beijing.192.168.8.12
<none>       <none>
nvidia-device-plugin-cn-beijing.192.168.8.13   1/1   Running  0      9d   192.168.8.13   cn-beijing.192.168.8.13
<none>       <none>
nvidia-device-plugin-cn-beijing.192.168.8.14   1/1   Running  0      9d   192.168.8.14   cn-beijing.192.168.8.14
<none>       <none>
```

通过NODE列找到集群中刚添加节点所对应的Pod名称为 `nvidia-device-plugin-cn-beijing.192.168.1.128` 。

3. 执行以下命令查看节点的驱动版本是否符合预期。

```
kubectl exec -ti nvidia-device-plugin-cn-beijing.192.168.1.128 -n kube-system -- nvidia-smi
```

预期输出：

```
Sun Feb  7 04:09:01 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.181.07   Driver Version: 418.181.07   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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   27C    P0    40W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:08.0 Off |                    0 |
| N/A   27C    P0    40W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:09.0 Off |                    0 |
| N/A   31C    P0    39W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:0A.0 Off |                    0 |
| N/A   27C    P0    41W / 300W |     0MiB / 16130MiB |     0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

从以上输出的信息，可以知道驱动版本为418.181.07，通过节点池升级节点NVIDIA驱动成功。

相关文档

- 通过节点池创建新NVIDIA驱动版本的节点

18.6.6. 通过节点池开启NVIDIA A100多实例GPU能力

NVIDIA A100能够安全地划分为多达七个独立的GPU实例，能够为多个用户提供单独的GPU资源，进而提升GPU利用率。本文介绍如何通过节点池开启NVIDIA A100的多实例MIG（Multi-Instance GPU）能力。

前提条件

- 已创建异构计算集群，且集群版本不低于1.20.4。具体操作，请参见创建异构计算集群。

> **注意** 目前只有弹性裸金属服务器A100机型（即名称首部为ecs.ebmgn7.的机型）支持MIG功能，直通ECS A100机型（即名称首部为ecs.gn7.的机型）并不支持开启MIG能力。关于弹性裸金属服务器的更多信息，请参见弹性裸金属服务器概述。

- 已通过Kubectl工具连接异构计算集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

GTC 2020，NVIDIA正式发布了搭载全新一代Amepere架构的Tesla A100，Ampere架构继承了前两代Volta和Turing架构的诸多优点，并增加了更多新的设计亮点，其中的一个重要特性是支持MIG。

NVIDIA A100由7个计算单元和8个内存单元组成，可以划分为多个大小相同（或不同）的 GPU 实例，每个GPU实例包含一定的计算单元和内存单元。

> **说明** 内存指显存，每个内存单元为5 GB。

关于NVIDIA A100多实例GPU特性的更多信息，请参见NVIDIA A100多实例GPU用户指南。
GPU实例规格的使用语法为：`[compute]g.[memory]gb` 。例如，一个拥有1个计算单元和一个内存单元的GPU实例规格为：1g.5gb。在一张NVIDIA A100上可划分的MIG实例规格和数量如下所示：

实例规格	实例规格ID	单张A100最多划分实例数	每个实例的计算单元个数	每个实例内存单元个数
1g.5gb	19	7	1	1（5 GB）
2g.10gb	14	3	2	2（10 GB）
3g.20gb	9	2	3	4（20 GB）
4g.20gb	5	1	4	4（20 GB）
7g.40gb	0	1	7	8（40 GB）

使用节点池标签划分MIG

在创建节点池时，通过给节点池添加特定的标签，达到开启节点池内的节点的MIG能力。可通过以下两种方式指定划分MIG实例：

- 指定单个MIG实例名称或ID
  该方式可在一张NVIDIA A100上划分多个相同的MIG实例规格，设置简单、便于理解。
- 指定MIG实例规格序列
  该方式可在一张NVIDIA A100上自定义划分多种MIG实例规格，设置灵活。

方式一：指定单个MIG实例名称或ID

- 节点池标签：ack.aliyun.com/gpu-partition-size 。
- 标签值：单个MIG实例名称或ID（例如：1g.5gb或19）。
- 示例：ack.aliyun.com/gpu-partition-size=1g.5gb 或 ack.aliyun.com/gpu-partition-size=19 。
- 含义：对每个NVIDIA A100按照指定的规格尽可能多的划分MIG。从上表中可以看到一张NVIDIA A100上最多可以划分7个规格为1g.5gb的MIG。

  容器服务控制台节点标签配置示例



方式二：指定MIG实例规格序列

- 节点池标签：ack.aliyun.com/gpu-partition-sequence
- 标签值：规格或ID序列（例如，1g.5gb-2g.10gb-3g.20gb或19-14-9），不过只有一些特定序列可用。A100目前支持18种序列，更多信息，请参见NVIDIA A100多实例GPU用户指南中的“A100 Supported Profiles”。
  ACK目前仅开放如下4种序列：

索引号序列	规格序列
19-19-19-19-19-19-19	1g.5gb-1g.5gb-1g.5gb-1g.5gb-1g.5gb-1g.5gb-1g.5gb
14-14-14-19	2g.10gb-2g.10gb-2g.10gb-1g.5gb
9-9	3g.20gb-3g.20gb
5-19-19-19	4g.20gb-1g.5gb-1g.5gb-1g.5gb

- 示例：ack.aliyun.com/gpu-partition-sequence=3g.20gb-3g.20gb 或 ack.aliyun.com/gpu-partition-sequence=9-9
- 含义：对每个NVIDIA A100按照指定规格序列创建MIG实例。例如，对于序列3g.20gb-3g.20gb，NVIDIA A100会被划分为两个规格为3g.20gb的MIG。

> **注意** 控制台标签的值不支持半角逗号（,），所以MIG序列需要使用短划线（-）连接。例如，在控制台填写标签 ack.aliyun.com/gpu-partition-sequence=1g.5gb,2g.10gb,3g.20gb 是不合法的，需要使用 ack.aliyun.com/gpu-partition-sequence=1g.5gb-2g.10gb-3g.20gb 。

容器服务控制台节点标签配置示例

创建节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面，单击**创建节点池**。

6. 在**创建节点池**对话框，配置节点池相关参数。

   有关配置项的详细说明，请参见创建托管GPU集群。部分配置项的说明如下所示：

参数	说明
实例规格	选择**弹性裸金属服务器 > GPU型**，选择名称首部为ecs.ebmgn7.的机型。
节点标签	在节点标签参数栏添加标签，单击图标，在**键**文本框中输入 `ack.aliyun.com/gpu-partition-sequence` 或 `ack.aliyun.com/gpu-partition-sequence` ，在**值**文本框中输入相应的MIG实例规格或MIG序列。 关于节点池标签的更多信息，请参见使用节点池标签划分MIG。此处以 `ack.aliyun.com/gpu-partition-sequence=3g.20gb-3g.20gb` 为例。 

7. 参数配置完成后，单击**确认配置**。

验证节点池中的节点是否开启MIG功能

当节点池创建完成后，根据以下操作验证节点池中的节点是否开启MIG功能。

1. 执行以下命令，查询开启MIG能力的节点。

   ```
   kubectl get nodes -l aliyun.accelerator/nvidia_mig_sequence
   ```

   预期输出：

   ```
   NAME          STATUS  ROLES   AGE  VERSION
   cn-beijing.192.168.XX.XX  Ready   <none>  99s  v1.20.4-aliyun.1
   ```

2. 执行以下命令，查看上述节点上报的MIG资源名称。

   资源名称的首部为 `nvidia.com/mig-` 。

   ```
   kubectl get nodes cn-beijing.192.168.XX.XX -o yaml | grep " nvidia.com/mig-" | uniq
   ```

   预期输出：

   ```
   nvidia.com/mig-3g.20gb: "16"
   ```

   从预期输出可得，节点的资源名称为 `nvidia.com/mig-3g.20gb` ，该节点总共有16个3g.20gb MIG实例可用。

   > **说明** 节点上报的MIG资源名称与指定的MIG实例规格或规格序列有关。本文档以 `ack.aliyun.com/gpu-partition-sequence=3g.20gb-3g.20gb` 为例。

3. 申请MIG资源。

   如果某个Pod需要申请MIG资源，需要在应用Pod的YAML中 `resources.limits` 部分填写MIG资源名称及个数。

```
// 省略其他。
    resources:
     limits:
      nvidia.com/mig-<实例规格名称，如3g.20gb>: <申请资源个数>
.....
```

i. 根据以下示例创建*mig-sample.yaml*文件。

该示例申请了2个3g.20gb的MIG资源。

```
cat > /tmp/mig-sample.yaml <<- EOF
apiVersion: batch/v1
kind: Job
metadata:
 name: pytorch-mnist
spec:
 parallelism: 1
 template:
  spec:
   containers:
   - name: pytorch-mnist
    image: registry.cn-beijing.aliyuncs.com/ai-samples/nvidia-pytorch-sample:20.11-py3
    command:
    - python
    - main.py
    resources:
     limits:
      nvidia.com/mig-3g.20gb: 2 #申请2个3g.20gb的MIG实例。
   restartPolicy: Never
EOF
```

ii. 执行以下命令，部署应用。

```
kubectl apply -f /tmp/mig-sample.yaml
```

4. 查看Pod是否处于Running状态，然后查看Pod日志。

i. 执行以下命令，查看Pod是否处于Running状态。

```
kubectl get pod
```

预期输出：

```
NAME         READY  STATUS   RESTARTS  AGE
pytorch-mnist-t2ncm  1/1   Running  0     2m25s
```

ii. 当Pod处于Running状态，执行以下命令查看Pod日志。

```
kubectl logs pytorch-mnist-t2ncm
```

预期输出：

```
/opt/conda/lib/python3.6/site-packages/torchvision/datasets/mnist.py:480: UserWarning: The given NumPy array is not writeable, and PyTorch does not support non-writeable tensors. This means you can write to the underlying (supposedly non-writeable) NumPy array using the tensor. You may want to copy the array to protect its data or make it writeable before converting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered internally at  ../torch/csrc/utils/tensor_numpy.cpp:141.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
cuda is available:  True
current device:  0
current device name:  A100-SXM4-40GB MIG 3g.20gb
Using downloaded and verified file: ../data/MNIST/raw/train-images-idx3-ubyte.gz
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
Using downloaded and verified file: ../data/MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
Using downloaded and verified file: ../data/MNIST/raw/t10k-images-idx3-ubyte.gz
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
Using downloaded and verified file: ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
Processing...
Done!
......
```

5. 进入容器并查看设备信息。

i. 执行以下命令，进入容器。

```
kubectl exec -ti pytorch-mnist-t2ncm bash
```

ii. 执行以下命令，查看设备信息。

```
nvidia-smi -L
```

预期输出：

```
GPU 0: A100-SXM4-40GB (UUID: GPU-da5ae315-b486-77ea-9527-3cbbf5b9****)
  MIG 3g.20gb Device 0: (UUID: MIG-GPU-da5ae315-b486-77ea-9527-3cbbf5b9****/1/0)
GPU 1: A100-SXM4-40GB (UUID: GPU-cc7ebb1f-902c-671c-3fcb-8daf49cc****)
  MIG 3g.20gb Device 0: (UUID: MIG-GPU-cc7ebb1f-902c-671c-3fcb-8daf49cc****/2/0)
```

从预期输出中可看到两个规格为3g.20gb的MIG实例。

6. Pod运行完成后，执行以下命令，查看Pod状态。

```
kubectl get pod
```

预期输出：

```
NAME          READY  STATUS    RESTARTS  AGE
pytorch-mnist-t2ncm  0/1    Completed  0      3m1s
```

若Pod处于Completed状态，表示节点池中的节点已开启MIG功能。

18.6.7. 手动升级GPU节点驱动

如果您使用的CUDA库需要匹配更高版本的NVIDIA驱动，需要升级节点的NVIDIA驱动。本文介绍如何手动升级GPU节点驱动。

前提条件

通过kubectl工具连接集群

操作步骤

1. 执行以下命令，将待升级驱动的GPU节点设置为不可调度状态。

```
kubectl cordon <$Node_ID>
```

> **说明** *<$Node_ID>*为节点的ID。

预期输出：

```
node/<$Node_ID> already cordoned
```

2. 执行以下命令，将待升级驱动的GPU节点从集群中移除。

```
kubectl drain <$Node_ID> --grace-period=120 --ignore-daemonsets=true  #设置120秒钟的宽限期限，让任务优雅的关闭。
```

预期输出：

```
node/cn-beijing.<$Node_ID> cordoned
WARNING: Ignoring DaemonSet-managed pods: flexvolume-9scb4, kube-flannel-ds-r2qmh, kube-proxy-worker-l62sf, logtail-ds-f9vbg
pod/nginx-ingress-controller-78d847fb96-5fkkw evicted
```

3. 卸载该GPU节点的NVIDIA驱动。

> **说明** 本步骤中卸载的是418.87.01版本的驱动包，如果您的目标驱动版本不是418.87.01，则需要到NVIDIA的官网中下载对应的驱动安装包，并将本步骤中的418.87.01版本的驱动包替换成您实际的版本。

   i. 登录到该GPU节点，执行以下命令，查看驱动版本。

   > **说明** 关于如何登录到GPU节点，请参见通过密码认证登录Linux实例和通过密码认证登录Windows实例。

   ```
   nvidia-smi -a | grep 'Driver Version'
   ```

   预期输出：

   ```
   Driver Version                  : 418.87.01
   ```

   ii. 执行以下命令，下载NVIDIA驱动安装包。

   ```
   cd /tmp/
   wget https://us.download.nvidia.com/tesla/418.87/NVIDIA-Linux-x86_64-418.87.01.run
   ```

   iii. 执行以下命令，卸载当前NVIDIA驱动。

   ```
   chmod u+x NVIDIA-Linux-x86_64-418.87.01.run
   ./NVIDIA-Linux-x86_64-418.87.01.run --uninstall -a -s -q
   ```

4. 执行以下命令，重启GPU节点。

```
reboot
```

5. 执行以下命令，从NVIDIA官网下载并安装您需要的NVIDIA驱动，本文以下载418.181.07版本为例。

```
cd /tmp/
wget https://us.download.nvidia.com/tesla/418.181.07/NVIDIA-Linux-x86_64-418.181.07.run
chmod u+x NVIDIA-Linux-x86_64-418.181.07.run
sh ./NVIDIA-Linux-x86_64-418.181.07.run -a -s -q
```

6. 参考以下命令，根据实际情况，添加优化参数。

```
nvidia-smi -pm 1 || true                    # 代表启用Persistence模式。
nvidia-smi -acp 0 || true                   # 切换权限要求为UNRESTRICTED。
nvidia-smi --auto-boost-default=0 || true       # 不开启自动提升模式，0代表不启用。
nvidia-smi --auto-boost-permission=0 || true     # 允许非管理员控制自动提升模式，0代表允许，1代表不允许。
nvidia-modprobe -u -c=0 -m || true            # 加载NVIDIA统一内存内核模块，而不是NVIDIA内核模块，且使用给定的编号创建NVIDIA设备文件。
```

7. 如果需要开机自启动NVIDIA驱动，请查看*/etc/rc.d/rc.local*文件，确认包含以下配置，如果没有请手动添加。

```
nvidia-smi -pm 1 || truenvidia-smi -acp 0 || truenvidia-smi --auto-boost-default=0 || truenvidia-smi --auto-boost-permission=0 || truenvidia-modprobe -u -c=0 -m || true
```

8. 执行以下命令，重启Kubelet和Docker服务。

```
systemctl restart docker && systemctl restart kubelet
```

9. 执行以下命令，将GPU节点重新设置为可调度状态。

```
kubectl uncordon <$Node_ID>
```

预期输出：

```
node/<$Node_ID> already uncordoned
```

10. 执行以下命令，在GPU节点的device-plugin Pod上验证版本信息。

```
kubectl exec -n kube-system -t nvidia-device-plugin-<$Node_ID> nvidia-smi
```

预期输出：

```
Thu Jan 17 00:33:27 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.181.07   Driver Version: 418.181.07   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 P100-PCIE...  On   | 00000000:00:09.0 Off |                    0 |
| N/A  27C    P0    28W / 250W |    0MiB / 16280MiB |    0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU     PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------
```

从上述输出信息，可以知道GPU节点的版本已成功升级到预期版本。
如果没有升级到预期版本，您可以通过执行 `docker ps` 命令，查看GPU节点的容器是否启动。关于GPU节点容器启动问题，请参见修复GPU节点容器启动问题。

18.6.8. 修复GPU实例重启或被置换后设备ID变更问题

GPU实例宕机后，GPU设备ID可能会变化，会导致容器无法正常启动。GPUOps检测GPU实例的GPU设备ID与*/var/lib/kubelet/device-plugins/kubelet_internal_checkpoint*中存储的GPU设备ID是否一致。如果不一致，GPUOps会删除 `checkpoint` 文件，由Kubelet生成新的 `checkpoint` 文件，保证存储的GPU设备ID与真实的GPU设备ID一致。本文介绍如何修复由GPU实例宕机引起的设备ID变更问题。

前提条件

- 创建托管GPU集群或创建专有GPU集群。
- 集群有一台启动SSH，且公网可访问的跳板机。具体操作，请参见如何为已有集群开启SNAT。

背景信息

- GPU实例宕机后，重启或置换GPU实例时，GPU设备ID可能会变化。该ID如果与*/var/lib/kubelet/device-plugins/kubelet_internal_checkpoint*中存储的GPU设备ID信息不一致，会导致容器无法正常启动。

  > **说明** 以下场景都有可能引起GPU设备ID变化：
  > - GPU实例宕机后，重启或置换GPU实例时。
  > - 手动重启GPU实例时。

- GPUOps是一个可执行的二进制程序，在Linux上可直接执行。具体下载地址，请参见GPUOps。
- 一个GPU实例上可以有多个GPU卡，每个GPU卡都有一个GPU设备ID。

步骤一：部署GPUOps

单机部署GPUOps

1. 执行以下命令，将GPUOps复制到*/usr/local/bin/*目录下，并赋予可执行权限。

   ```
   cp ./gpuops /usr/local/bin/
   chmod +x gpuops
   ```

2. 执行以下命令，把GPUOps添加到开机自动启动的服务。

   ```
   cat > /etc/systemd/system/gpuops.service <<EOF
   [Unit]
   Description=Gpuops: check kubelet checkpoint gpu status
   After=network-online.target
   Wants=network-online.target
   [Service]
   Type=oneshot
   RemainAfterExit=yes
   ExecStart=/usr/local/bin/gpuops check
   [Install]
   WantedBy=multi-user.target
   EOF
   systemctl enable gpuops.service
   ```

批量部署GPUOps

在GPU集群中，使用以下YAML文件样例部署DaemonSet。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: gpuops-deploy
 labels:
  k8s-app: gpuops-deploy
spec:
 selector:
  matchLabels:
   name: gpuops-deploy
 template:
  metadata:
   labels:
    name: gpuops-deploy
  spec:
   hostPID: true
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
       - matchExpressions:
         - key: aliyun.accelerator/nvidia_name
           operator: Exists
   containers:
    - name: gpuops
      image: registry.cn-beijing.aliyuncs.com/acs/gpuops:latest
      command:
      securityContext:
       privileged: true
      volumeMounts:
       - name: hostbin
         mountPath: /workspace/host/usr/local/bin
       - name: hostsystem
         mountPath: /workspace/host/etc/systemd/system
   terminationGracePeriodSeconds: 30
   volumes:
    - name: hostbin
      hostPath:
       path: /usr/local/bin
    - name: hostsystem
      hostPath:
       path: /etc/systemd/system
```

成功部署该DaemonSet后，GPU集群会自动在所有Worker节点执行以下操作：

- 将GPUOps复制到*/usr/local/bin/*目录下，并赋予可执行权限。
- 把GPUOps添加到开机自动启动的服务。

以后集群中的GPU实例宕机后，重启或置换GPU实例时，GPUOps会保证存储的GPU设备ID与真实的GPU设备ID一致。

步骤二：验证GPUOps修复GPU设备ID变更情况

1. GPUOps安装完成后，在GPU集群部署1个Pod。

   部署Pod的YAML文件样例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: app-3g-v1
 labels:
  app: app-3g-v1
spec:
 replicas: 1
 serviceName: "app-3g-v1"
 podManagementPolicy: "Parallel"
 selector: # define how the deployment finds the pods it manages
  matchLabels:
   app: app-3g-v1
 template: # define the pods specifications
  metadata:
   labels:
    app: app-3g-v1
  spec:
   containers:
   - name: app-3g-v1
    image: registry.cn-shanghai.aliyuncs.com/tensorflow-samples/cuda-malloc:3G
    resources:
     limits:
      nvidia.com/gpu: 1
```

2. 执行以下命令，验证Pod部署情况。

```
kubectl get pod
```

预期输出：

```
NAME              READY  STATUS   RESTARTS  AGE
app-3g-v1-0          1/1   Running   0     22s
```

3. 登录到该GPU实例，执行以下命令，重启GPU节点。

> **说明** 关于如何登录到GPU实例，请参见通过密码认证登录Linux实例。

```
sudo reboot
```

4. 执行以下命令，查看GPUOps的执行日志信息。

```
journalctl -u gpuops
```

   - 如果输出的日志内容如下，说明此时 `checkpoint` 文件中的GPU设备ID和真实的GPU设备ID是一致的，无需处理。

```
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z gpuops[21976]: {"level":"info","msg":"check gpu start...","time":"
2020-06-28T14:49:00+08:00"}
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z gpuops[21976]: {"level":"info","msg":"find nvidia gpu: [\"GPU-383
cd6a5-00a6-b455-62c4-c163d164b837\"]","time":"2020-06-28T14:49:00+08:00"}
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z gpuops[21976]: {"level":"info","msg":"read checkpoint: {\"Data\":
{\"PodDeviceEntries\":[{\"PodUID\":\"300f47eb-e5c3-4d0b-9a87-925837b67732\",\"ContainerName\":\"app-3
g-v1\",\"ResourceName\":\"aliyun.com/g
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z systemd[1]: Started Gpuops: check kubelet checkpoint gpu status
.
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z gpuops[21976]: {"level":"info","msg":"find registered gpu: [\"GPU
-383cd6a5-00a6-b455-62c4-c163d164b837\"]","time":"2020-06-28T14:49:00+08:00"}
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z gpuops[21976]: {"level":"info","msg":"find pod gpu: null","time":"
2020-06-28T14:49:00+08:00"}
6月 28 14:49:00 iZ2vc1mysgx8bqdv3oyji9Z gpuops[21976]: {"level":"info","msg":"cached gpu info in checkpo
int is up to date, check gpu finished","time":"2020-06-28T14:49:00+08:00"}
```

- 如果输出的日志有以下 warning 级别的日志时，则说明GPU设备ID发生了变化，GPUOps会删除 checkpoint ，Kubernetes会重新生成 checkpoint 文件，保证设备信息同步。

```
6月 28 14:41:16 iZ2vc1mysgx8bqdv3oyji9Z systemd[1]: Starting Gpuops: check kubelet checkpoint gpu statu
s...
6月 28 14:41:16 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"check gpu start...","time":"20
20-06-28T14:41:16+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"find nvidia gpu: [\"GPU-68ce6
4fb-ea68-5ad6-7e72-31f3f07378df\"]","time":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"read checkpoint: {\"Data\":{\"
PodDeviceEntries\":[{\"PodUID\":\"300f47eb-e5c3-4d0b-9a87-925837b67732\",\"ContainerName\":\"app-3g-
v1\",\"ResourceName\":\"aliyun.com/gpu
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"find registered gpu: [\"GPU-7f
d52bfc-364d-1129-a142-c3f10e053ccb\"]","time":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"find pod gpu: [\"GPU-7fd52bf
c-364d-1129-a142-c3f10e053ccb\"]","time":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"warning","msg":"the registered gpu uuid n
ot equal with nvidia gpu","time":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"warning","msg":"cached gpu info in check
point is out of date","time":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"delete checkpoint file success
","time":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"kubelet restart success","tim
e":"2020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z gpuops[951]: {"level":"info","msg":"check gpu finished","time":"2
020-06-28T14:41:17+08:00"}
6月 28 14:41:17 iZ2vc1mysgx8bqdv3oyji9Z systemd[1]: Started Gpuops: check kubelet checkpoint gpu status
.
```

  - 执行以下命令，查看真实的GPU设备ID。

    ```
    nvidia-smi -L
    ```

    预期输出：

    ```
    GPU 0: Tesla T4 (UUID: GPU-0650a168-e770-3ea8-8ac3-8a1d419763e0)
    ```

- 执行以下命令，查看 checkpoint 文件里存储的GPU设备ID。

  ```
  cat /var/lib/kubelet/device-plugins/kubelet_internal_checkpoint
  ```

  预期输出：

  ```
  {"Data":{"PodDeviceEntries":null,"RegisteredDevices":{"nvidia.com/gpu":["GPU-0650a168-e770-3ea8-8ac3-8a1d419763e0"]}},"Checksum":3952659280}
  ```

  从上述输出信息，可知 checkpoint 文件里存储的GPU设备ID与真实的GPU设备ID一致，说明GPUOps成功修复了设备ID不一致的问题。

18.6.9. 更新ACK集群中GPU虚拟化型（vGPU）实例的NVIDIA驱动License

如果集群中存在vGPU实例类型的节点，您需要购买NVIDIA官方提供的GRID License，且自建License服务器该节点才能正常工作。本文介绍如何更新ACK集群中vGPU实例的NVIDIA驱动License。

前提条件

- 购买相应的License，更多信息，请参见NVIDIA License购买。
- 本文适用的ACK集群包括专有版、托管版和Pro版。

License服务器搭建完成后，按照以下步骤更新节点GRID License。

> **说明** 请购买ECS实例并参考NVIDIA官网教程搭建License服务器。更多信息，请参见NVIDIA。

vGPU实例已加入ACK集群

1. 登录vGPU节点，修改*/etc/nvidia/gridd.conf*文件中以下字段的值。

   > **说明** 关于如何登录到GPU节点，请参见通过密码认证登录Linux实例和通过密码认证登录Windows实例。

   ```
   ServerAddress=<your License Server address>
   ServerPort=<License Server port>
   ```

2. 执行以下命令重启NVIDIA驱动。

   ```
   systemctl daemon-reload
   systemctl restart nvidia-gridd
   ```

3. 执行以下命令，确认该命令是否能正常执行。

   ```
   nvidia-smi
   ```

   预期输出：

```
root@i[redacted]:~# nvidia-smi
Thu Nov  5 17:01:38 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 430.99       Driver Version: 430.99       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:07.0 Off |                    0 |
| N/A   31C    P0    39W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

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

从上述输出可知，`nvidia-smi`命令可以被正常执行，说明您已成功更新该vGPU实例的NVIDIA驱动License。

18.7. NPU调度

18.7.1. Kubernetes集群支持NPU调度

通过Kubernetes管理ALI NPU设备，能够帮助您快速部署AI推理任务，监控NPU资源使用情况等。本文主要为您介绍如何使用带有NPU资源的Kubernetes集群。

前提条件

您需要开通容器服务和访问控制（RAM）服务。

创建NPU集群。

背景信息

与大型算法模型中CPU相比，NPU处理器最明显的优势是信息处理能力快。NPU采用了“数据驱动并行计算”架构，颠覆了CPU所采用的传统冯·诺依曼计算机架构，这种数据流类型的处理器大大提升了计算能力与功耗的比率。NPU特别适合处理视频、图像类的海量多媒体数据的场景，不仅速度比CPU要高出100～1000倍，同时功耗也远远低于CPU。

您可以通过阿里云Kubernetes集群 + ALI NPU运行机器学习，图像处理等高运算密度等任务，实现快速部署和弹性扩缩容等功能。

> **说明** 如果想了解ALI NPU的相关信息，请参见AliNPU Website。

本文将以创建Kubernetes集群时，添加ecs.ebman1.26xlarge实例为例说明如何使用NPU。

使用NPU资源

如果某一个Pod需要使用NPU资源，需要在`resources.limits`定义`aliyun.com/npu`的值。

```
apiVersion: v1
kind: Pod
metadata:
 name: <pod名称>
spec:
 containers:
  - name: <容器名称>
    image: <镜像名称>
    resources:
     limits:
      aliyun.com/npu: <请求npu资源数>
```

运行TensorFLow的NPU实验环境

您可以在集群中使用NPU资源完成模型训练。本例中，将会启动一个使用NPU资源的Pod进行模型训练。

1. 连接集群，请参见在CloudShell上通过kubectl管理Kubernetes集群。

   在CloudShell界面执行以下操作。

   ```
   cat > test-pod.yaml <<- EOF
   apiVersion: v1
   kind: Pod
   metadata:
    name: test-npu-pod
   spec:
    restartPolicy: Never
    imagePullSecrets:
     - name: regsecret
    containers:
     - name: resnet50-npu
       image: registry.cn-shanghai.aliyuncs.com/hgai/tensorflow:v1_resnet50-tensorflow1.9.0-toolchain1.0.2-centos7.6
       resources:
        limits:
         aliyun.com/npu: 1 # requesting  NPUs
   EOF
   ```

2. 执行以下命令，创建Pod。

   ```
   kubectl apply -f test-pod.yaml
   ```

3. 执行以下命令，查看Pod的状态。

   ```
   kubectl get po test-npu-pod
   ```

   > **说明** 如果Pod为Error状态，请执行 `kubectl logs test-npu-pod` 命令监听Pod日志，并排查修改。

执行结果

等待一段时间后，您可以通过执行以下命令查看Pod的状态。

```
kubectl get po test-npu-pod
```

如果查看到Pod的状态为Completed，再执行以下命令查看日志。

```
kubectl logs test-npu-pod
```

此时看到日志显示结果如下，表示训练任务完成。

```
2019-10-30 12:10:50.389452: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 AVX512F FMA
100%|##########| 98/98 [00:26<00:00, 3.67it/s]
resnet_v1_50, result = {'top_5': 0.9244321584701538, 'top_1': 0.7480267286300659}
```

18.8. GPU/NPU FAQ

- GPU问题诊断
- 如何在GPU节点中升级Kernel?
- 如何修复GPU节点的容器启动异常？
- 创建或扩容支持GPU计算的Kubernetes集群时如何安装NVIDIA驱动？
- 排查GPU监控常见问题
- 可用的GPU数目少于实际GPU数目
- GPU节点中重启Kubelet和Docker服务异常
- 修复GPU实例重启或被置换后设备ID变更问题
- 阿里云容器服务是否支持GPU虚拟化型（vGPU）实例？

阿里云容器服务是否支持GPU虚拟化型（vGPU）实例？

vGPU实例需要购买NVIDIA官方提供的GRID License才能正常工作，而阿里云并不提供License服务器。因此即使您创建了GPU虚拟化集群，vGPU实例也无法使用。因此，阿里云容器服务已不再支持在控制台选择vGPU实例作为集群节点。

如果您的业务对vGPU实例有强依赖，您可以向NVIDIA购买GRID License，自建License服务器。

> 说明
> - 更新ACK集群中vGPU实例的NVIDIA驱动License时，需要使用License服务器。
> - 购买ECS实例并参考NVIDIA官网教程搭建License服务器。更多信息，请参见NVIDIA。

如果您的License服务器已经搭建完成，请参考以下步骤将vGPU实例加入ACK集群。

将vGPU实例加入ACK集群

1. 提交工单，申请开放自定义系统镜像功能。
2. 基于CentOS 7.X和Alibaba Cloud Linux 2制作自定义系统镜像，镜像中需要安装NVIDIA GRID驱动并且正确配置GRID License。具体操作，请参见使用实例创建自定义镜像和在GPU虚拟化型实例中安装GRID驱动（Linux）。
3. 创建节点池。具体操作，请参见管理节点池。
4. 将vGPU实例加入到步骤3创建的节点池中，具体操作，请参见添加已有节点。

后续相关步骤：更新ACK集群中vGPU实例的NVIDIA驱动License

更新ACK集群中vGPU实例的NVIDIA驱动License，具体操作，请参见更新ACK集群中GPU虚拟化型（vGPU）实例的NVIDIA驱动License。

19.调度

19.1. CPU拓扑感知调度

19.1.1. CPU拓扑感知调度

ACK基于新版的Scheduling framework实现CPU拓扑感知调度及控制自动绑核策略，针对CPU敏感型的工作负载提供更好的性能。本文介绍如何使用CPU拓扑感知调度，及如何通过设置CPU Policy控制自动绑核策略。

前提条件

- 已创建ACK Pro版集群。具体操作，请参见创建ACK Pro版集群。

  > 注意 目前CPU拓扑感知调度仅支持ACK Pro托管版集群。如果您需要专有版集群，请提交工单申请白名单。

- 在使用CPU拓扑感知调度前，需要预先部署组件resource-controller。具体操作，请参见管理组件。
- 系统组件版本要求具体如下表所示。

组件	版本要求
Kubernetes	1.16及以上
Helm版本	3.0及以上版本
Docker版本	19.03.5
操作系统	CentOS 7.6、CentOS 7.7、Ubuntu 16.04、Ubuntu 18.04、Alibaba Cloud Linux 2

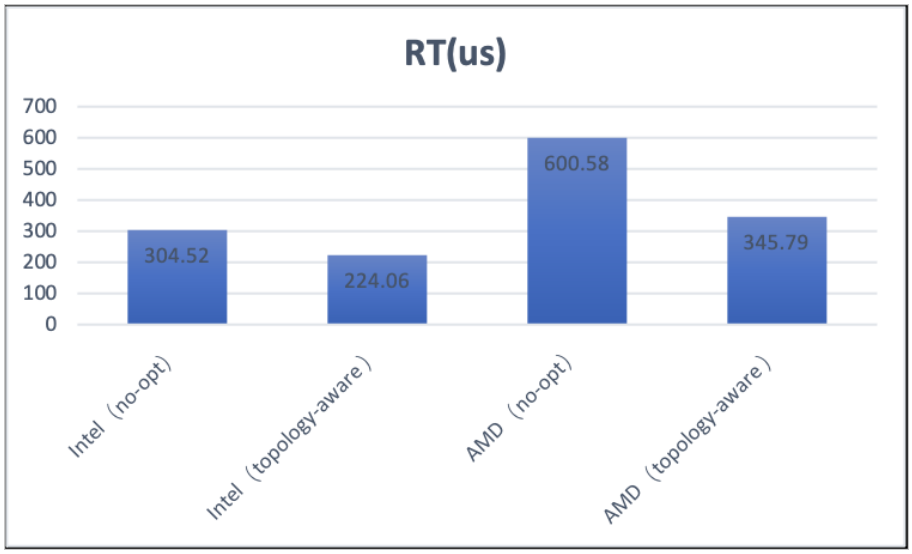
背景信息

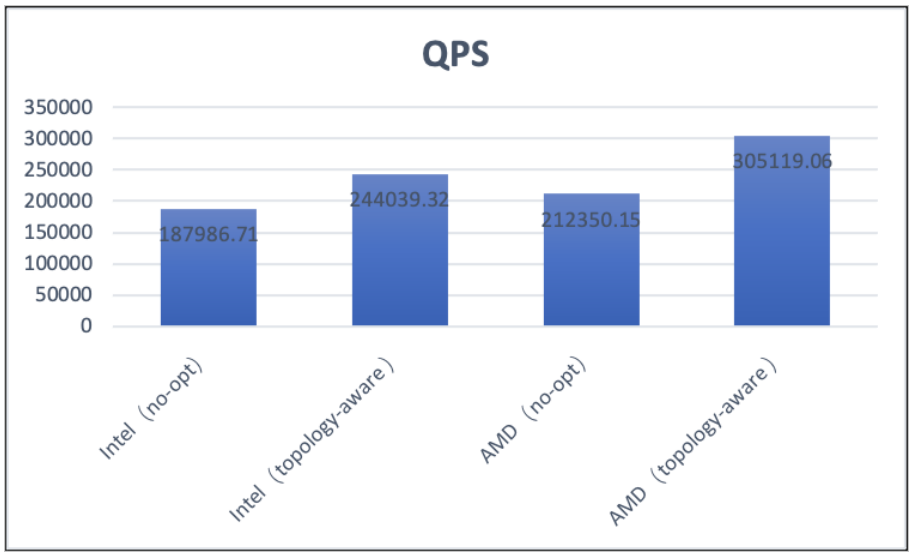
Kubernetes的节点会运行多个Pod，其中部分的Pod属于CPU密集型工作负载。在这种情况下，Pod之间会争抢节点的CPU资源。当争抢剧烈的时候，Pod会在不同的CPU Core之间进行频繁的切换，更糟糕的是在NUMA Node之间的切换。这种大量的上下文切换，会影响程序运行的性能。Kubernetes虽然有CPU Manager解决方案处理以上问题，但是因为CPU Manager特性是节点级别的CPU调度选择，所以无法在集群维度中选择最优的CPU Core组合。同时CPU Manager特性要求Pod是Guaranteed时（Pod中的每个容器必须指定CPU请求和CPU限制，并且两者要相等）才能生效，且无法适用于所有类型的Pod。

在以下场景中，建议使用CPU拓扑感知调度：

- 工作负载为计算密集型。
- 应用程序对CPU敏感。
- 运行在神龙裸金属（Intel、AMD）等多核机器上。

通过在物理机上部署两个4核8 GB的Nginx并进行压测，可以发现在Intel（104核）、AMD（256核）的物理机上，使用CPU拓扑感知调度能够将应用性能提升22%~43%。具体如以下两个图所示：





性能指标	Intel	AMD
QPS	提升22.9%	提升43.6%
AVG RT	下降26.3%	下降42.5%

在激活CPU拓扑感知调度时，可通过在所创建Pod的annotations处设置cpu-policy为static-burst，从而控制自动绑核策略。该策略能够对计算密集性负载，有效避免进程的物理核心争用及跨片内存访问，最大化利用碎片化CPU，在不改变硬件及虚拟机资源的前提下为密集计算型负载优化可用资源，进一步提升CPU使用率。

注意事项

- 在使用CPU拓扑感知调度前，请保证已部署组件resource-controller。
- 在使用CPU拓扑感知调度时，请保证节点的 `cpu-policy=none` 。

使用CPU拓扑感知调度

激活CPU拓扑感知调度前，需要在提交Pod时设置annotations和Containers的值。满足以下条件：

- 在annotations处，设置cpuset-scheduler为true，激活CPU拓扑感知调度。

- 需要设置Containers的 `resources.limit.cpu` 值，且该值为整数。

1. 使用以下模板创建*cal-pi.yaml*文件，在Pod中使用CPU拓扑感知调度。

```
apiVersion: v1
kind: Pod
metadata:
 name: cal-pi
 annotations:
  cpuset-scheduler: 'true' #添加注解，表示CPU拓扑感知调度。
  #cpu-policy: 'static-burst' #添加注解，控制自动绑核策略同时提升CPU碎片利用率。
spec:
 restartPolicy: Never
 containers:
 - image: registry.cn-zhangjiakou.aliyuncs.com/xianlu/java-pi
  name: cal-pi
  resources:
   requests:
    cpu: 4
   limits:
    cpu: 4 #需要设置resources.limit.cpu值。
  env:
  - name: limit
   value: "20000"
  - name: threadNum
   value: "3000"
```

> **注意** 在配置CPU拓扑感知调度时，可通过在annotations下设置cpu-policy为static-burst，从而控制自动绑核策略。需要配置时，请删除cpu-policy前的 `#` 。

2. 使用以下模板创建*go-demo.yaml*文件，在Deployment中使用CPU拓扑感知调度。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: go-demo
spec:
 replicas: 4
 selector:
  matchLabels:
   app: go-demo
 template:
  metadata:
   annotations:
    cpuset-scheduler: "true" #添加注解，表示CPU拓扑感知调度。
    #cpu-policy: 'static-burst' #添加注解，控制自动绑核策略，提升CPU碎片利用率。
   labels:
    app: go-demo
  spec:
   containers:
   - name: go-demo
    image: registry.cn-hangzhou.aliyuncs.com/haoshuwei24/go-demo:1k
    imagePullPolicy: Always
    ports:
    - containerPort: 8080
    resources:
     requests:
      cpu: 1
     limits:
      cpu: 4 #需要设置resources.limit.cpu值。
```

> **注意**
>
> - 在配置CPU拓扑感知调度时，可通过在annotations下设置cpu-policy为static-burst，从而控制自动绑核策略。需要配置时，请删除cpu-policy前的 `#` 。
> - 在 `template.metadata` 下配置Pod的 `annotations` 。

3. 执行以下命令，创建Pod及Deployment。

```
kubectl create -f cal-pi.yaml go-demo.yaml
```

验证CPU拓扑感知对应用性能的提升

本示例使用场景为：

- ACK Pro版集群版本为1.20。
- 所选实例的处理器型号为Intel Xeon(Cascade Lake) Platinum 8269CY或AMD Rome 7H12。
- 需要包含两个节点：一台压测机、一台测试机。

1. 执行以下命令，为测试机打上标签。

```
kubectl label node 192.168.XX.XX policy=intel/amd
```

2. 在测试机上部署Nginx服务。
   i. 使用以下YAML示例，创建Nginx服务配置文件。

- *service.yaml*

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service-nodeport
spec:
 selector:
   app: nginx
 ports:
  - name: http
    port: 8000
    protocol: TCP
    targetPort: 80
    nodePort: 32257
 type: NodePort
```

- *configmap.yaml*

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: nginx-configmap
data:
 nginx_conf: |-
  user  nginx;
  worker_processes  4;
  error_log  /var/log/nginx/error.log warn;
  pid      /var/run/nginx.pid;
  events {
    worker_connections  65535;
  }
  http {
    include     /etc/nginx/mime.types;
    default_type  application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
            '$status $body_bytes_sent "$http_referer" '
            '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  /var/log/nginx/access.log  main;
    sendfile      on;
    #tcp_nopush    on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;
  }
```

- *nginx.yaml*

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   annotations:
    #cpuset-scheduler: "true"  默认情况，不使用CPU拓扑感知调度。
   labels:
    app: nginx
  spec:
   nodeSelector:
    policy: intel7
   containers:
   - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    resources:
     requests:
      cpu: 4
      memory: 8Gi
     limits:
      cpu: 4
      memory: 8Gi
    volumeMounts:
     - mountPath: /etc/nginx/nginx.conf
      name: nginx
      subPath: nginx.conf
   volumes:
    - name: nginx
     configMap:
      name: nginx-configmap
      items:
       - key: nginx_conf
        path: nginx.conf
```

ii. 执行以下命令，创建Nginx服务。

```
kubectl create -f service.yaml configmap.yaml nginx.yaml
```

3. 登录压测机，执行以下命令下载 `wrk` 。

> **说明** 关于如何登录到节点，请参见通过密码认证登录Linux实例或通过密码认证登录Windows实例。

```
wget https://caishu-oss.oss-cn-beijing.aliyuncs.com/wrk?versionId=CAEQEBiBgMCGk565xxciIDdiNzg4NWIzMzZhZTQ1OTlhYzZhZjFhNmQ2MDNkMzA2 -O wrk
chmod +755 wrk
mv wrk /usr/local/bin
```

4. 执行以下命令，在压测机上进行测试并记录数据。

```
taskset -c 32-45 wrk --timeout 2s -t 20 -c 100 -d 60s --latency http://<测试机IP>:32257
```

预期输出：

```
20 threads and 100 connections
Thread Stats  Avg     Stdev    Max  +/- Stdev
Latency   600.58us    3.07ms 117.51ms   99.74%
Req/Sec    10.67k     2.38k   22.33k    67.79%
Latency Distribution
50%  462.00us
75%  680.00us
90%  738.00us
99%    0.90ms
12762127 requests in 1.00m, 10.10GB read
Requests/sec: 212350.15Transfer/sec:    172.13MB
```

5. 执行以下命令，删除已部署的Nginx。

```
kubectl delete deployment nginx
```

预期输出：

```
deployment "nginx" deleted
```

6. 使用以下YAML示例，部署开启CPU拓扑感知调度的Nginx。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   annotations:
    cpuset-scheduler: "true"
   labels:
    app: nginx
  spec:
   nodeSelector:
    policy: intel7
   containers:
   - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    resources:
     requests:
      cpu: 4
      memory: 8Gi
     limits:
      cpu: 4
      memory: 8Gi
    volumeMounts:
     - mountPath: /etc/nginx/nginx.conf
       name: nginx
       subPath: nginx.conf
   volumes:
    - name: nginx
      configMap:
       name: nginx-configmap
       items:
        - key: nginx_conf
          path: nginx.conf
```

7. 执行以下命令，在压测机上进行测试并记录数据进行对比。

```
taskset -c 32-45 wrk --timeout 2s -t 20 -c 100 -d 60s --latency http://<测试机IP>:32257
```

预期输出：

```
20 threads and 100 connections
ls Thread Stats  Avg      Stdev    Max    +/- Stdev
Latency       345.79us   1.02ms   82.21ms  99.93%
Req/Sec       15.33k    2.53k    25.84k   71.53%
Latency Distribution
50%  327.00us
75%  444.00us
90%  479.00us
99%  571.00us
18337573 requests in 1.00m, 14.52GB read
Requests/sec: 305119.06Transfer/sec:   247.34MB
```

通过两次数据的对比可以看出，开启CPU拓扑感知调度后，Nginx的性能提升43%。

验证自动绑核策略对应用性能的提升

本示例介绍在一个64核的节点上设置CPU Policy。在设置控制自动绑核策略后，相比于CPU拓扑感知调度一般可以提升7%~8%的CPU使用效率。

1. Pod或Deployment成功运行后，执行以下命令，查看Pod。

   ```
   kubectl get pods | grep cal-pi
   ```

   预期输出：

   ```
   NAME         READY   STATUS   RESTARTS  AGE
   cal-pi-d****    1/1    Running  1      9h
   ```

2. 执行以下命令，查看应用 `cal-pi-d****` 的日志。

   ```
   kubectl logs cal-pi-d****
   ```

   预期输出：

   ```
   computing Pi with 3000 Threads...computed the first 20000 digets of pi in 620892 ms!
   the first digets are: 3.14159264
   writing to pi.txt...
   finished!
   ```

3. 根据使用CPU拓扑感知调度，请删除cpu-policy前的 `#` ，在Pod及Deployment中使用CPU拓扑感知调度并控制自动绑核策略。

4. Pod或Deployment成功运行后，执行以下命令，查看Pod。

   ```
   kubectl get pods | grep cal-pi
   ```

   预期输出：

   ```
   NAME         READY   STATUS   RESTARTS  AGE
   cal-pi-e****    1/1    Running  1      9h
   ```

5. 执行以下命令，查看应用 `cal-pi-e****` 的日志。

   ```
   kubectl logs cal-pi-e****
   ```

   预期输出：

   ```
   computing Pi with 3000 Threads...computed the first 20000 digets of pi in 571221 ms!
   the first digets are: 3.14159264
   writing to pi.txt...
   finished!
   ```

通过与步骤2的日志数据对比，设置CPU Policy后Pod的性能提升8%。

19.1.2. 通过resource-controller组件动态修改Pod资源上限

resource-controller是一款基于Kubernetes CRD（CustomResourceDefinition）的控制器，它能够在不重启Pod的情况下，动态地修改Pod的CPU、内存等资源，从而保证容器能够在不影响当前业务的情况下按照您期望的资源使用率去运行。本文主要介绍部署resource-controller组件后如何动态地修改Pod资源上限。

前提条件

- 部署resource-controller组件。具体操作，请参见管理组件。
- 通过kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

当您在使用Kubernetes的过程中遇到以下情况时，建议部署组件resource-controller动态地修改Pod资源上限：

- Pod已经在运行中了，突然发现CPU Limit设置的值偏小，限制了进程的运行速度。
- Pod初始化时没有设置Limit，后来发现CPU负载过高，希望限制CPU Limit，但又不要影响其他应用。
- Pod内存使用率逐渐升高，快要接近Memory Limit，且马上就会触发OOM（Out Of Memory）Killer，希望在不重启Pod的前提下提高内存的Limit。

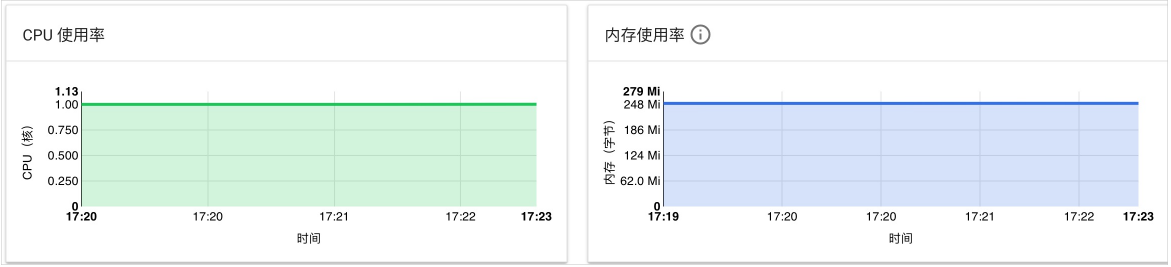
不论您处于以上情况的哪一种情况，根据目前Kubernetes的设计规范，只能重新编辑PodSpec，设计完资源的Limit后触发Pod重建。如果您使用的Pod是一个在线应用，可能会导致应用再次访问失败，且流量异常升高；如果您使用的Pod是一个离线任务，可能会导致前面几个小时计算数据全部丢失。

动态调整Pod的CPU和内存

1. 部署一个模拟的Pod任务，它是一个压力测试程序，会尝试使用2个CPU和256 MB的内存资源。

   使用以下模板样例部署一个模拟的Pod任务，将CPU的限制值设置为1。

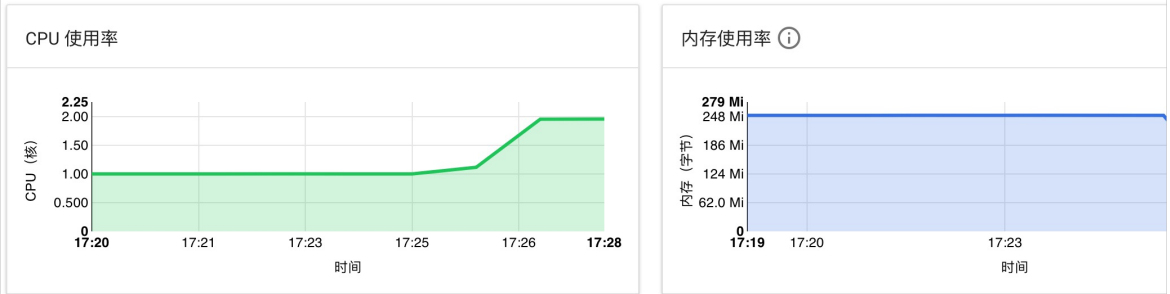
```
apiVersion: v1
kind: Pod
metadata:
 name: pod-demo
spec:
 containers:
 - name: pod-demo
  image: polinux/stress
  resources:
   requests:
    memory: "50Mi"
   limits:
    memory: "1000Mi"
    cpu: 1
  command: ["stress"]
  args: ["--vm", "1", "--vm-bytes", "256M", "-c", "2", "--vm-hang", "1"]
```



由上图可以看出，目前Pod被限制住了，只能使用1个CPU。

2. 提交以下模板所示的CRD，动态地修改CPU和内存的限制值。

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
 name: cgroups-sample
spec:
 pod:
  name: pod-demo
  namespace: default
  containers:
  - name: pod-demo
   cpu: 2000m
   memory: 5000Mi
```



由上图可以看出，Pod的CPU使用率由原来的1个CPU飙升到了2个CPU。

3. 执行以下命令，查看Pod运行情况。

```
kubectl describe pod pod-demo
```

系统输出类似以下结果，说明Pod运行正常，没有被重启。

```
Events:
 Type   Reason       Age  From                    Message
 ----   ------       ---- ----                    -------
 Normal Scheduled    13m  default-scheduler       Successfully assigned default/pod-demo to cn-zhangj
iakou.192.168.3.238
 Normal Pulling      13m  kubelet, cn-zhangjiakou.192.168.3.238  Pulling image "polinux/stress"
 Normal Pulled       13m  kubelet, cn-zhangjiakou.192.168.3.238  Successfully pulled image "polinux/stress"
 Normal SuccessfulChange 60s  cgroups-controller      Change pod pod-demo cpu to 2000
 Normal SuccessfulChange 60s  cgroups-controller      Change pod pod-demo memory to 524288000000
```

绑定Pod到CPU

1. 使用以下模板样例，让Pod运行一个压测程序，使用的资源为0.5个CPU。

```
apiVersion: v1
kind: Pod
metadata:
 name: pod-demo
 annotations:
  cpuset-scheduler: 'true' #添加注解，表示CPU拓扑感知调度。
spec:
 containers:
 - name: pod-demo
  image: polinux/stress
  resources:
   requests:
    memory: "50Mi"
   limits:
    memory: "1000Mi"
    cpu: 0.5
  command: ["stress"]
  args: ["--vm", "1", "--vm-bytes", "556M", "-c", "2", "--vm-hang", "1"]
```

2. 查看节点 cn-beijing.192.168.8.241 上各CPU利用率，可以发现各个CPU的利用率各不相同，且一直在动态变化。

```
top - 22:17:34 up 4 days, 10:29,  1 user,  load average: 0.33, 0.88, 0.95
Tasks: 179 total,   3 running, 176 sleeping,   0 stopped,   0 zombie
%Cpu0  : 13.1 us,  0.7 sy,  0.0 ni, 85.9 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  7.3 us,  7.7 sy,  0.0 ni, 84.7 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 12.4 us,  0.7 sy,  0.0 ni, 86.6 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 18.3 us,  0.7 sy,  0.0 ni, 80.7 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
```

3. 使用以下模板样例，提交绑定CPU的CRD，将Pod绑定到CPU2和CPU3上。

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
 name: cgroups-sample-cpusetpod
spec:
 pod:
  name: pod-demo
  namespace: default
  containers:
  - name: pod-demo
   cpuset-cpus: 2-3
```

4. 查看该节点各CPU利用率，发现CPU2、CPU3的使用率加起来一直稳定在了50%左右，每个CPU的利用率稳定在25%左右，说明该Pod被绑定在了CPU2和CPU3上，与预期保持一致，且Pod没有被重启。

```
top - 22:11:02 up 4 days, 10:22,  1 user,  load average: 0.04, 0.36, 0.84
Tasks: 177 total,   3 running, 174 sleeping,   0 stopped,   0 zombie
%Cpu0  :  2.7 us,  0.7 sy,  0.0 ni, 96.3 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  3.3 us,  1.0 sy,  0.0 ni, 95.3 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 27.2 us,  0.7 sy,  0.0 ni, 71.8 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 21.4 us,  5.7 sy,  0.0 ni, 72.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

绑定Deployment到CPU

1. 使用以下模板样例，让Deployment运行一个共有两个实例的压测程序，每个实例使用的资源为0.5个CPU。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: go-demo
 labels:
  app: go-demo
spec:
 replicas: 2
 selector:
  matchLabels:
   app: go-demo
 template:
  metadata:
   annotations:
    cpuset-scheduler: "true" #添加注解，表示CPU拓扑感知调度。
   labels:
    app: go-demo
  spec:
   nodeName: cn-beijing.192.168.8.240 #选择调度到同一个节点上。
   containers:
   - name: go-demo
    image: polinux/stress
    command: ["stress"]
    args: ["--vm", "1", "--vm-bytes", "556M", "-c", "1", "--vm-hang", "1"]
    imagePullPolicy: Always
    resources:
     requests:
      cpu: 0.5
     limits:
      cpu: 0.5 #需要设置resources.limit.cpu值。
```

2. 查看节点 `cn-beijing.192.168.8.240` 上各CPU利用率，可以发现各个CPU的利用率各不相同，且一直在动态变化。

```
top - 11:39:01 up 23:50,  2 users,  load average: 1.76, 1.91, 1.39
Tasks: 189 total,   4 running, 185 sleeping,   0 stopped,   0 zombie
%Cpu0  : 30.4 us,  5.4 sy,  0.0 ni, 64.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 29.4 us,  4.7 sy,  0.0 ni, 65.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  7.0 us,  8.7 sy,  0.0 ni, 84.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 50.3 us,  1.3 sy,  0.0 ni, 48.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

3. 使用以下模板样例，提交绑定CPU的CRD，将该Deployment中的Pod绑定在CPU2和CPU3上。

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
 name: cgroups-cpuset-sample
spec:
 deployment:
  name: go-demo
  namespace: default
  containers:
  - name: go-demo
   cpuset-cpus: 2,3 #绑定Pod到CPU2、CPU3上。
```

4. 查看该节点各CPU利用率，发现CPU2、CPU3的使用率一直稳定在了50%左右，说明该Deployment的两个实例分别绑定在了CPU2、CPU3上，与预期保持一致，且Pod没有被重启。

```
top - 11:30:56 up 23:42,  2 users,  load average: 2.01, 1.95, 1.12
Tasks: 180 total,   4 running, 176 sleeping,   0 stopped,   0 zombie
%Cpu0  :  4.4 us,  2.4 sy,  0.0 ni, 93.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  4.4 us,  2.3 sy,  0.0 ni, 92.6 id,  0.3 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  : 52.7 us,  8.0 sy,  0.0 ni, 39.0 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 50.7 us, 10.7 sy,  0.0 ni, 38.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

控制磁盘IOPS

> **说明** 如果需要控制磁盘IOPS，请您使用Alibaba Cloud Linux 2操作系统创建Worker节点。

1. 部署用于测试的Fio容器，该容器使用Fio工具对磁盘进行写的压测。

   使用以下模板样例，创建一块挂载在 `cn-beijing.192.168.0.182` 节点上云盘，挂载的设备号为 `/dev/vdb1` ，挂载路径为*/mnt*。

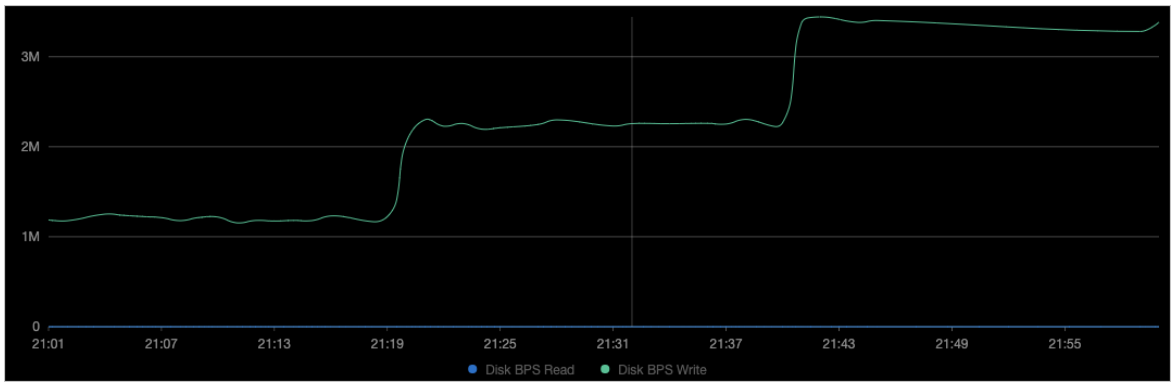
```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: fio
 labels:
  app: fio
spec:
 selector:
  matchLabels:
   app: fio
 template:
  metadata:
   labels:
    app: fio
  spec:
   nodeName: cn-beijing.192.168.0.182  # Pod部署在节点cn-beijing.192.168.0.182上。
   containers:
   - name: fio
    image: registry.cn-beijing.aliyuncs.com/shuangkun/fio:v1
    command: ["sh", "-c"]
    # 使用Fio工具对磁盘IOPS进行写测试。
    args: ["fio -filename=/data/test -direct=1 -iodepth 1 -thread -rw=write -ioengine=psync -bs=16k -size=2G -n
umjobs=10 -runtime=12000 -group_reporting -name=mytest"]
    volumeMounts:
     - name: pvc
       mountPath: /data  #挂载在/data的路径下。
   volumes:
    - name: pvc
      hostPath:
       path: /mnt
```

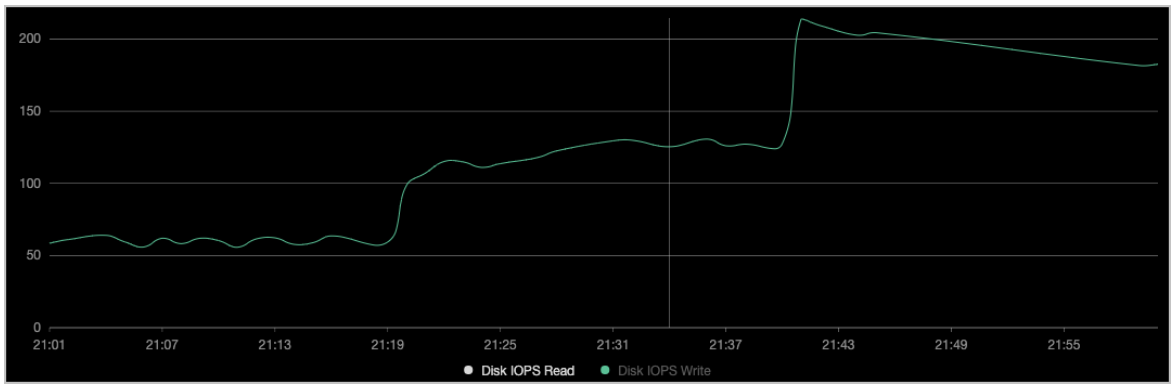
2. 部署控制磁盘IOPS的CRD，将Pod的吞吐进行限制。

   使用以下模板样例，分别把设备 `/dev/vdb1` 的 `BPS` 限制为：1048576、2097152、3145728。

```
apiVersion: resources.alibabacloud.com/v1alpha1
kind: Cgroups
metadata:
 name: cgroups-sample-fio
spec:
 pod:
  name: fio-6b6c469fdf-44h7v
  namespace: default
  containers:
  - name: fio
   blkio:
    device_write_bps: [{device: "/dev/vdb1", value: "1048576"}]
```

3. 查看节点 cn-beijing.192.168.0.182 上的磁盘监控，如下图所示。





如上图所示，说明Pod的吞吐 BPS 和预期保持了一致，且修改过程中Pod没有被重启。

CPU拓扑感知调度

resource-controller结合ACK的调度器可以在多核物理机（Intel、AMD和ARM）上实现更优的CPU绑定和自动选择功能。具体操作，请参见CPU拓扑感知调度。

CPU拓扑感知调度实现对物理核心和超线程的合理使用，避免L1或L2换入换出，以及跨NUMA的片外传输和L3的频繁刷新，实现对多线程CPU密集应用的最大化CPU使用效率。关于更多CPU拓扑感知调度能力，请参见ACK团队在KubeCon 2020的演讲《Practice of Fine-grained Cgroups Resources Scheduling in Kubernetes》。

相关文档

- CPU拓扑感知调度

19.1.3. 通过控制L3 cache和MBA提高不同优先级任务的隔离能力

不同优先级的任务通常会被部署在同一台机器上，这将导致不同优先级任务之间的资源竞争问题，且无法保证高优先级任务的服务质量QoS（Quality of Service）。RDT（Resource Director Technology）通过控制L3 cache（Last Level Cache，即三级缓存LLC）和MBA（Memory Bandwidth Allocation）内存带宽，对不同优先级的任务进行隔离，从而保证高优先级任务的QoS，同时提高整体的资源利用率。本文主要介绍如何通过控制L3 cache和MBA提高不同优先级任务的隔离能力。

前提条件

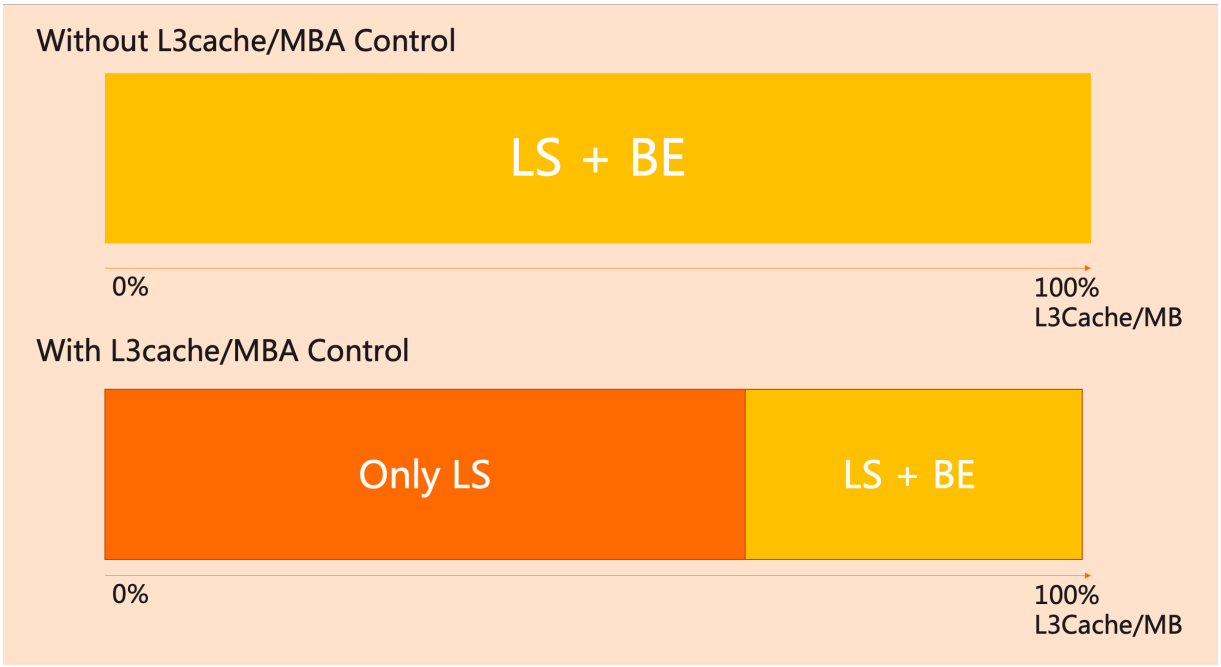
已安装resource-controller组件。更多信息，请参见resource-controller。

背景信息

RDT提供了一个由多个组件功能（包括CMT、CAT、CDP、MBM和MBA）组成的框架，用于L3 cache和内存的监控及分配功能。该框架可以跟踪和控制平台上同时运行的多个应用程序、容器或VM使用的共享资源，例如：L3 cache和主内存（DRAM）带宽。RDT可以帮助检测吵闹的邻居并减少性能干扰，从而确保复杂环境中关键工作负载的性能。更多信息，请参见resource-director-technology。

为了充分利用机器中的资源，通常会将高优先延迟敏感性LS（Latency-Sensitive）和低优先尽力而为BE（Best-Effort）的任务部署在同一台机器上，但这将导致两种不同优先级任务之间的资源竞争问题。若不加隔离控制，这两种不同优先级任务将平等地抢占L3 cache和MBA内存带宽等资源，从而无法保证给LS业务分配足够的资源，使其性能受到影响。

因此，基于RDT resource-controller提供了对不同优先级任务的隔离能力。通过resource-controller提供的L3 cache和MBA控制功能可以限制BE任务对L3 cache和MBA内存带宽的最大使用量，维持高优先延迟敏感性LS类型负载的资源占用率，保证业务运行的高效稳定。同时，resource-controller允许动态地修改对低优先尽力而为BE类型负载的资源限制程度，以适应不同的部署场景，进一步提高整体的资源使用率。



操作步骤

1. 使用以下YAML样例，创建名称为*resource-controller-configmap.yaml*的文件。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: resource-controller-config
 namespace: kube-system
data:
 default.qosClass: LS #LS（Latency Sensitive with High priority scheduling)、BE(Best Effort with Low priority scheduling)
  default.cpushare.qos.low.namespaces: "besteffort-ns1, besteffort-ns2" # By default NULL, suppress existing BE task within specified namespaces immediately.
 default.cpushare.qos.low.l3-percent: "10" #optional
 default.cpushare.qos.low.mb-percent: "20" #optional
```

其中：

- `default.cpushare.qos.low.l3-percent`：表示BE应用的最大可用L3 cache块数占总L3 cache块数的百分比。
- `default.cpushare.qos.low.mb-percent`：表示BE应用的最大可用内存带宽占全部内存带宽的百分比。
- `default.cpushare.qos.low.mb-percent` 和 `default.cpushare.qos.low.l3-percent`：均为可选设置，对所有的BE任务生效，实际设置的L3 cache与MBA内存带宽百分比值精度由应用部署所在的机器决定。

> **说明** 增加或修改 `resource-controller-config` 中的 `default.cpushare.qos.low.l3-percent` 或 `default.cpushare.qos.low.mb-percent` 会对目前已有的以及此后创建的所有BE应用生效。

2. 部署应用，且在Annotation处打标。

   使用以下YAML样例，部署一个低优先级的视频转码应用，部署任务前把视频上传至所在节点的*/root/mp4*文件夹内，并修改YAML文件中对应的视频名称。

```
apiVersion: batch/v1
kind: Job
metadata:
 generateName: ffmpeg-convert-mp4-job-nomi-real-
spec:
 backoffLimit: 1
 parallelism: 1
 template:
  metadata:
   annotations:
    alibabacloud.com/qosClass: "BE"  #低优先级别应用应设置为BE。
    alibabacloud.com/cache-qos: "low" #optional, 可设置为low、middle、high。
  spec:
   containers:
   - image: jrottenberg/ffmpeg
    name: ffmpeg
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - mountPath: /root/mp4
     name: mp4-volume
    resources:
     requests:
      cpu: "4"
      memory: 6Gi
     limits:
      cpu: "4"
      memory: 6Gi
    command: ["/bin/sh"]
    args: ["-c","while true;do ffmpeg -i /root/mp4/*.mp4 -c:v libx264 -crf 2 -threads 2 destination.flv -y;sleep 1;done"] #将*.mp4修改为待转码视频名称
   volumes:
   - name: mp4-volume
    hostPath:
     path: /root/mp4
     type: Directory
   dnsPolicy: ClusterFirst
   restartPolicy: OnFailure
```

其中：

- alibabacloud.com/qosClass：表示该任务的性质，默认值在*resource-controller-configmap.yaml*的default.qosClass中设置，高优先延迟敏感性负载应设置为LS，低优先尽力而为类型负载应设置为BE。
- alibabacloud.com/cache-qos：表示BE任务的三个默认配置：
  - 设置为low时：l3-percent=10%,mb-percent=20%。
  - 设置为middle时：l3-percent=30%,mb-percent=50%。
  - 设置为high时：l3-percent=50%,mb-percent=100%。
- 默认配置中的l3-percent和mb-percent：
  - 只会在*resource-controller-configmap.yaml*中default.cpushare.qos.low.l3-percent为空时生效。
  - *resource-controller-configmap.yaml*对应值更新为空时，BE任务的资源将自动按照alibabacloud.com/cache-qos值恢复默认配置。

> **说明** 以任何方式设置的l3-percent或mb-percent均只对BE任务生效，LS任务的l3-percent和mb-percent的值恒等于100%。

19.1.4. 通过控制动态水位提高不同优先级负载的资源利用效率

为了充分利用节点资源，通常会将不同优先级的负载部署在同一节点上，且不同优先级负载在不同时间段占用的资源量不同。通过动态控制不同优先级的负载资源使用量及资源水位，可以提升负载的运行效率。本文主要介绍如何通过动态水位、定时任务CronJob及强隔离策略提高不同优先级负载的资源利用效率。

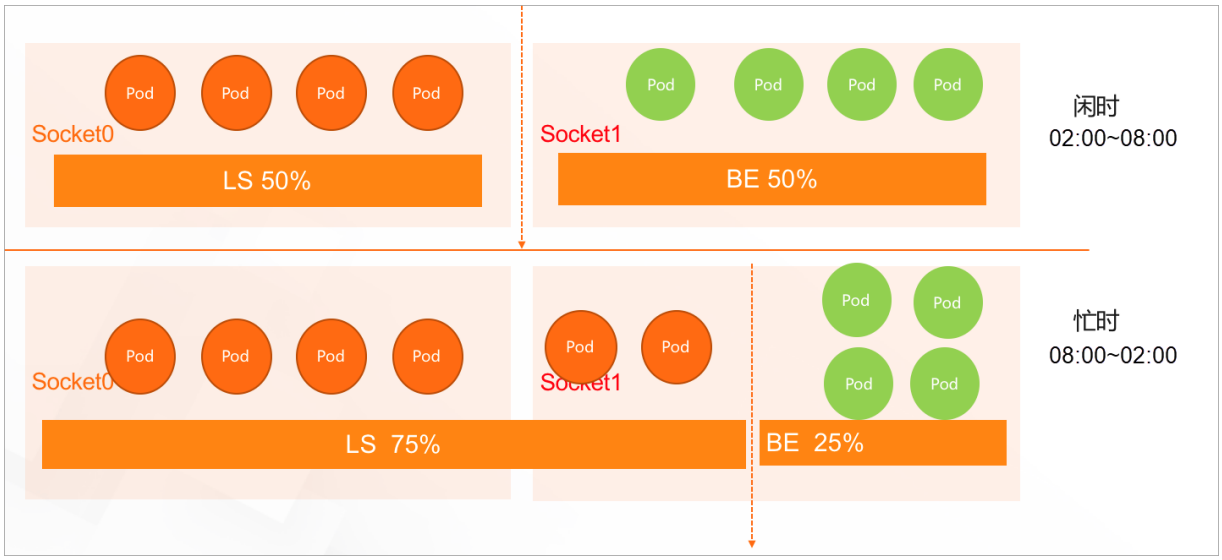
前提条件

已安装resource-controller组件。更多信息，请参见resource-controller。

背景信息

为了充分利用节点资源，通常会将不同优先级的负载部署在同一节点上，比如高优先延迟敏感性（LS）和低优先尽力而为（BE）的负载。但是不同的负载往往具有不同的高峰期，不同时间段需要占用的资源量不一样。例如，高优先的延迟敏感型负载（LS）在白天资源量需求比较大，而在夜晚就比较少。

通过控制动态水位，可以在白天降低低优先BE类型负载的资源使用量，控制其资源水位，保证高优先LS类型负载的资源使用量（即QoS）。在夜晚可以提高低优先BE类型负载的资源使用量及资源水位，让其充分利用空闲资源，从而提高整体的资源利用率。



使用动态水位控制低优先级任务的使用资源量

1. 创建ConfigMap。

i. 使用以下内容，创建*resource-controller-config.yaml*文件。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: resource-controller-config
 namespace: kube-system
data:
 default.qosClass: LS #LS（Latency Sensitive with High priority scheduling)、BE(Best Effort with Low priority scheduling).
 default.cpushare.qos.low.cpu-watermark: "50" # 25、50、100, set maximum water mark of workload of low qos.
 default.cpushare.qos.low.namespaces: "besteffort-ns1, besteffort-ns2" # By default NULL, suppress existing workload within specified namespaces immediately.
 default.cpushare.qos.high.force-reserved: "false" # By default false, true: turn on this switch to reserve dedicated CPUs for high priority tasks (LS).
```

参数	说明
default.cpushare.qos.low.cpu-watermark	表示低优先级任务可以占用整机资源的百分比。
default.cpushare.qos.low.namespaces	表示需要控制的低优先级任务的命名空间。
default.cpushare.qos.high.force-reserved	表示是否将高、低优先级的任务完全隔离，分别部署在不同的CPU上。 该配置在创建之后生效，对于之前已经部署的任务，只对低优先级任务生效，高优先级任务保持不变。

ii. 执行以下命令创建ConfigMap。

```
kubectl create -f resource-controller-config.yaml
```

2. 创建应用。

本示例部署一个低优先级的计算Pi的任务，在 `annotations` 下打标。

i. 使用以下内容，创建*cal-pi.yaml*文件。

```
apiVersion: v1
kind: Pod
metadata:
 name: cal-pi
 annotations:
  alibabacloud.com/qosClass: "BE"
spec:
 restartPolicy: Never
 containers:
 - image: registry.cn-zhangjiakou.aliyuncs.com/xianlu/java-pi
  name: cal-pi
  resources:
   requests:
    cpu: 4
   limits:
    cpu: 4 #需要设置resources.limit.cpu值。
  env:
  - name: limit
   value: "20000"
  - name: threadNum
   value: "3000"
```

`alibabacloud.com/qosClass` 表示该任务的性质：

- 该字段的默认值在*resource-controller-configmap.yaml*的 `default.qosClass` 中设置，高优先延迟敏感性负载应设置为 `LS` ，低优先尽力而为类型负载应设置为 `BE` 。
- 您也可以在以下场景中对该字段进行设置：
  - Pod的 `metadata`
  - Deployment的 `spec.metada`
  - Daemonset、Statefulset或Job的 `template.metadata`

ii. 执行以下命令创建应用。

```
kubectl create -f cal-pi.yaml
```

使用CronJob分时策略控制低优先级任务的使用资源量

通过定时任务CronJob配置低优先级任务水位，即修改 `schedule` （执行时间）和 `json` （配置文件）文件可以定时控制低优先级任务水位。

本示例，在早上七点以后高优先任务优先，把所有低优先BE任务动态压缩到CPU利用率为25%的水位。

1. 使用以下内容，创建*mix-strategy-1.yaml*文件。

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
 name: mix-strategy-1
 namespace: kube-system
spec:
 schedule: "0 7 * * *"
 jobTemplate:
  spec:
   completions: 1
   parallelism: 1
   template:
    spec:
     containers:
      - args:
       - $(JSON_INPUT)
       env:
       - name: JSON_INPUT
        value:
          '{
           "apiVersion":"v1",
           "kind":"ConfigMap",
           "metadata":{
            "name":"resource-controller-config",
            "namespace":"kube-system"
           },
           "data":{
            "default.cpushare.qos.high.force-reserved": "false",
            "default.cpushare.qos.low.cpu-watermark":"25",
            "default.qosClass":"LS",
            "default.cpushare.qos.low.namespaces": "low-qos-workload-namespace"
           }
           }
          '
       name: strategy
       command: ["change-config"]
       image: registry.cn-hangzhou.aliyuncs.com/shuangkun/change-config:v1
       imagePullPolicy: Always
     serviceAccount: admin
     restartPolicy: OnFailure
```

2. 执行以下命令，创建定时任务。

```
kubectl create -f mix-strategy-1.yaml
```

同样的，可以通过定时任务设置其他低优先级任务在任何时间段的水位，进而提高整体资源利用率。

使用强隔离策略控制任务的使用资源量

当低优先级任务的CPU利用率低于50%，且高优先级任务的CPU利用率也低于50%时，推荐使用强制隔离。配置参数 `default.cpushare.qos.high.force-reserved` 为 `true` ，配置 `default.cpushare.qos.low.cpu-watermark` 为50，将高、低优先级任务分隔在不同的NUMA和CCX上，能为高优先任务提供更好的性能环境。

1. 使用以下内容，修改*resource-controller-config.yaml*文件。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: resource-controller-config
 namespace: kube-system
data:
 default.qosClass: LS #LS（Latency Sensitive with High priority scheduling)、BE(Best Effort with Low priority scheduling)
 default.cpushare.qos.low.cpu-watermark: "50" # 25、50、100, set maximum water mark of workload of low qos
 default.cpushare.qos.low.namespaces: "besteffort-ns1, besteffort-ns2" # By default NULL, suppress existing workload within specified namespaces immediately.
 default.cpushare.qos.high.force-reserved: "true" # By default false, true: turn on this switch to reserve dedicated CPUs for high priority tasks (LS)
```

2. 执行以下命令修改ConfigMap。

```
kubectl apply -f resource-controller-config.yaml
```

19.2. GPU调度

本文介绍调度GPU资源的不同方法，包括使用Kubernetes默认GPU调度、提升GPU资源使用率的共享GPU调度、以及加速任务执行的GPU拓扑感知调度。

普通GPU调度

申请Kubernetes GPU集群后，通过运行TensorFlow的GPU实验环境，关于如何使用Kubernetes默认调度器独占GPU，请参见使用Kubernetes默认GPU调度。

您也可通过使用GPU节点标签自定义调度，具体操作，请参见使用节点标签自定义GPU调度。

共享GPU调度

阿里云容器服务Kubernetes版ACK（Container Service for Kubernetes）开源了GPU共享调度之后，您能在阿里云、AWS、GCE和自己数据中心的容器集群上通过GPU共享调度框架实现多个容器运行在同一个GPU设备上的目标。ACK开源GPU共享调度降低了使用GPU的经济成本。通过隔离，限制运行在同一个GPU上的多个容器能够按照自己申请的资源使用量运行，避免因为其资源用量超标影响同一个GPU上的其他容器的正常工作，同时也使更小颗粒度的使用GPU提供了可能。想进一步了解：

关于什么是共享GPU调度，请参见共享GPU调度概述。

关于如何开启共享GPU，请参见安装共享GPU组件；关于如何关闭共享GPU显存隔离能力，请参见关闭共享GPU显存隔离能力。

关于如何使用共享GPU调度，请参见运行共享GPU示例、监控和隔离GPU资源、基于节点池管理共享GPU。

GPU拓扑感知调度

Kubernetes对节点的GPU拓扑信息不感知，调度过程中对GPU的选择比较随机，选择不同的GPU组合训练速度会存在较大的差异。基于以上问题，ACK基于Scheduling Framework机制，实现GPU拓扑感知调度，在节点的GPU组合中选择具有最优训练速度的组合。关于如何使用GPU拓扑感知调度，请参见以下文档：

- GPU拓扑感知调度概述
- 安装GPU拓扑感知组件
- Tensorflow分布式训练使用GPU拓扑感知调度
- Pytorch分布式训练使用GPU拓扑感知调度

19.3. FPGA调度

19.3.1. 利用阿里云Kubernetes的FPGA节点标签进行调度

在使用Kubernetes集群实现FPGA计算时，为了有效利用FPGA设备，可根据需要将应用调度到具有FPGA设备的节点上。本文介绍根据FPGA节点标签进行节点调度。

前提条件

- 您已成功创建一个拥有FPGA节点的Kubernetes集群。具体操作，请参见创建托管FPGA集群。
- 您已连接到Kubernetes集群，方便快速查看节点标签等信息。具体操作，请参见通过kubectl工具连接集群。

背景信息

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

- 可以快速筛选FPGA节点。
- 部署时可以作为调度条件使用。

步骤一：查看FPGA节点的标签

方式一：通过控制台查看FPGA节点的标签

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点列表**页面中，选择FPGA节点，单击**操作**列的**更多 > 详情**。

   查看FPGA节点的标签。



方式二：通过kubectl命令查看FPGA节点的标签

1. 执行以下命令查看FPGA节点详情。

   ```
   kubectl get nodes
   ```

   预期输出：

   ```
   NAME          STATUS  ROLES   AGE   VERSION
   cn-beijing.192.168.XX.X1  Ready   <none>  3h51m  v1.18.8-aliyun.1
   cn-beijing.192.168.XX.X2  Ready   <none>  3h41m  v1.18.8-aliyun.1
   ```

2. 选择一个FPGA节点，执行以下命令查看该FPGA节点的标签。

```
kubectl describe node cn-beijing.192.168.XX.X2
```

预期输出：

```
Name:         cn-beijing.192.168.XX.X2
Roles:        <none>
Labels:       ack.aliyun.com=c05888610e***
              alibabacloud.com/nodepool-id=npfda879b6***
              beta.kubernetes.io/arch=amd64
              beta.kubernetes.io/instance-type=ecs.f3-c4f1.xlarge
              beta.kubernetes.io/os=linux
              failure-domain.beta.kubernetes.io/region=cn-beijing
              failure-domain.beta.kubernetes.io/zone=cn-beijing-h
              fpga.k8s.aliyun.com=f3
              kubernetes.io/arch=amd64
              kubernetes.io/hostname=cn-beijing.192.168.XX.X2
              kubernetes.io/os=linux
              node.kubernetes.io/instance-type=ecs.f3-c4f1.xlarge
              topology.diskplugin.csi.alibabacloud.com/zone=cn-beijing-h
              topology.kubernetes.io/region=cn-beijing
              topology.kubernetes.io/zone=cn-beijing-h
```

从预期输出可得，该FPGA节点包含节点标签（NodeLabel）`fpga.k8s.aliyun.com=f3`，下方示例使用该节点标签实现应用调度。

步骤二：调度应用到FPGA节点

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面中，单击**使用YAML创建资源**。
6. 选择自定义示例模板，将以下示例复制到**模板**中。

```
apiVersion: batch/v1
kind: Job
metadata:
 name: fpga-run-task1
spec:
 backoffLimit: 0
 completions: 1
 parallelism: 1
 template:
  spec:
   nodeSelector:
    fpga.k8s.aliyun.com: f3
   containers:
   - image: <your image>
    imagePullPolicy: Always
    name: fpga-run-task1
    resources:
     limits:
      xilinx.com/fpga-aliyun-f3: 1
    securityContext:
     privileged: true
```

> **说明** <your image> 替换为您自定义的应用镜像地址。

7. 单击**创建**。
   在集群管理页左侧导航栏选择**工作负载 > 容器组**。
   在容器组列表中，您可看到示例的Pod（容器组）已被成功调度到对应的节点上，从而实现基于FPGA节点标签的灵活调度。

19.4. 任务调度

19.4.1. Gang scheduling

ACK基于新版的Kube-scheduler框架实现Gang scheduling的能力，解决原生调度器无法支持All-or-Nothing作业调度的问题。本文举例说明如何使用Gang scheduling能力。

前提条件

- 您已创建ACK Pro版集群。具体步骤，请参见创建ACK Pro版集群。

> **注意** 目前Gang scheduling仅支持ACK Pro托管版集群。如果您需要专有版集群，请提交工单申请白名单。

- 系统组件版本要求具体如下表所示。

组件	版本要求
Kubernetes	1.16及以上
Helm版本	3.0及以上版本
Docker版本	19.03.5
操作系统	CentOS 7.6、CentOS 7.7、Ubuntu 16.04、Ubuntu 18.04、Alibaba Cloud Linux 2

背景信息

Gang scheduling是在并发系统中将多个相关联的进程调度到不同处理器上同时运行的策略，其最主要的原则是保证所有相关联的进程能够同时启动，防止部分进程的异常，导致整个关联进程组的阻塞。例如，您提交一个批量Job，这个批量Job包含多个任务，要么这多个任务全部调度成功，要么一个都调度不成功。这种All-or-Nothing调度场景，就被称作Gang scheduling。

Kubernetes目前已经广泛的应用于在线服务编排，为了提升集群的利用率和运行效率，ACK希望将Kubernetes作为一个统一的管理平台来管理在线服务和离线作业。但是由于调度器的限制，使得一些离线的工作负载无法迁移到Kubernetes。例如对于有All-or-Nothing特点的作业，它要求所有的任务在同一时间被调度，如果只是部分任务启动的话，启动的任务将持续等待剩余的任务被调度。在最坏的情况下，所有作业都处于挂起状态，从而导致死锁。为了解决这个问题，需要对调度器支持Gang scheduling。

功能介绍

ACK将一组需要同时调度的Pod称为PodGroup。您如果在提交All-or-Nothing作业时，可以通过设置labels字段的形式指定所属PodGroup的名称以及保证作业正常运行Task的最少运行个数。调度器会根据您指定的最少运行个数进行调度，只有当集群资源满足该Task最少运行个数时，才会统一调度，否则作业将一直处于Pending状态。

使用方式

使用Gang scheduling时，在创建的Pod处通过设置labels的形式配置min-available和name。

```
labels:
  pod-group.scheduling.sigs.k8s.io/name: tf-smoke-gpu
  pod-group.scheduling.sigs.k8s.io/min-available: "3"
```

- name：用于表示PodGroup的名称。
- min-available：用于表示当前集群资源至少满足最少可运行Pod数时，才能统一调度创建。

> **说明** 要求属于同一个PodGroup的Pod必须保持相同的优先级。

示例

文本通过运行Tensorflow的分布式作业来展示Gang scheduling的效果。当前测试集群有4个GPU卡。

1. 执行以下命令通过Kubeflow的Arena工具在已有Kubernetes集群中部署Tensorflow作业运行环境。

   > **说明** Arena是基于Kubernetes的机器学习系统开源社区Kubeflow中的子项目之一。Arena用命令行和SDK的形式支持了机器学习任务的主要生命周期管理（包括环境安装、数据准备，到模型开发、模型训练、模型预测等），有效提升了数据科学家工作效率。

   ```
   git clone https://github.com/kubeflow/arena.git
   kubectl create ns arena-system
   kubectl create -f arena/kubernetes-artifacts/jobmon/jobmon-role.yaml
   kubectl create -f arena/kubernetes-artifacts/tf-operator/tf-crd.yaml
   kubectl create -f arena/kubernetes-artifacts/tf-operator/tf-operator.yaml
   ```

   执行以下命令检查Tensorflow作业运行环境是否成功部署。当出现Running状态时，说明成功部署。

   ```
   kubectl  get pods -n arena-system
   ```

   ```
   NAME                READY  STATUS   RESTARTS  AGE
   tf-job-dashboard-56cf48874f-gwlhv  1/1   Running  0     54s
   tf-job-operator-66494d88fd-snm9m   1/1   Running  0     54s
   ```

2. 使用以下模板向集群中提交Tensorflow分布式作业，含有1个PS和4个Worker，每个Worker类型的Pod需要2个GPU。

```
apiVersion: "kubeflow.org/v1"
kind: "TFJob"
metadata:
 name: "tf-smoke-gpu"
spec:
 tfReplicaSpecs:
  PS:
   replicas: 1
   template:
    metadata:
     creationTimestamp: null
     labels:
      pod-group.scheduling.sigs.k8s.io/name: tf-smoke-gpu
      pod-group.scheduling.sigs.k8s.io/min-available: "5"
    spec:
     containers:
     - args:
      - python
      - tf_cnn_benchmarks.py
      - --batch_size=32
      - --model=resnet50
      - --variable_update=parameter_server
      - --flush_stdout=true
      - --num_gpus=1
      - --local_parameter_device=cpu
      - --device=cpu
      - --data_format=NHWC
      image: registry.cn-hangzhou.aliyuncs.com/kubeflow-images-public/tf-benchmarks-cpu:v20171202-bdab
599-dirty-284af3
      name: tensorflow
      ports:
      - containerPort: 2222
       name: tfjob-port
      resources:
       limits:
        cpu: '1'
      workingDir: /opt/tf-benchmarks/scripts/tf_cnn_benchmarks
     restartPolicy: OnFailure
  Worker:
   replicas: 4
   template:
    metadata:
     creationTimestamp: null
     labels:
      pod-group.scheduling.sigs.k8s.io/name: tf-smoke-gpu
      pod-group.scheduling.sigs.k8s.io/min-available: "5"
    spec:
     containers:
     - args:
      - python
      - tf_cnn_benchmarks.py
      - --batch_size=32
      - --model=resnet50
      - --variable_update=parameter_server
      - --flush_stdout=true
      - --num_gpus=1
      - --local_parameter_device=cpu
      - --device=gpu
```

```
      - --data_format=NHWC
      image: registry.cn-hangzhou.aliyuncs.com/kubeflow-images-public/tf-benchmarks-gpu:v20171202-bdab
599-dirty-284af3
      name: tensorflow
      ports:
      - containerPort: 2222
        name: tfjob-port
      resources:
        limits:
          nvidia.com/gpu: 2
      workingDir: /opt/tf-benchmarks/scripts/tf_cnn_benchmarks
    restartPolicy: OnFailure
```

- 不使用Gang scheduling功能
  执行以下命令查看Pod状态。集群中只用2个Worker类型的Pod在运行，剩余2个处于Pending状态。

  ```
  kubectl get pods
  ```

  ```
  NAME          READY  STATUS   RESTARTS  AGE
  tf-smoke-gpu-ps-0    1/1   Running  0     6m43s
  tf-smoke-gpu-worker-0  1/1   Running  0     6m43s
  tf-smoke-gpu-worker-1  1/1   Running  0     6m43s
  tf-smoke-gpu-worker-2  0/1   Pending  0     6m43s
  tf-smoke-gpu-worker-3  0/1   Pending  0     6m43s
  ```

  执行以下命令查看其中正在运行的Worker类型Pod的日志。处于等待剩余两个Worker类型Pod启动的状态，GPU占用却没有使用。

  ```
  kubectl logs -f tf-smoke-gpu-worker-0
  ```

  ```
  INFO|2020-05-19T07:02:18|/opt/launcher.py|27| 2020-05-19 07:02:18.199696: I tensorflow/core/distributed_r
  untime/master.cc:221] CreateSession still waiting for response from worker: /job:worker/replica:0/task:3
  INFO|2020-05-19T07:02:28|/opt/launcher.py|27| 2020-05-19 07:02:28.199798: I tensorflow/core/distributed_r
  untime/master.cc:221] CreateSession still waiting for response from worker: /job:worker/replica:0/task:2
  ```

- 使用Gang scheduling功能时
  执行以下命令查看Pod状态。因为集群的资源无法满足设定的最小数要求，则PodGroup无法正常调度，所有的Pod一直处于Pending状态。

  ```
  kubectl get pods
  ```

  ```
  NAME          READY  STATUS   RESTARTS  AGE
  tf-smoke-gpu-ps-0    0/1   Pending  0     43s
  tf-smoke-gpu-worker-0  0/1   Pending  0     43s
  tf-smoke-gpu-worker-1  0/1   Pending  0     43s
  tf-smoke-gpu-worker-2  0/1   Pending  0     43s
  tf-smoke-gpu-worker-3  0/1   Pending  0     43s
  ```

  当集群中新增4个GPU卡的资源，则当前集群的资源满足设定的最小数要求，则PodGroup正常调度，4个Worker类型Pod开始运行。执行以下命令查看Pod状态。

  ```
  kubectl get pods
  ```

```
NAME          READY  STATUS   RESTARTS  AGE
tf-smoke-gpu-ps-0    1/1   Running  0     3m16s
tf-smoke-gpu-worker-0  1/1   Running  0     3m16s
tf-smoke-gpu-worker-1  1/1   Running  0     3m16s
tf-smoke-gpu-worker-2  1/1   Running  0     3m16s
tf-smoke-gpu-worker-3  1/1   Running  0     3m16s
```

执行以下命令查看其中一个Worker类型Pod的日志，显示训练任务已经开始。

```
kubectl logs -f tf-smoke-gpu-worker-0
```

```
INFO|2020-05-19T07:15:24|/opt/launcher.py|27| Running warm up
INFO|2020-05-19T07:21:04|/opt/launcher.py|27| Done warm up
INFO|2020-05-19T07:21:04|/opt/launcher.py|27| Step  Img/sec loss
INFO|2020-05-19T07:21:05|/opt/launcher.py|27| 1 images/sec: 31.6 +/- 0.0 (jitter = 0.0) 8.318
INFO|2020-05-19T07:21:15|/opt/launcher.py|27| 10  images/sec: 31.1 +/- 0.4 (jitter = 0.7) 8.343
INFO|2020-05-19T07:21:25|/opt/launcher.py|27| 20  images/sec: 31.5 +/- 0.3 (jitter = 0.7) 8.142
```

19.4.2. Capacity Scheduling

通过Kubernetes原生的ResourceQuota方式进行固定资源分配，集群的整体资源利用率不高。阿里云借鉴Yarn Capacity Scheduling的设计思路，基于Scheduling Framework的扩展机制，在调度侧通过引入弹性配额组实现了Capacity Scheduling功能，在确保用户资源分配的基础上通过资源共享的方式来提升集群的整体资源利用率。本文介绍如何使用Capacity Scheduling功能。

前提条件

- 创建ACK Pro版集群。
- 集群的Kubernetes版本不低于1.20。

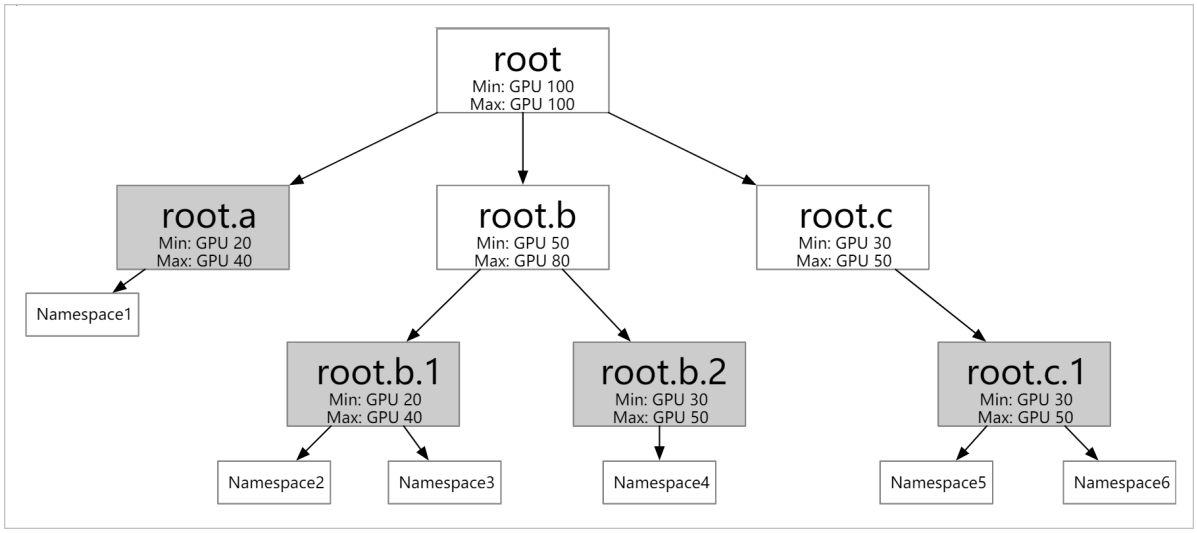
背景信息

当集群中有多个用户时，为了保障用户有足够的资源使用，管理员会将集群的资源固定分配给不同的用户。传统的方法是通过Kubernetes原生的ResourceQuota方式进行固定资源的分配。因为不同的用户使用资源的周期和方式不同，可能会出现在某一时刻，一些用户的资源不够用，而另一些用户的资源闲置。如果出现多个类似的情况，则整个集群会有很多资源浪费，导致集群的整体资源利用率下降。

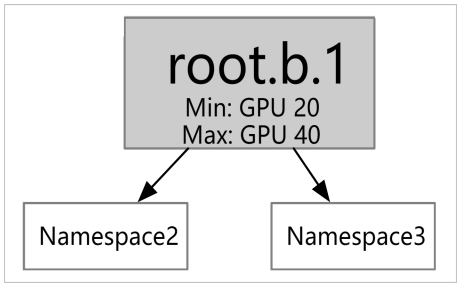
Capacity Scheduling核心功能

针对上述的问题，阿里云基于Scheduling Framework的扩展机制，在调度侧实现了Capacity Scheduling的功能，在确保用户的资源分配的基础上通过资源共享的方式来提升整体资源的利用率。Capacity Scheduling有以下功能：

- 支持定义不同层级的资源配额。如下图所示，您可以根据具体情况（比如：公司的组织结构）配置多个层级的弹性配额。弹性配额组的叶子节点可以对应多个Namespace，但同一个Namespace只能归属于一个叶子节点。



- 支持不同弹性配额之间的资源借用和回收。
  - Min：您可以使用的保障资源（Guaranteed Resource）。当整个集群资源紧张时，所有用户使用的Min总和需要小于集群的总资源量。
  - Max：您可以使用的资源上限。



> **说明**
>
> - 您的工作负载可以向其他用户借用其未使用的资源额度，但借用后可使用的资源总量依然不能超过Max。
> - 您未使用的Min资源配额可以被其他用户借用，但当该用户需要使用资源时，可以通过抢占的方式回收借出的资源。

- 支持多种资源的配置。您除了可以配置CPU和内存资源，也可以配置如GPU等任何Kubernetes支持的扩展资源（Extended Resource）。

使用capacity scheduling功能的使用示例

本文测试集群节点资源为1台ecs.sn2.13xlarge机器（56 vCPU 224 GiB）。

1. 执行以下命令，创建对应的Namespace。

```
kubectl create ns namespace1
kubectl create ns namespace2
kubectl create ns namespace3
kubectl create ns namespace4
```

2. 使用以下YAML文件样例，创建相应的弹性配额组。

```
apiVersion: scheduling.sigs.k8s.io/v1beta1
kind: ElasticQuotaTree
metadata:
  name: elasticquotatree
```

```
  name: elasticquotatree
  namespace: kube-system # 只有kube-system下才会生效
spec:
  root:
    name: root # Root节点的Max必须等于Min
    max:
      cpu: 40
      memory: 40Gi
      nvidia.com/gpu: 4
    min:
      cpu: 40
      memory: 40Gi
      nvidia.com/gpu: 4
    children: # 配置第二级
      - name: root.a
        max:
          cpu: 40
          memory: 40Gi
          nvidia.com/gpu: 4
        min:
          cpu: 20
          memory: 20Gi
          nvidia.com/gpu: 2
        children: # 配置第三级
          - name: root.a.1
            namespaces: # 配置对应的Namespace
              - namespace1
            max:
              cpu: 20
              memory: 20Gi
              nvidia.com/gpu: 2
            min:
              cpu: 10
              memory: 10Gi
              nvidia.com/gpu: 1
          - name: root.a.2
            namespaces: # 配置对应的Namespace
              - namespace2
            max:
              cpu: 20
              memory: 40Gi
              nvidia.com/gpu: 2
            min:
              cpu: 10
              memory: 10Gi
              nvidia.com/gpu: 1
      - name: root.b
        max:
          cpu: 40
          memory: 40Gi
          nvidia.com/gpu: 4
        min:
          cpu: 20
          memory: 20Gi
          nvidia.com/gpu: 2
        children: # 配置第三级
          - name: root.b.1
            namespaces: # 配置对应的Namespace
```

```
    - namespace3
   max:
    cpu: 20
    memory: 20Gi
    nvidia.com/gpu: 2
   min:
    cpu: 10
    memory: 10Gi
    nvidia.com/gpu: 1
  - name: root.b.2
   namespaces: # 配置对应的Namespace
    - namespace4
   max:
    cpu: 20
    memory: 20Gi
    nvidia.com/gpu: 2
   min:
    cpu: 10
    memory: 10Gi
    nvidia.com/gpu: 1
```

根据以上YAML文件示例配置，在 `namespaces` 字段配置对应的命名空间，在 `children` 字段配置对应子级的弹性配额。配置需要满足以下几点：

- 同一个弹性配额中的Min≤Max。
- 子级弹性配额的Min之和≤父级的Min。
- Root节点的Min=Max≤集群总资源。
- Namespace只与弹性配额的叶子节点有一对多的对应关系，且同一个Namespace只能归属于一个叶子节点。

3. 执行以下命令，查看弹性配额组是否创建成功。

```
kubectl get ElasticQuotaTree -n kube-system
```

预期输出：

```
NAME          AGE
elasticquotatree  68s
```

4. 使用以下YAML文件样例，在 `namespace1` 中部署服务，Pod的副本数为5个，每个Pod请求CPU资源量为5核。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx1
 namespace: namespace1
 labels:
  app: nginx1
spec:
 replicas: 5
 selector:
  matchLabels:
   app: nginx1
 template:
  metadata:
   name: nginx1
   labels:
    app: nginx1
  spec:
   containers:
   - name: nginx1
    image: nginx
    resources:
     limits:
      cpu: 5
     requests:
      cpu: 5
```

5. 执行以下命令，查看集群Pod的部署情况。

```
kubectl get pods -n namespace1
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx1-744b889544-52dbg  1/1   Running  0     70s
nginx1-744b889544-6l4s9  1/1   Running  0     70s
nginx1-744b889544-cgzlr  1/1   Running  0     70s
nginx1-744b889544-w2gr7  1/1   Running  0     70s
nginx1-744b889544-zr5xz  0/1   Pending  0     70s
```

   - 因为当前集群的资源有空闲（ root.max.cpu=40 ），当在 namespace1 下的Pod申请CPU资源量超过 root.a.1 设置的 min.cpu=10 时，还可以继续借用其他处于空闲的资源，最多可以申请使用 root.a.1 设置的配额 max.cpu=20 。
   - 当Pod申请CPU资源量超过 max.cpu=20 时，再申请的Pod会处于Pending状态。所以此时申请的5个Pod中，有4个处于Running状态，1个处于Pending状态。

6. 使用以下YAML文件样例，在 namespace2 中部署服务，其中Pod的副本数为5个，每个Pod请求CPU资源量为5核。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx2
 namespace: namespace2
 labels:
  app: nginx2
spec:
 replicas: 5
 selector:
  matchLabels:
   app: nginx2
 template:
  metadata:
   name: nginx2
   labels:
    app: nginx2
  spec:
   containers:
   - name: nginx2
    image: nginx
    resources:
     limits:
      cpu: 5
     requests:
      cpu: 5
```

7. 执行以下命令，查看集群Pod的部署情况。

```
kubectl get pods -n namespace1
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx1-744b889544-52dbg  1/1   Running  0     111s
nginx1-744b889544-6l4s9  1/1   Running  0     111s
nginx1-744b889544-cgzlr  1/1   Running  0     111s
nginx1-744b889544-w2gr7  1/1   Running  0     111s
nginx1-744b889544-zr5xz  0/1   Pending  0     111s
```

```
kubectl get pods -n namespace2
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx2-556f95449f-4gl8s  1/1   Running  0     111s
nginx2-556f95449f-crwk4  1/1   Running  0     111s
nginx2-556f95449f-gg6q2  0/1   Pending  0     111s
nginx2-556f95449f-pnz5k  1/1   Running  0     111s
nginx2-556f95449f-vjpmq  1/1   Running  0     111s
```

- 同 `nginx1` 一样，因为当前集群有资源空闲（ `root.max.cpu=40` ），当在 `namespace2` 下Pod申请CPU资源量超过 `root.a.2` 设置的 `min.cpu=10` 时，还可以继续借用其他处于空闲的资源，最多可以申请使用 `root.a.2` 设置的配额 `max.cpu=20` 。
- 当Pod申请CPU资源量超过 `max.cpu=20` 时，再申请的Pod处于Pending状态。所以此时申请的5个Pod中，有4个处于Running状态，1个处于Pending状态。
- 此时集群中 `namespace1` 和 `namespace2` 中的Pod所占用的资源已经为 `root` 设置的 `root.max.cpu=40` 。

8. 使用以下YAML文件样例，在 namespace3 中部署服务，其中Pod的副本数为5个，每个Pod请求CPU资源量为5核。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx3
 namespace: namespace3
 labels:
  app: nginx3
spec:
 replicas: 5
 selector:
  matchLabels:
   app: nginx3
 template:
  metadata:
   name: nginx3
   labels:
    app: nginx3
  spec:
   containers:
   - name: nginx3
    image: nginx
    resources:
     limits:
      cpu: 5
     requests:
      cpu: 5
```

9. 执行以下命令，查看集群Pod的部署情况。

```
kubectl get pods -n namespace1
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx1-744b889544-52dbg  1/1   Running  0     6m17s
nginx1-744b889544-cgzlr  1/1   Running  0     6m17s
nginx1-744b889544-nknns  0/1   Pending  0     3m45s
nginx1-744b889544-w2gr7  1/1   Running  0     6m17s
nginx1-744b889544-zr5xz  0/1   Pending  0     6m17s
```

```
kubectl get pods -n namespace2
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx2-556f95449f-crwk4  1/1   Running  0     4m22s
nginx2-556f95449f-ft42z  1/1   Running  0     4m22s
nginx2-556f95449f-gg6q2  0/1   Pending  0     4m22s
nginx2-556f95449f-hfr2g  1/1   Running  0     3m29s
nginx2-556f95449f-pvgrl  0/1   Pending  0     3m29s
```

```
kubectl get pods -n namespace3
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx3-578877666-msd7f  1/1    Running  0      4m
nginx3-578877666-nfdwv  0/1    Pending  0      4m10s
nginx3-578877666-psszr  0/1    Pending  0      4m11s
nginx3-578877666-xfsss  1/1    Running  0      4m22s
nginx3-578877666-xpl2p  0/1    Pending  0      4m10s
```

`nginx3` 的弹性配额 `root.b.1` 的 `min` 设置为 `10` ，为了保障其设置的 `min` 资源，调度器会将 `root.a` 下之前借用 `root.b` 的Pod资源归还，使得 `nginx3` 能够至少得到配额 `min.cpu=10` 的资源量，保证其运行。
调度器会综合考虑 `root.a` 下作业的优先级、可用性以及创建时间等因素，选择相应的Pod归还之前抢占的资源（10核）。因此， `nginx3` 得到配额 `min.cpu=10` 的资源量后，有2个Pod处于Running状态，其他3个仍处于Pending状态。

10. 使用以下YAML文件样例，在 `namespace4` 中部署服务 `nginx4` ，其中Pod的副本数为5个，每个Pod请求CPU资源量为5核。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx4
 namespace: namespace4
 labels:
  app: nginx4
spec:
 replicas: 5
 selector:
  matchLabels:
   app: nginx4
 template:
  metadata:
   name: nginx4
   labels:
    app: nginx4
  spec:
   containers:
   - name: nginx4
    image: nginx
    resources:
     limits:
      cpu: 5
     requests:
      cpu: 5
```

11. 执行以下命令，查看集群Pod的部署情况。

```
kubectl get pods -n namespace1
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx1-744b889544-cgzlr  1/1    Running  0      8m20s
nginx1-744b889544-cwx8l  0/1    Pending  0      55s
nginx1-744b889544-gjkx2  0/1    Pending  0      55s
nginx1-744b889544-nknns  0/1    Pending  0      5m48s
nginx1-744b889544-zr5xz  1/1    Running  0      8m20s
```

```
kubectl get pods -n namespace2
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx2-556f95449f-cglpv  0/1   Pending  0     3m45s
nginx2-556f95449f-crwk4  1/1   Running  0     9m31s
nginx2-556f95449f-gg6q2  1/1   Running  0     9m31s
nginx2-556f95449f-pvgrl  0/1   Pending  0     8m38s
nginx2-556f95449f-zv8wn  0/1   Pending  0     3m45s
```

```
kubectl get pods -n namespace3
```

预期输出：

```
NAME          READY  STATUS   RESTARTS  AGE
nginx3-578877666-msd7f  1/1   Running  0     8m46s
nginx3-578877666-nfdwv  0/1   Pending  0     8m56s
nginx3-578877666-psszr  0/1   Pending  0     8m57s
nginx3-578877666-xfsss  1/1   Running  0     9m8s
nginx3-578877666-xpl2p  0/1   Pending  0     8m56s
```

```
kubectl get pods -n namespace4
```

预期输出：

```
nginx4-754b767f45-g9954  1/1   Running  0     4m32s
nginx4-754b767f45-j4v7v  0/1   Pending  0     4m32s
nginx4-754b767f45-jk2t7  0/1   Pending  0     4m32s
nginx4-754b767f45-nhzpf  0/1   Pending  0     4m32s
nginx4-754b767f45-tv5jj  1/1   Running  0     4m32s
```

同理，`nginx4` 的弹性配额 `root.b.2` 的 `min` 设置为 `10`，为了保障其设置的 `min` 资源，调度器会将 `root.a` 下之前借用 `root.b` 的Pod资源归还，使得 `nginx4` 能够至少得到配额 `min.cpu=10` 的资源量，保证其运行。
调度器会综合考虑 `root.a` 下作业的优先级、可用性以及创建时间等因素，选择相应的Pod归还之前抢占的资源（10核）。因此，`nginx4` 得到配额 `min.cpu=10` 资源量后，有2个Pod处于Running状态，其他3个仍处于Pending状态。
此时，集群所有的弹性配额都使用其 `min` 设置的保障资源（Guaranteed Resource）。

19.5. 弹性调度

19.5.1. 使用ECI弹性调度

ECI弹性调度是阿里云提供的弹性调度策略。您可以在部署服务时通过添加Annotations来申明只使用ECS或ECI弹性资源，或者是在ECS资源不足时自动申请ECI资源。通过ECI弹性调度可以满足您在不同工作负载的场景下对弹性资源的不同需求。

前提条件

- 已创建ACK Pro版集群，且集群的Kubernetes版本不低于1.18。具体操作步骤，请参见创建ACK Pro版集群。
- 已在ACK Pro版集群中部署**ack-virtual-node**。具体操作，请参见ACK使用ECI。

操作步骤

> **说明** 您可以通过添加Annotations来申明弹性资源的类型。`alibabacloud.com/burst-resource` 的取值有以下选择：
>
> - 默认不填Annotations时：只使用集群现有的ECS资源。
> - eci：当前集群ECS资源不足时，使用ECI弹性资源。
> - eci_only：只使用ECI弹性资源，不使用集群的ECS资源。

1. 创建并拷贝以下内容到*nginx-deployment.yaml*文件中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 replicas: 4
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   name: nginx
   annotations:
    alibabacloud.com/burst-resource: eci #添加注解，选择弹性调度的资源类型。
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
    resources:
     limits:
      cpu: 2
     requests:
      cpu: 2
```

2. 执行以下命令，创建使用ECI弹性调度的Pod。

```
kubectl apply -f nginx-deployment.yaml
```

19.6. 负载感知调度

负载感知调度是ACK基于Scheduling Framework实现感知节点实际资源负载的调度策略。调度过程中，通过参考节点负载的历史统计，将Pod优先调度到负载较低的节点，实现节点负载均衡的目标，避免出现因单个节点负载过高而导致的应用程序或节点故障。

关于如何使用负载感知调度，请参见使用负载感知调度。

20.命名空间与配额

20.1. 管理命名空间

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。本文为您介绍如何在Kubernetes集群中创建、编辑和删除命名空间。

前提条件

创建Kubernetes托管版集群

背景信息

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

创建命名空间

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，单击**命名空间与配额**。
5. 单击页面右上角的**创建**。
6. 在**创建命名空间**对话框配置命名空间，然后单击**确定**。
   - **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。

   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。

编辑命名空间

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，单击**命名空间与配额**。
5. 在**命名空间**页面单击目标命名空间右侧**操作**列的**编辑**。
6. 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**确定**。

删除命名空间

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，单击**命名空间与配额**。
5. 单击目标命名空间右侧**操作**列的**删除**。
6. 在**提示**对话框中，单击**确定**。
   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。

20.2. 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

- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建命名空间。具体操作，请参见管理命名空间。
- 已通过kubectl连接到Kubernetes集群。具体操作，请参见步骤二：选择集群凭证类型。

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，请参见资源配额。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，单击**命名空间与配额**。
5. 单击目标命名空间右侧**操作**列的**资源配额与限制**。
6. 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）。更多信息，请参见资源配额。

i. 为命名空间配置资源限额（ResourceQuota）。



ii. 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。更多信息，请参见为命名空间配置默认的内存请求和限制。



后续步骤

单击目标命名空间右侧**操作**列的**资源配额与限制**可以查看已配置的资源限制。

21.服务网格ASM

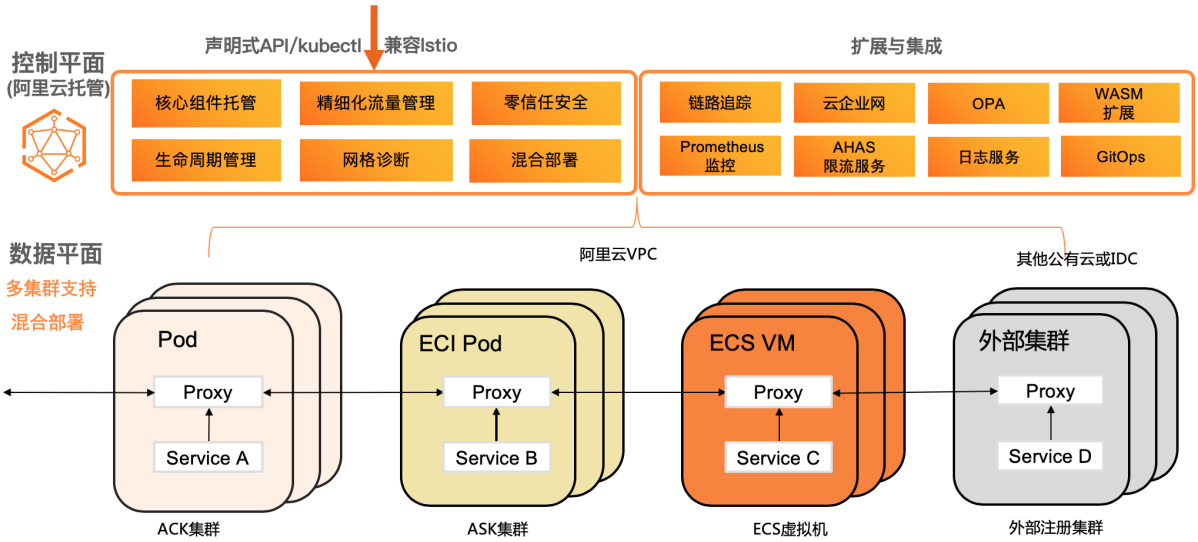
您可以将部署在容器服务Kubernetes版（ACK）中的应用接入阿里云服务网格ASM，简化服务的网格化治理，实现服务的可观测性，减轻开发与运维的工作负担。

背景信息

阿里云服务网格（Alibaba Cloud Service Mesh，简称ASM）提供一个全托管式的服务网格平台，兼容社区Istio开源服务网格，用于简化服务的治理，包括服务调用之间的流量路由与拆分管理、服务间通信的认证安全以及网格可观测性能力。服务网格ASM适用于需要对应用服务进行流量管理、安全管理、故障恢复、观测监控以及微服务架构迁移的应用场景。

产品架构

ASM的产品架构如下图所示。



在ASM中，Istio控制平面的组件全部托管，降低您使用的复杂度，您只需要专注于业务应用的开发部署。同时，保持与Istio社区的兼容，支持声明式的方式定义灵活的路由规则，支持网格内服务之间的统一流量管理。

一个托管了控制平面的ASM实例可以支持来自多个Kubernetes集群的应用服务或者运行于ECI Pod上的应用服务。此外，也可以把一些非Kubernetes服务（例如运行于虚拟机或物理裸机中的服务）集成到同一个服务网格中。

功能特性

ASM定位于混合云、多云、多集群、非容器应用迁移等核心场景中，构建托管式统一的服务网格能力，能够为阿里云用户提供以下功能：

- 一致的管理方式
  以一致的方式来管理运行于ACK托管Kubernetes集群、专有Kubernetes集群、Serverless Kubernetes集群、混合云或多云场景下的接入集群上的应用服务，从而提供一致的可观测性和流量控制。
- 统一的流量管理
  支持容器或者虚拟机混合环境下统一的流量管理。
- 控制平面核心组件托管化
  托管控制平面的核心组件，最大限度地降低用户资源开销和运维成本。

相关操作

关于ASM的入门操作，请参见入门指引。

22.应用市场

22.1. 容器镜像服务ACR

您可以在部署应用时使用容器镜像服务ACR，实现镜像的自动构建和拉取，同时提升镜像拉取速度，减少应用部署时间。

背景信息

容器镜像服务ACR企业版是企业级云原生应用制品管理平台，提供容器镜像、Helm Chart，符合OCI规范制品的生命周期管理；支持大规模、多地域、多场景下应用制品的高效分发；与容器服务ACK无缝集成，帮助企业降低交付复杂度。

容器镜像服务ACR的入门操作，请参见创建企业版实例和使用企业版实例推送和拉取镜像。

产品功能

- 多样OCI制品托管：支持多架构容器镜像（如Linux、Windows、ARM等架构的容器镜像）、支持Helm Chart v2/v3，符合OCI规范的制品管理。
- 多维度安全保障：云原生制品加密存储，支持镜像安全扫描及多维度漏洞报告，保障存储及内容安全；分别提供容器镜像和Helm Chart的网络访问控制管理，细粒度的操作审计，保障制品访问安全。
- 加速应用分发：支持全球多地域间同步，提高容器镜像分发效率；提供P2P分发加速方式，保障业务的极速部署和快速扩展。
- 提效云原生应用交付：提供云原生应用交付链功能，全链路可观测、可追踪、可自主配置；支持基于策略的自动阻断， 实现一次应用变更，全球化多场景自动交付，提升云原生应用交付效率及安全性。

使用场景

- 业务全球地域部署
  在国内地域研发迭代，业务在全球多个地域部署，由于网络不稳定等因素，无法直接从国内地域拉取容器镜像。ACR企业版支持基于地域选购，并通过设置自定义规则实现镜像全球自动同步，满足业务拓展至全球时就近拉取容器镜像的需求，并可实现跨地域容灾。
- 业务大规模部署
  业务大规模部署，在弹性场景或发布迭代场景下，无法快速获取容器镜像部署以应对峰值流量。ACR企业版提供多实例规格，满足不同规模集群的并发拉取需求，支持多种大规模分发加速方式，保障容器业务的快速扩展和极速部署。
- 容器化DevSecOps提效
  业务部署在容器服务ACK、容器服务ASK、企业级分布式应用服务EDAS、弹性容器实例ECI等平台，需要保障交付部署全链路的安全和效率。ACR企业版提供云原生应用交付链功能，全链路可追踪、可观测、可自主配置。实现代码变更后，自动构建成云原生应用制品，自动触发安全扫描和安全加签。基于配置的安全策略，主动阻断存在安全风险的交付链流程。通过安全策略的制品自动触发全球化分发，最终实现多场景下的云原生应用的部署。基于云原生应用交付链，可以将安全内置到交付的全链路中，将DevOps全面升级成DevSecOps，保证云原生应用制品更加安全、高效的交付上线。
- 搬站上云
  在云原生搬站场景，云原生应用制品迁移是首要任务。ACR企业版提供镜像极速导入功能，实现基于OSS Bucket极速导入企业版实例，提升迁移的效率和体验；也支持迁移工具，从Google、Azure、AWS等容器镜像服务平滑迁移至ACR企业版。

22.2. 模板管理

22.2.1. 创建编排模板

编排模板是对一组资源的定义和描述，您可通过编排模板快速创建应用。本文介绍如何创建编排模板。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。
3. 在**模板列表**页面右上角单击**创建**。
4. 在**创建**对话框中，配置编排模板，最后单击**保存**。
   - **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。

     > **说明**
     > - 可包含多个资源对象，以 `---` 进行分割。
     > - 可在YAML中添加 `${params}` 格式的值（例如 `app: ${nginx}` ），该值会在您部署应用时被自动识别为参数。

   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）您也可以把容器服务内置的模板保存为编排模板。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
   iv. 在**无状态**页面单击右上角的**使用YAML创建资源**。
   v. 选择一个内置模板，然后单击**保存模板**。
   vi. 在**保存模板**的对话框中，配置模板信息，包括名称、描述和模板。然后单击**保存**。

   > **说明** 您可在内置模板的基础上进行修改。

后续步骤

您可使用**我的模板**下的编排模板，快速创建应用。

22.2.2. 更新编排模板

您可对已有的编排模板进行编辑，从而更新编排模板。

前提条件

您已经创建一个编排模板，参见创建编排模板。

操作步骤

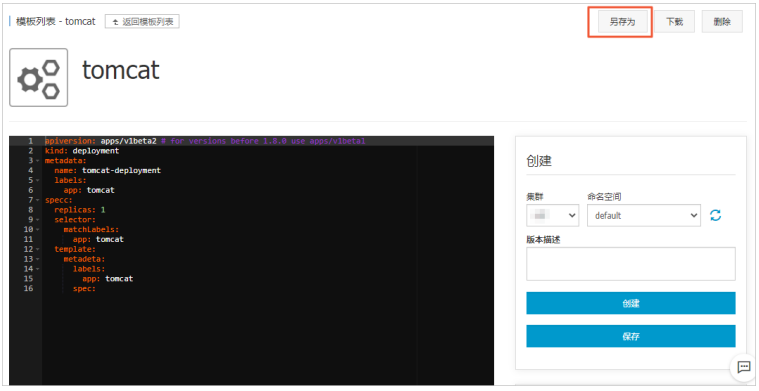
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。
3. 选择所需的模板，单击**详情**。
4. 在该模板的详情页中，编辑**集群**、**命名空间**、**版本描述**和模板，完成配置后，单击**保存**。
5. 返回**模板列表**页面，在**我的模板**下，您可看到该模板已生变化。

22.2.3. 另存编排模板

您可对已有模板进行另存，保存为一个新的编排模板。

前提条件
您已经创建一个编排模板，参见创建编排模板。

操作步骤
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。
3. 选择所需的模板，单击**详情**。
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**。

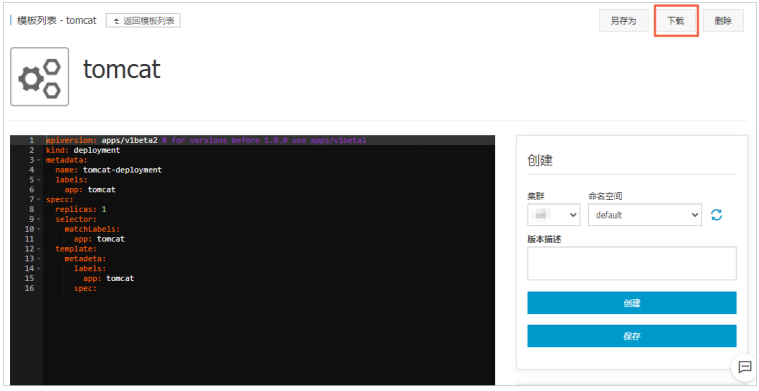


5. 在**模板另存为**对话框中配置模板名称，然后单击**确定**。
返回**模板列表**页面，在**我的模板**下，您可看到另存的模板出现在该列表下

22.2.4. 下载编排模板
您可下载已有的编排模板。

前提条件
您已经创建一个编排模板，参见创建编排模板。

操作步骤
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。
3. 选择所需的模板，单击**详情**。
4. 在该模板的详情页中，您可单击右上角**下载**，下载后缀为yml格式的模板文件。



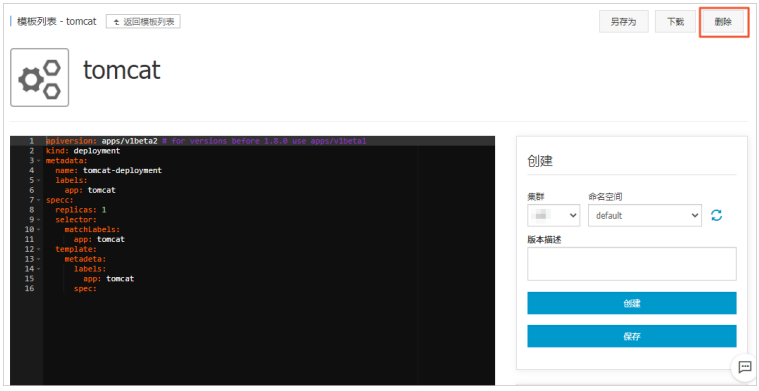
22.2.5. 删除编排模板
您可删除不再需要的编排模板。

前提条件

您已经创建一个编排模板，参见创建编排模板。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。
3. 选择所需的模板，单击**详情**。
4. 在该模板的详情页中，您可单击右上角**删除**。



5. 在**提示**对话框中，单击**确定**。

22.3. 应用目录管理

22.3.1. 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，具体操作，请参见使用Helm简化应用部署。

免责声明

应用目录中一部分软件是由ACK基于开源软件做了适配和二次开发。对于该类软件，ACK提供完整的技术支持服务。但对于在使用过程中因开源软件本身产生的缺陷而导致业务受损的情况，ACK不提供赔偿或者补偿等商务服务。

22.3.2. 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. 单击控制台左侧导航栏中的**市场 > 应用目录**，进入应用目录列表。

   单击**阿里云应用**或**APP Hub**页签，您可以查看对应页签下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，具体请参见使用Helm简化应用部署 。

23.弹性容器实例ECI

23.1. 通过部署ACK虚拟节点组件创建ECI Pod

虚拟节点（Virtual Node）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。您可以灵活动态地按需创建ECI Pod，免去集群容量规划的麻烦。本文主要介绍虚拟节点和ECI，通过ack-virtual-node组件如何部署虚拟节点及如何创建ECI Pod。

前提条件

- 您需要创建一个Kubernetes托管版或者专有版集群（高于1.14版本）。具体操作，请参见创建Kubernetes托管版集群或创建Kubernetes专有版集群。
- 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。
- 您需要确认集群所在区域在ECI支持的地域列表内。登录弹性容器实例控制台查看已经支持的地域和可用区。

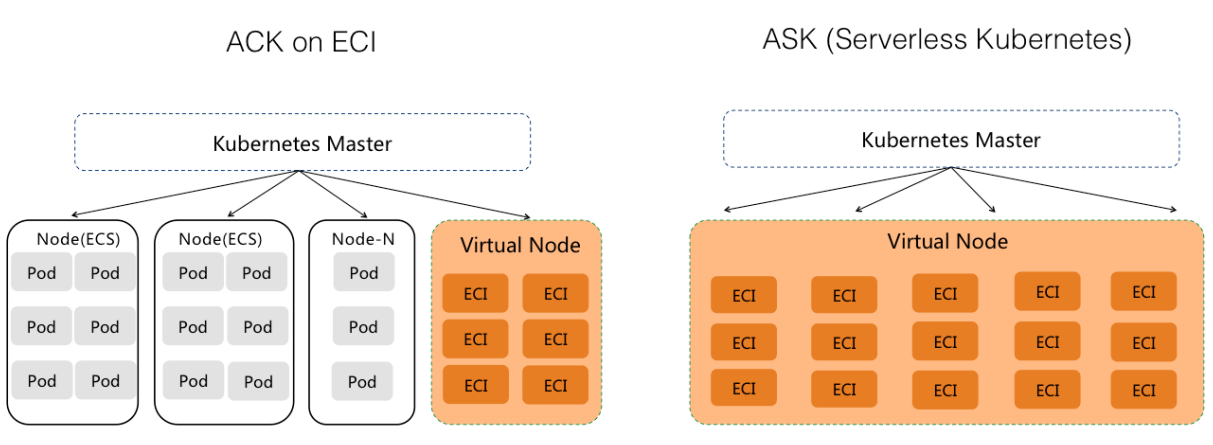
虚拟节点和弹性容器实例ECI

阿里云弹性容器实例ECI（Elastic Container Instance）是面向容器的无服务器弹性计算服务，提供免运维、强隔离、快速启动的容器运行环境。使用ECI无需购买和管理底层ECS服务器，让您更加关注在容器应用而非底层基础设施的维护工作。您可按需创建ECI，仅为容器配置的资源付费（按量按秒计费）。

虚拟节点Virtual Node非常适合运行在如下多个场景，帮助用户极大降低计算成本，提升计算弹性效率：

- 在线业务的波峰波谷弹性伸缩：如在线教育、电商等行业有着明显的波峰波谷计算特征。使用虚拟节点可以显著减少固定资源池的维护，降低计算成本。
- 数据计算：使用虚拟节点承载Spark、Presto等计算场景，有效降低计算成本。
- CI/CD Pipeline：Jenkins、Gitlab-Runner。
- Job任务：定时任务、AI。

阿里云容器服务基于虚拟节点和ECI提供了多种Serverless Container产品形态，包括Serverless Kubernetes（ASK）和ACK on ECI，充分支撑各种弹性和免节点运维场景的用户诉求。



步骤一：在ACK集群中部署ack-virtual-node组件

> **说明**
>
> - 在Serverless Kubernetes集群中您无需手动部署ack-virtual-node组件，Virtual Node Controller默认被托管，您可以直接创建ECI Pod。托管版或专有版则需要先部署ack-virtual-node组件后才可以创建ECI Pod。
> - 在Kubernetes版（ACK）托管版集群中通过**组件管理**页面部署ack-virtual-node组件，Virtual Node Controller默认被托管。
> - 在Kubernetes版（ACK）专有版集群中通过**应用目录**页面部署ack-virtual-node组件，Virtual Node Controller安装在集群节点上。

ACK托管版集群

在ACK托管版集群中部署ack-virtual-node组件的操作步骤如下：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**组件管理**页面，选择**ack-virtual-node**组件，然后单击**安装**。

   这时会以集群默认虚拟交换机和安全组作为ack-virtual-node的初始ECI配置参数。如果需要修改ECI配置参数，请参见相关操作。



ACK专有版集群

在ACK专有版集群中部署ack-virtual-node组件的操作步骤如下：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，然后选中**ack-virtual-node**应用。

   阿里云应用包含较多应用，您可在**应用目录**页面左上角的**名称**文本框中搜索**ack-virtual-node**，支持关键字搜索。
4. 在**应用目录-ack-virtual-node**页面右侧的**创建**页面，选择目标集群。

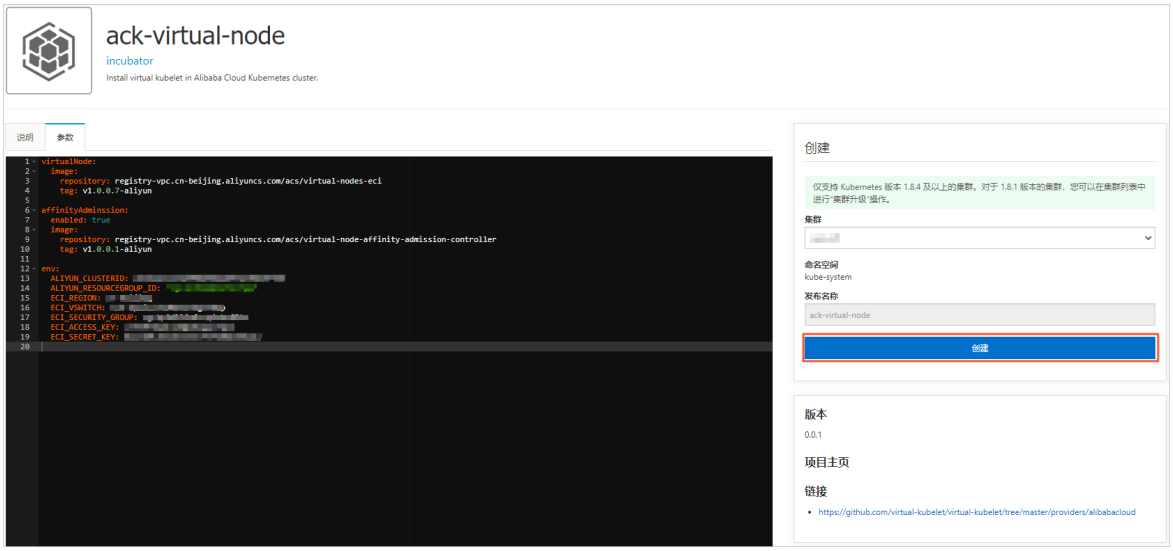
   命名空间已设定为*kube-system*，**发布名称**已设定为*ack-virtual-node*。
5. 在**应用目录-ack-virtual-node**页面，单击**参数**页签，配置虚拟节点参数。

参数	描述	获取路径
ALIYUN_CLUSTERID	集群ID	您可以在集群**基本信息**的**基本信息**区域中，获取**集群ID**的值。

参数	描述	获取路径
ALIYUN_RESOURCEGROUP_ID	资源组ID	如果不配置则为默认资源组。如果需要配置，请登录资源管理控制台，获取**资源组ID**的值。
ECI_REGION	地域名称	您可以在集群**基本信息**的**基本信息**区域中，获取**地域**的值。 **说明** 例如，华东1：cn-hangzhou
ECI_VSWITCH	虚拟交换机	您可以在**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。 **说明** 请确认当前交换机在ECI支持的可用区列表中。 虚拟交换机支持多可用区。因此，这里可以填写多个vSwitch，例如 ECI_VSWITCH: "vsw-xxxxxxx1, vsw-xxxxxxx2, vsw-xxxxxxx3" 。
ECI_SECURITY_GROUP	安全组ID	您可以在**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 ID	请参见获取AccessKey。 请授权RAM的**AliyunECIFullAccess**策略。更多信息，请参见为RAM用户授权。
ECI_SECRET_KEY	用户AccessKey Secret	请参见获取AccessKey。 请授权RAM的**AliyunECIFullAccess**策略。更多信息，请参见为RAM用户授权。

6. 在**应用目录-ack-virtual-node**页面右侧的**创建**页面，选择目标集群。

   命名空间已设定为*kube-system*，**发布名称**已设定为*ack-virtual-node*，然后单击**创建**。



7. 查看**virtual-node-controller**部署状态。

   更多信息，请参见在CloudShell上通过kubectl管理Kubernetes集群。
   执行以下命令查看**virtual-node-controller**部署状态。

```
kubectl -n kube-system get deploy ack-virtual-node-controller
```

预期输出：

```
NAME                        READY  AGE
virtual-node-controller 1/1   1m
```

步骤二：创建ECI Pod

> 说明　在Serverless Kubernetes集群中所有Pod都是ECI Pod，无需给Pod配置特殊标签，因此以下操作不适用于Serverless Kubernetes集群。此操作以Kubernetes版（ACK）集群为例创建ECI Pod。

您可以通过以下两种方法创建ECI Pod。

- **配置Pod标签。**
  给Pod添加标签 `alibabacloud.com/eci=true` ，Pod将以ECI方式运行，其节点是虚拟节点，示例如下：
  i. 执行以下命令给Pod添加标签。
  ```
  kubectl run nginx --image nginx -l alibabacloud.com/eci=true
  ```
  ii. 执行以下命令查看Pod。
  ```
  kubectl get pod -o wide|grep virtual-kubelet
  ```
  预期输出：
  ```
  nginx-7fc9f746b6-r4xgx   0/1   ContainerCreating   0     20s   192.168.XX.XX   virtual-kubelet   <none>     <none>
  ```
- **配置命名空间标签。**
  给Pod所在的命名空间添加标签 `alibabacloud.com/eci=true` ，Pod将以ECI方式运行，其节点是虚拟节点，示例如下：
  i. 执行以下命令创建虚拟节点。
  ```
  kubectl create ns vk
  ```
  ii. 执行以下命令给Pod所在的命名空间添加标签。
  ```
  kubectl label namespace vk alibabacloud.com/eci=true
  ```
  iii. 执行以下命令让命名空间中的Pod调度到虚拟节点上。
  ```
  kubectl -n vk run nginx --image nginx
  ```
  iv. 执行以下命令查看Pod。
  ```
  kubectl -n vk get pod -o wide|grep virtual-kubelet
  ```
  预期输出：
  ```
  nginx-6f489b847d-vgj4d   1/1   Running     0     1m   192.168.XX.XX   virtual-kubelet   <none>     <none>
  ```

相关操作

> 说明　此操作只针对ACK集群，不适用于Serverless Kubernetes集群。

修改ACK虚拟节点配置

关于修改ACK虚拟节点配置的操作，请参见配置ECI Profile。

删除ACK虚拟节点

1. 卸载ack-virtual-node组件。
   - 在ACK托管版集群中，删除所有ECI Pod后，在**组件管理**页面卸载ack-virtual-node组件即可。
   - 在ACK专有版集群中，删除所有ECI Pod后，在**Helm管理**页面删除ack-virtual-node组件即可。
2. 通过命令 `kubectl delete node <node name>` 删除相关虚拟节点。

> **说明** 当集群中存在ECI Pod时，卸载ack-virtual-node组件会导致ECI实例的残留。

相关文档

- 基于ECI运行Job任务

23.2. 配置ECI Profile

在Kubernetes场景中通过Virtual Kubelet使用ECI，如果遇到Pod无法无缝调度或者运行在ECI的问题，您可以尝试通过配置ECI Profile的方式来解决。本文介绍什么是ECI Profile以及如何配置ECI Profile。

ECI Profile介绍

在Kubernetes场景中通过Virtual Kubelet（简称VK）使用ECI时，可能会碰到Pod无法无缝调度到ECI，或者Pod调度到ECI后，为了生效ECI的功能特性（如镜像缓存）需要添加Pod Annotation。这类问题一般由集群管理员处理，但目前只能由研发人员通过修改Pod的YAML文件解决。
针对上述场景，ECI支持通过配置ECI Profile来解决。ECI Profile包括以下两个能力：

- ECI Scheduler
  在混合使用ECI和普通Node的场景下，一般可以通过以下几种方式将Pod调度到ECI:
  - 配置Pod标签
  - 配置Namespace标签
  - 配置ECI弹性调度

  但这些方式均需要对存量资源做一定的修改，无法做到零侵入。
  对于上述情况，ECI Scheduler基于Mutating Webhook机制实现了一种新的调度机制，即在ECI Profile中，您可以声明需要匹配的Namespace或者Pod的Label，对于Label能够匹配上的Pod，将被自动调度到ECI。
- ECI Effect
  对于ECI的一些功能特性，例如指定ECS实例规格，启用镜像缓存，设置NTP服务等，需要在Pod中追加Annotation或者Label来实现。更多信息，请参见ECI Pod Annotation。
  对于上述情况，ECI Effect实现了自动追加Annotation和Label的功能，即在ECI Profile中，您可以声明需要匹配的Namespace或者Pod的Label，以及需要追加的Annotation和Label，对于Label能够匹配上的Pod，将自动追加配置的Annotation和Label。

配置说明

ECI Profile会读取kube-system命名空间下的eci-profile配置文件，然后按照配置文件中的selectors去匹配Pod，对于Label能够匹配上的Pod，将自动调度到ECI，或者追加Annotation和Label，以便生效ECI的功能特性。
您可以通过以下两种方式编辑eci-profile配置文件：

- 通过kubectl edit命令

```
kubectl edit configmap eci-profile -n kube-system
```

- 通过容器服务管理控制台
  i. 在容器服务管理控制台的集群页面，找到要配置的集群，单击集群名称进入集群详情页面。
  ii. 在左侧导航栏选择**配置管理>配置项**。
  iii. 选择命名空间为**kube-system**。

iv. 找到eci-profile进行编辑。

一个典型的eci-profile配置文件示例如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: eci-profile
 namespace: kube-system
data:
 vpcId: "vpc-xxx"
 securityGroupId: "sg-xxx"
 vswitchIds: "vsw-111,vsw-222"
 enableClusterIp: "true"
 enableHybridMode: "false"
 enablePrivateZone: "false"
 selectors: |
  [
    {
      "name":"default-selector-1",
      "objectSelector":{
        "matchLabels":{
          "alibabacloud.com/eci":"true"
        }
      }
    },
    {
      "name":"default-selector-2",
      "objectSelector":{
        "matchLabels":{
          "eci":"true"
        }
      }
    },
    {
      "name":"default-selector-3",
      "namespaceSelector":{
        "matchLabels":{
          "alibabacloud.com/eci":"true"
        }
      }
    },
    {
      "name":"default-selector-4",
      "namespaceSelector":{
        "matchLabels":{
          "eci":"true"
        }
      }
    },
    {
      "name":"default-selector-5",
      "namespaceSelector":{
        "matchLabels":{
          "virtual-node-affinity-injection":"enabled"
        }
      },
      "effect":{ #需要动态追加的Annotation和Label
      "annotations":{
```

```
      annotations":{
        "k8s.aliyun.com/eci-image-cache": "true"
      },
      "labels":{
        "created-by-eci":"true"
      }
    }
  }
]
```

在上述配置文件中，selectors包含了ECI Scheduler和ECI Effect的配置。在每个selector中，您必须声明selector的名称，可以根据需要声明以下信息：

- namespaceSelector：要匹配的Namespace Label。
- objectSelector：要匹配的Pod Label。
- effect：需要动态追加的Annotation和Label。

> **说明**
> eci-profile除了支持配置ECI Scheduler和ECI Effect外，还支持配置安全组、交换机、资源组等信息实现动态更新（无需重启VK）。本文主要说明如何配置ECI Scheduler和ECI Effect，即上述配置文件中的selectors。

配置ECI Scheduler示例

在eci-profile的selectors中，您可以以数组的形式声明需要匹配的Namespace或者Pod的Label。对于Label能够匹配上的Pod，将会被自动调度到ECI。
配置示例如下：

```
selectors: |
[
  {
    "name":"demo", #必填，不能为空
    "namespaceSelector":{ #选填，K8s的Namespace Label（namespaceSelector和objectSelector至少配置一个）
      "matchLabels":{ #需要匹配的Label，如果填写多个，为与关系
        "department":"bigdata"
      }
    },
    "objectSelector":{ #选填，K8s的Pod Label（namespaceSelector和objectSelector至少配置一个）
      "matchLabels":{ #需要匹配的Label，如果填写多个，为与关系
        "type":"offline-task"
      }
    }
  }
]
```

> **注意**
> 在一个selector中，您至少要配置namespaceSelector和objectSelector中的一个，如果两者同时配置了，则Pod需要同时匹配两者。

上述selectors中，声明了一个名为demo的selector，可以实现以下功能：
如果Pod所属命名空间含有department=bigdata标签，并且Pod本身含有type=offline-task标签，则该Pod将自动被调度到ECI。

配置ECI Effect示例

在eci-profile的selectors中，您可以以数组的形式声明需要匹配的Namespace或者Pod的Label，以及需要动态追加的Annotation和Label。对于Label能够匹配上的Pod，将自动追加配置的Annotation和Label。

配置示例如下：

```
selectors: |
 [
  {
    "name":"demo", #必填，不能为空
    "namespaceSelector":{ #选填，K8s的Namespace Label
      "matchLabels":{ #需要匹配的Label，如果填写多个，为与关系
        "department":"bigdata"
      }
    },
    "objectSelector":{ #选填，K8s的Pod Label
      "matchLabels":{ #需要匹配的Label，如果填写多个，为与关系
        "type":"offline-task"
      }
    },
    "effect":{ #需要动态追加的Annotation和Label
      "annotations":{
        "k8s.aliyun.com/eci-image-cache": "true"
      },
      "labels":{
        "created-by-eci":"true"
      }
    }
  }
 ]
```

> 注意
>
> - 在一个selector中，您可以根据需要选择配置namespaceSelector和objectSelector。如果两者均未配置，则effect对所有调度到ECI的Pod均生效。
> - 如果配置了多个selector，将按照顺序匹配selector。匹配成功后，将自动把effect中声明的Annotation和Label追加到Pod中（只追加，不覆盖）。对于重复的Annotation和Label，Pod原有声明的值优先级最高，其次是顺序匹配上的第一个selector中的effect，以此类推。

上述selectors中，声明了一个名为demo的selector，可以实现以下功能：
如果Pod所在的命名空间有department=bigdata标签，并且Pod本身含有type=offline-task标签，则该Pod将自动调度到ECI，并且自动开启镜像缓存功能，增加created-by-eci=true标签。

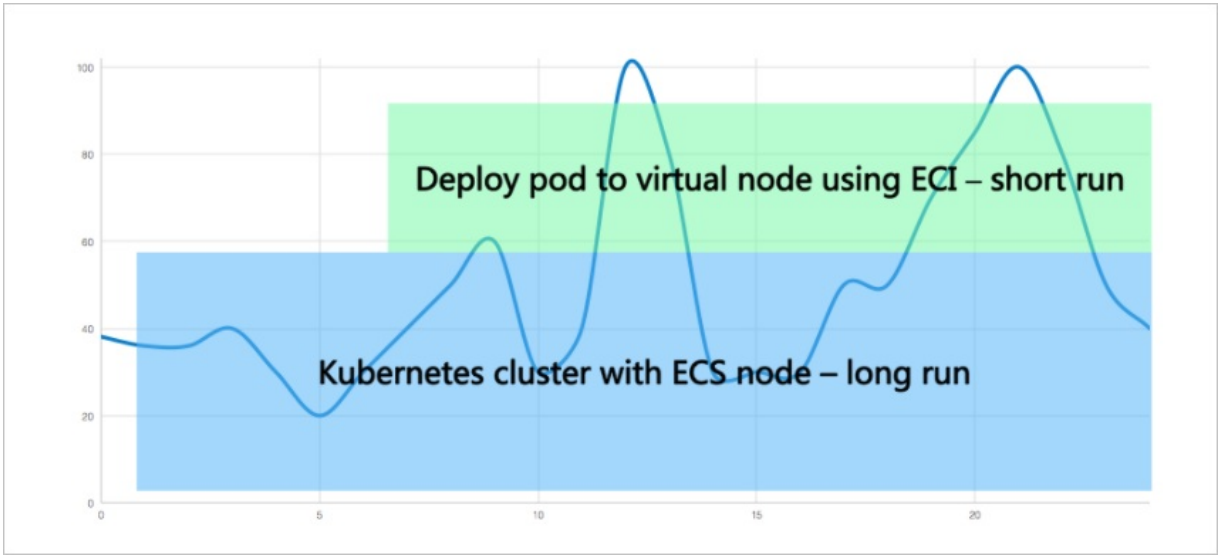
23.3. 基于ECI运行Job任务

本文主要为您介绍通过虚拟节点运行Job任务的方式，您可以用最小的运维成本（无需调整节点数量）来应对集群计算资源高峰压力。

背景信息

Kubernetes集群的节点资源配置不足会导致Pod无法及时运行，购买过多的节点又会导致资源的闲置浪费。而虚拟节点就像是kubernetes集群的“魔法口袋”，让我们摆脱节点计算力不足的烦扰，也避免了节点的闲置浪费，满足无限计算力的想象，Pod按需创建，轻松应对计算的波峰波谷。Kubernetes集群大规模使用虚拟节点有以下特点：

- 业务有明显的波峰谷特征。
- 使用ECI作为弹性资源池，应对突发流量和Job任务，最大程度降低计算成本。



本例中，创建的Kubernetes托管版集群中，Master节点被托管（用户无需付费），2台Worker均为4c8g规格，所以整个Kubernetes托管版集群的可用计算容量是8c16g。需要运行一个Job离线任务做数据处理，Job的cpu/mem需求为16c32g。此时Job无法在您创建的Kubernetes托管版集群中运行，因为其资源需求已经超过Kubernetes托管版集群的8c16g计算容量。因此，您可以选择把这个Job任务调度到虚拟节点上，而不占用集群本身节点的计算资源。

前提条件

- 您已经创建了一个Kubernetes托管版集群。具体操作，请参见创建Kubernetes托管版集群。
- 您已经在该集群上部署了一个虚拟节点。具体操作，请参见通过部署ACK虚拟节点组件创建ECI Pod。
- 您已经给namespace vk打上alibabacloud.com/eci=true标签。具体操作，请参见通过配置namespace标签的方式创建Pod。
- 通过kubectl连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

操作步骤

1. 通过kubectl客户端创建*job.yaml*文件，并复制以下内容到该文件中。

```
apiVersion: batch/v1
kind: Job
metadata:
 name: pi
spec:
 template:
  spec:
   containers:
   - name: pi
    image: perl
    command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
    resources:
     requests:
      cpu: 16
      memory: 32Gi
   restartPolicy: Never
 backoffLimit: 4
```

2. 执行以下命令部署一个job任务。

```
kubectl -n vk apply -f job.yaml
```

3. 查看Pod的运行状态。

执行以下命令查看Pod的状态。

```
kubectl -n vk get pod -a
```

预期输出：

```
NAME    READY   STATUS    RESTARTS  AGE
pi-7cmwv  0/1    Completed  0      2m
```

执行以下命令查看Pod的具体运行状态。

```
kubectl -n vk describe pod
```

预期输出：

```
Name:         pi-7cmwv
Namespace:      vk
Priority:       0
PriorityClassName: <none>
Node:         virtual-kubelet/
...
Events:
 Type   Reason          Age   From         Message
 ----   ------          ----  ----         -------
 Normal Scheduled       3m    default-scheduler  Successfully assigned vk/pi-7cmwv to virtual-kubelet
 Normal SuccessfulMountVolume  2m    kubelet, eci     MountVolume.SetUp succeeded for volume "default-tok
en-b2tff"
 Normal Pulling         2m    kubelet, eci     pulling image "perl"
 Normal Pulled          2m    kubelet, eci     Successfully pulled image "perl"
 Normal Created         2m    kubelet, eci     Created container
 Normal Started         2m    kubelet, eci     Started container
```

> **说明** 虚拟节点中的Pod完全是按需收费，当Pod执行完成后就会停止收费。

因此，可以看到使用虚拟节点运行Job短时间任务可以节省计算成本，减轻集群运维负担，您无需关心集群的计算资源是否充足，也无需关心节点的扩容和缩容。

23.4. 部署Ingress应用

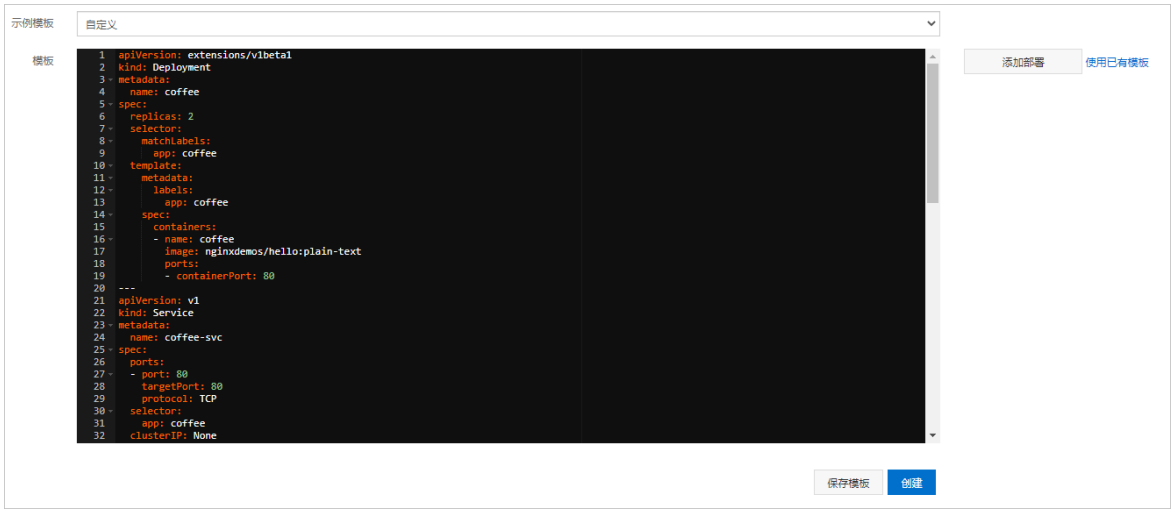
本文主要介绍在虚拟节点上部署Ingress应用，使得集群无需创建新节点即可为该应用扩充无限容量，满足业务高峰低谷的弹性需求。

前提条件

- 您已经部署了一个虚拟节点。具体操作，请参见通过部署ACK虚拟节点组件创建ECI Pod。
- 您已经给命名空间**vk**打上*virtual-node-affinity-injection: enabled*标签。具体操作，请参见通过配置namespace标签的方式创建Pod。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 单击右上角的**使用YAML创建资源**。
6. 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Ingress。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: coffee
spec:
 replicas: 2
 selector:
  matchLabels:
   app: coffee
 template:
  metadata:
   labels:
    app: coffee
  spec:
   containers:
   - name: coffee
    image: nginxdemos/hello:plain-text
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
 name: coffee-svc
spec:
 ports:
 - port: 80
  targetPort: 80
  protocol: TCP
 selector:
  app: coffee
 clusterIP: None
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tea
spec:
 replicas: 3
```

```
  selector:
   matchLabels:
    app: tea
  template:
   metadata:
    labels:
     app: tea
   spec:
    containers:
    - name: tea
     image: nginxdemos/hello:plain-text
     ports:
     - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
 name: tea-svc
 labels:
spec:
 ports:
 - port: 80
  targetPort: 80
  protocol: TCP
 selector:
  app: tea
 clusterIP: None
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: cafe-ingress
spec:
 rules:
 - host: cafe.example.com
  http:
   paths:
   - path: /tea
    backend:
     serviceName: tea-svc
     servicePort: 80
   - path: /coffee
    backend:
     serviceName: coffee-svc
     servicePort: 80
```

预期结果

- 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**，可以看到刚刚创建的coffee和tea。
- 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**，可以看到Pod都运行在Virtual-Kubelet节点上。
- 在集群管理页左侧导航栏中，选择**网络 > 路由**，可以看到刚刚创建的路由。
- 您可以执行如下命令，确保可以访问Ingress应用。

```
kubectl get ing
```

预期输出：

```
NAME      HOSTS        ADDRESS      PORTS  AGE
cafe-ingress  cafe.example.com  114.55.252.185  80    6m18s
```

执行以下命令，验证访问Ingress应用的 "Host:cafe.example.com" <EXTERNAL_IP>/tea 地址。

```
curl -H "Host:cafe.example.com" <EXTERNAL_IP>/tea
```

预期输出：

```
Server address: 192.168.xx.xx:80
Server name: tea-658d56f6cc-cxxxx
Date: 25/Sep/2020:12:36:50 +0000
URI: /tea
Request ID: b01d5bab9ae07abb8bc385377193xxxx
```

执行以下命令，验证访问Ingress应用的 "Host:cafe.example.com" <EXTERNAL_IP>/coffee 地址。

```
curl -H "Host:cafe.example.com" <EXTERNAL_IP>/coffee
```

预期输出：

```
Server address: 192.168.xx.xx:80
Server name: coffee-8c8ff9b4f-hxxxx
Date: 25/Sep/2020:12:36:47 +0000
URI: /coffee
Request ID: 722fe41a65a7fb552613c56e0a9axxxx
```

23.5. 使用ECI弹性调度

ECI弹性调度是阿里云提供的弹性调度策略。您可以在部署服务时通过添加Annotations来申明只使用ECS或ECI弹性资源，或者是在ECS资源不足时自动申请ECI资源。通过ECI弹性调度可以满足您在不同工作负载的场景下对弹性资源的不同需求。

更多信息，请参见使用ECI弹性调度。

23.6. 虚拟节点基于云解析PrivateZone的服务发现

阿里云Kubernetes集群的虚拟节点已经支持服务发现功能，目前支持Intranet service、Headless service、ClusterIP service。

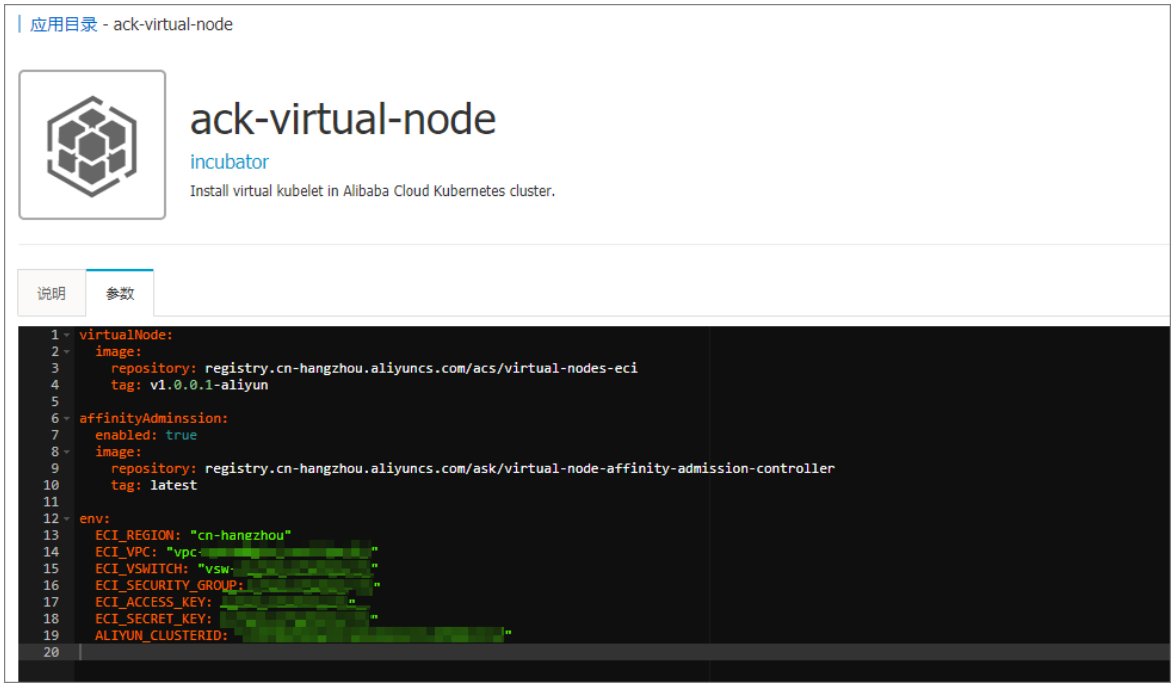
前提条件

- 需要先开通云解析PrivateZone，在云解析DNS控制台中开通。
- 您在Kubernetes集群创建一个虚拟节点，请参见通过部署ACK虚拟节点组件创建ECI Pod。
- 您已成功连接到Kubernetes集群，请参见通过kubectl工具连接集群。

环境准备

通过Chart部署虚拟节点后，需要确认已正确配置相关环境变量。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**阿里云应用**页签中，选择**ack-virtual-node**。
4. 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**。
5. 在**应用目录 - ack-virtual-node**中，单击**参数**，可以通过修改参数配置进行定制化，如下所示。



- `ECI_VPC`：Kubernetes集群使用的VPC ID，需要替换成您当前集群使用的VPC ID。
- `ALIYUN_CLUSTERID`：Kubernetes集群ID，需要替换成您当前使用集群的ID。不允许为空或者设置为default。

操作步骤

1. 部署Deployment和创建Service。

   样例模板如下所示，在YAML文件中复制如下YAML代码，然后执行`kubectl create -f nginx-service-ack.yaml`命令进行创建。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-headless-service
 annotations:
   service.beta.kubernetes.io/alibaba-cloud-private-zone-enable: "true"
spec:
 ports:
 - port: 80
   protocol: TCP
 selector:
   app: nginx
 clusterIP: None
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-clusterip-service
 annotations:
   service.beta.kubernetes.io/alibaba-cloud-private-zone-enable: "true"
spec:
 ports:
 - port: 80
   protocol: TCP
 selector:
```

```
  app: nginx
 type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-intranet-service
 annotations:
  service.beta.kubernetes.io/alicloud-loadbalancer-address-type: intranet
  service.beta.kubernetes.io/alibaba-cloud-private-zone-enable: "true"
spec:
 ports:
 - port: 80
  protocol: TCP
 selector:
  app: nginx
 type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image:  nginx:alpine
    ports:
    - containerPort: 80
```

说明 Kubernetes集群中的Service默认是不会同步到PrivateZone，需要给Service加上以下信息：

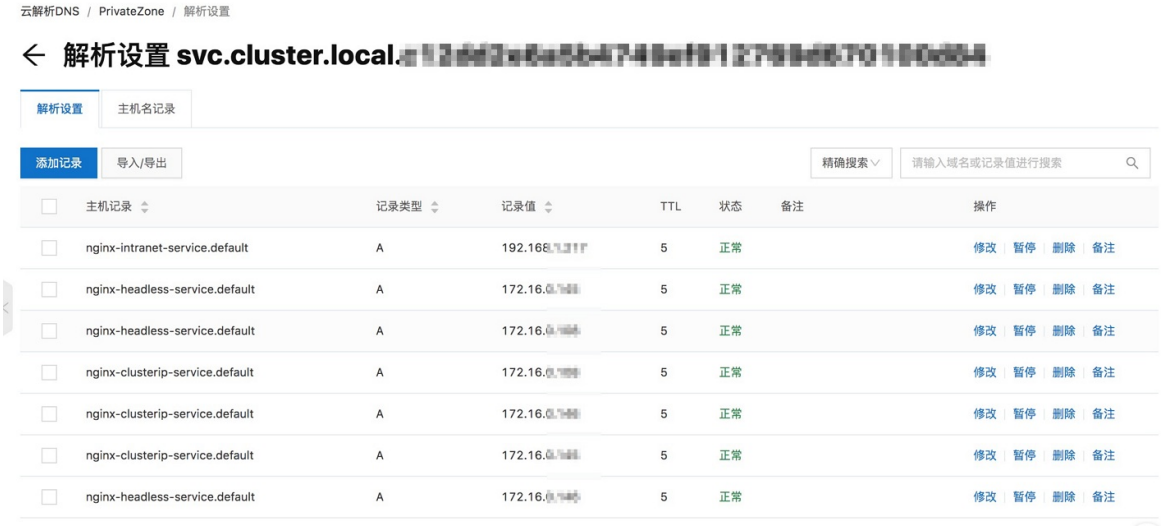
```
annotations:
  service.beta.kubernetes.io/alibaba-cloud-private-zone-enable: "true"
```

此时，虚拟节点Controller才会把Service的解析记录同步到PrivateZone中。

2. 登录云解析DNS控制台。
3. 在左侧导航栏中单击**PrivateZone**，在**权威Zone**页签，可看到在Zone中自动生成IP解析记录。



4. 在目标Zone右侧单击**解析设置**，弹出解析设置详情页面。



> **说明** Zone里面的Record格式为 `$svc.$ns` ，对应相应的IP解析。解析规则如下：
> - LoadBalancer service：PrivateZone中只对应一条解析Record，为SLB IP。
> - ClusterIP service：PrivateZone中对应多条解析Record，分别为后端Pod的IP。
> - Headless service：PrivateZone中对应多条解析Record，分别为后端Pod的IP。

您可在该VPC网络环境中通过私有域名访问Service。

- 长域名访问： `$svc.$ns.svc.cluster.local.$clusterId` ，通过这种方式也可以访问其他集群中同步到PrivateZone的service。
- 短域名访问：您可以通过 `$svc` 访问本Namespace下的Service，通过 `$svc.$ns` 访问其他Namespace中的Service。

23.7. 安装virtual-kubelet-autoscaler

virtual-kubelet-autoscaler是阿里云提供的Kubernetes Autoscaler插件，在Pod进行调度时，如果没有匹配的node节点（例如：已有的ECS work节点资源不足），会由virtual-kubelet-autoscaler进行接管，把调度失败的Pod进行重新调度，调度到ECI（虚拟节点）上。

前提条件

- 创建一个Kubernetes托管版集群。具体操作，请参见创建Kubernetes托管版集群。
- 已在集群上部署了一个虚拟节点。具体操作，请参见通过部署ACK虚拟节点组件创建ECI Pod。

部署virtual-kubelet-autoscaler

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面**阿里云应用**页签，然后选中ack-virtual-kubelet-autoscaler应用。
   阿里云应用包含较多应用，您可在**名称**文本框中搜索ack-virtual-kubelet-autoscaler。
4. 在**应用目录 - ack-virtual-kubelet-autoscaler**页面右侧的**创建**区域，选择目标集群，单击**创建**。



执行结果

1. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
2. 在集群管理页左侧导航栏中，选择**应用 > Helm**，可以找到ack-virtual-kubelet-autoscaler的应用。

23.8. 安装Elastic Workload

弹性负载会监听原始负载，并根据弹性单元设定的调度策略，克隆并生成弹性单元的负载。根据弹性负载中副本的变化，动态的分配原始负载和弹性单元上面的副本数目。本文介绍如何安装以及使用Elastic Workload（弹性负载）。

前提条件

- 您已经成功创建一个Kubernetes集群，请参见创建Kubernetes托管版集群。
- 如果创建的是ACK集群，您还需要部署ack-virtual-node（请使用v2.0.0.102-045a06eb4-aliyun及以上版本），请参见ACK使用ECI。

部署ack-kubernetes-elastic-workload

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，然后选中ack-kubernetes-elastic-workload应用。
   阿里云应用包含较多应用，您可在页面右上角搜索ack-kubernetes-elastic-workload，支持关键字搜索。
4. 在**应用目录 - ack-kubernetes-elastic-workload**页面右侧的**创建**区域，选择目标集群，单击**创建**。



5. 在控制台左侧导航栏中，单击**集群**。
6. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
7. 在集群管理页左侧导航栏中，选择**应用 > Helm**，可以找到ack-kubernetes-elastic-workload的应用。

使用elastic-workload

在Kubernetes中，任何一种负载都要解决两个问题，一个是调度问题，一个是生命周期管理问题。要实现上面描述的场景，我们核心要解决以下两个问题：

- 当副本数目到达某个数值后，如何控制调度策略的变化。
- 在生命周期管理时，如何优先处理某些Pod。

针对上面的问题介绍弹性负载的使用方式与解决方法。

使用模板创建一个简单的Deployment类型应用。

```
apiVersion: apps/v1 # 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:
  #  nodeSelector:
  #    env: test-team
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

使用弹性负载表达Deployment类型应用。

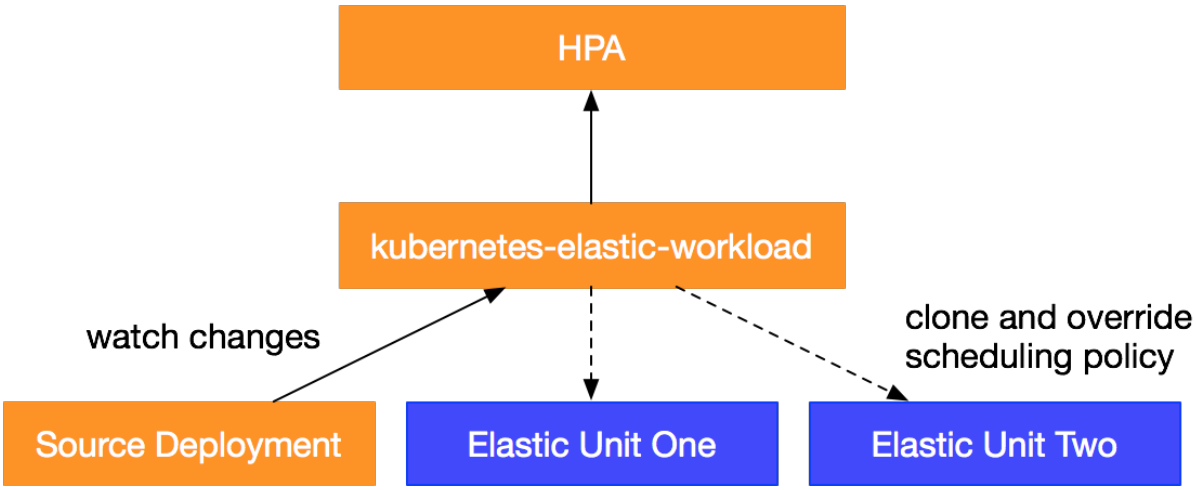
```
# 弹性负载定义。
apiVersion: autoscaling.alibabacloud.com/v1beta1
kind: ElasticWorkload
metadata:
 name: elasticworkload-sample
spec:
 sourceTarget:
  name: nginx-deployment-basic
  kind: Deployment
  apiVersion: apps/v1
  min: 2
  max: 4
 replicas: 6
 elasticUnit:
 - name: virtual-kubelet
  labels:
   alibabacloud.com/eci: "true"
  # min: 0 每个单元也可以指定自己的上下限。
  # max: 10
```

以上就是这个场景的弹性负载定义，弹性负载的使用方式与HPA类似，通过外部挂载的方式使用，对原有的业务无侵入。

一个典型的弹性负载主要分为两个部分：

1. SourceTarget部分主要定义原始负载的类型、副本数目可变化的范围。
2. elasticUnit部分是一个数组，定义弹性单元的调度策略，如果有多个弹性单元，则按照模板的顺序定义。

在上面的例子中，SourceTarget的副本上下限位2~4，表示当ElasticWorkload的replicas为2~4个副本时，会分配到sourceTarget，当超过4个副本时，会分配到弹性单元virtual-kubelet，而在弹性单元virtual-kubelet中可以定义这个单元所独有的调度策略，包含label、annotation、nodeSelector、affinity、toleration等。



弹性负载会监听原始负载，并根据弹性单元设定的调度策略，克隆并生成弹性单元的负载。根据弹性负载中副本的变化，动态的分配原始负载和弹性单元上面的副本数目。

执行弹性负载模板后，您可以通过命令行查看当前的状态，其中status中的每个单元的Desired Replicas表示弹性负载的分配副本数目。

执行以下命令查看弹性负载状态。

```
kubectl describe ew elasticworkload-sample   # same as kubectl get elasticworkload
```

预期输出：

```
Name:       elasticworkload-sample
Namespace:  default
Labels:     <none>
Annotations: <none>
API Version: autoscaling.alibabacloud.com/v1beta1
Kind:       ElasticWorkload
Metadata:
  Creation Timestamp: 2021-04-30T06:58:03Z
  Generation:       2
  Resource Version: 1126368056
  Self Link:       /apis/autoscaling.alibabacloud.com/v1beta1/namespaces/default/elasticworkloads/elasticworkload-sample
  UID:          a465de9e-1253-4cbe-8cd5-98393393e990
Spec:
  Elastic Unit:
    Labels:
      alibabacloud.com/eci: true
    Name:              virtual-kubelet
  Replicas:            6
  Source Target:
    API Version: apps/v1
    Kind:       Deployment
    Max:        4
    Min:        2
    Name:       nginx-deployment-basic
Status:
  Elastic Units Status:
    Desired Replicas: 2
    Name:          nginx-deployment-basic-unit-virtual-kubelet
    Update Timestamp: 2021-04-30T06:58:03Z
  Replicas:         6
  Selector:         app=nginx
  Source Target:
    API Version:     apps/v1
    Desired Replicas: 4
    Kind:          Deployment
    Name:          nginx-deployment-basic
    Update Timestamp: 2021-04-30T06:58:03Z
Events:
  Type    Reason           Age            From         Message
  ----    ------           ----           ----         -------
  Normal  UnitCreation     110s           ElasticWorkload  ElasticWorkloadUnit nginx-deployment-basic-unit-virtual-kubelet created
  Normal  ElasticWorkloadUpdate  110s (x2 over 110s)  ElasticWorkload  ElasticWorkload update
  Normal  UnitUpdate       110s           ElasticWorkload  ElasticWorkloadUnit virtual-kubelet has been updated
```

当下发弹性负载的模板后，可以查看Pod的情况。可以发现弹性负载克隆出了新的Deployment与Pod，并且Deployment的Pod副本数目是根据上述的规则进行动态分配的。

执行以下命令查看Pod详情。

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

预期输出：

```
NAME                                READY  STATUS   RESTARTS  AGE  IP        NODE              NOMINATED NODE  RE
ADINESS GATES
nginx-deployment-basic-769f84b5cf-hdmlw            1/1   Running  0     46m  172.26.240.69  cn-hangzhou.10.1.8
4.111     <none>      <none>
nginx-deployment-basic-769f84b5cf-lmd99            1/1   Running  0     46m  172.26.240.6   cn-hangzhou.10.1.84.
112     <none>      <none>
nginx-deployment-basic-769f84b5cf-nbp5c            1/1   Running  0     30m  172.26.240.75  cn-hangzhou.10.1.8
4.111     <none>      <none>
nginx-deployment-basic-769f84b5cf-scj68            1/1   Running  0     30m  172.26.240.11  cn-hangzhou.10.1.84.
112     <none>      <none>
nginx-deployment-basic-unit-virtual-kubelet-594f86b5c9-8z876  1/1   Running  0     10m  10.1.84.119   virtual-ku
belet-cn-hangzhou-i  <none>      <none>
nginx-deployment-basic-unit-virtual-kubelet-594f86b5c9-drxvq  1/1   Running  0     10m  10.1.84.118   virtual-ku
belet-cn-hangzhou-i  <none>      <none>
```

此外，弹性负载也支持与HPA配合使用，可以将HPA作用在弹性负载上，如上图，弹性负载会根据HPA的状态动态调整每个单元的副本分布，例如如果当前是从6个副本缩容到4个副本，那么会优先将弹性单元的副本进行缩容。

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
 name: elastic-workload-demo
 namespace: default
spec:
 scaleTargetRef:
  apiVersion: autoscaling.alibabacloud.com/v1beta1
  kind: ElasticWorkload
  name: elasticworkload-sample
 minReplicas: 2
 maxReplicas: 10
 metrics:
 - type: Resource
  resource:
   name: cpu
   target:
    type: Utilization
    averageUtilization: 50
```

说明 弹性负载一方面通过克隆和覆写调度策略的方式生成多个Deployment，实现了调度策略的管理，另一方面通过上层的副本计算，调整原始负载和弹性单元的副本分配，实现了针对一部分Pod的优先处理。

24.Windows容器

24.1. 创建Windows节点池

您可以通过节点池管理集群中的一组节点资源，例如在节点池中统一管理节点的标签和污点。本文介绍如何基于容器服务Kubernetes版（ACK）集群中的节点池功能创建Windows节点。

前提条件

- 您已创建一个ACK集群，请参见创建Kubernetes托管版集群。
- 请确保您集群的Kubernetes版本大于1.9。

注意事项

- Windows操作系统的节点池仅支持Flannel网络，不支持Terway网络。
- 默认情况下，每个集群中最多可包含100个节点。如果您需要添加更多节点，请提交工单申请。
- 添加已有云服务器时，请确保您的专有网络中的ECS实例绑定一个弹性公网IP（EIP），或者相应VPC已经配置了NAT网关。您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。
- Windows节点池支持的操作系统为Windows Server 2019和Windows Server Core，Version 1909/2004。

  > **说明** ACK控制台支持选择Windows Server 2019和Windows Server Core，Version 2004。如果您有使用Windows Server Core，Version 1909的需求，请根据ECS公共镜像发布记录来选择最新的镜像ID创建节点池。更多信息，请参见公共镜像发布记录。

- Windows节点池的机器规格请选择4核8 GB以上的规格。
  - Windows容器在内存使用超出Limitation后并不会执行OOM Killed。从2021年5月起，在v1.16及之后版本的ACK集群，新添加的Windows节点启动时会预留一定的资源（1.5核CPU、2.5 GB RAM、3 GB DISK），以保证Windows系统、kubelet、容器运行时的稳定运行。更多关于OOM的信息，请参见内存预留与处理。

    > **说明** ACK通过预留资源机制，可以减少或避免因Windows工作负载的过度分配而导致Windows节点完全不可用的场景产生。但当Windows容器应用发生内存泄露时，依然存在击垮Windows节点的可能性。

  - Windows容器有一定的Footprint。更多信息，请参见Windows容器的内存要求。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面右上角，单击**创建节点池**。

   您还可以在**节点池**页面右上角，单击**创建托管节点池**或者**自动弹性伸缩配置**创建托管节点池或者具有弹性功能的节点池。
6. 在**创建节点池**页面，设置创建节点池的配置项。

   有关配置项的详细说明，请参见创建Kubernetes托管版集群。部分配置项说明如下：
   - 数量：设置节点池初始节点数量。如不需要创建节点，可以填写为0。
   - 操作系统：选择Windows操作系统。
   - 节点标签：为集群节点添加标签。
   - ECS标签：为ECS实例添加标签。

7. 单击**确认配置**。
   在**节点池**页面，如果节点池**状态**显示**初始化中**，则说明节点池正在创建中。创建完成后，**状态**显示为**已激活**。



24.2. 创建Windows应用

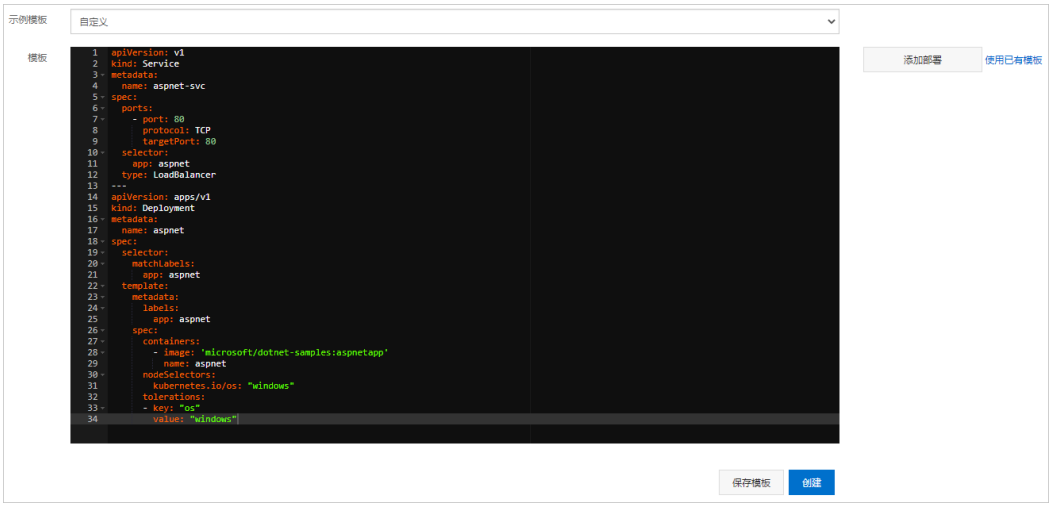
本例演示如何通过一个编排模板创建Web应用，包含一个Deployment和Service，后端Deployment会创建Pod资源对象，Service会绑定到后端Pod上，形成一个完整的Web应用。

前提条件

- 已创建Windows节点。具体操作，请参见创建Windows节点池。
- 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。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页面单击右上角的**使用YAML创建资源**。
6. 对模板进行相关配置，完成配置后单击**创建**。
   - **命名空间**：在**创建**页面的顶部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。
   - **添加工作负载**：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。
   - **保存模板**：您可以保存设置好的编排模板。



下面是一个Web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Web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:
 labels:
  app: web-windows
 name: web-windows
spec:
 type: LoadBalancer
 ports:
  - port: 80
    protocol: TCP
    targetPort: 80
 selector:
  app: web-windows
---
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  app: web-windows
 name: web-windows
spec:
 selector:
  matchLabels:
   app: web-windows
 template:
  metadata:
   labels:
    app: web-windows
  spec:
   restartPolicy: Always
   terminationGracePeriodSeconds: 30
   tolerations:
   - key: os
    value: windows
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/os
          operator: In
          values:
          - windows
   containers:
  # 根据不同集群的地域，您需要修改以下镜像地址中的地域<cn-hangzhou>信息。
    - image: registry-vpc.cn-hangzhou.aliyuncs.com/acs/sample-web-windows:v1.0.1
      name: windows
      ports:
      - containerPort: 80
        protocol: TCP
```

> **说明**
> - 镜像 registry-vpc.{region}.aliyuncs.com/acs/sample-web-windows 是阿里云容器服务提供的Windows Web容器示例镜像（基于mcr.microsoft.com/windows/nanoserver）。
> - 也可使用微软提供的ASP.NET示例镜像： mcr.microsoft.com/dotnet/samples:aspnetapp 。更多信息，请参见.NET Samples。

7. 单击**创建**后，会提示部署状态信息。成功后，返回**无状态**页面可以查看新创建的Web应用。
8. 访问Web应用。
   i. 在控制台左侧导航栏中，单击**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iii. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
   iv. 在**服务**页面您可以看到**web-windows**服务已成功部署，并暴露了外部入口。
   v. 单击**web-windows**服务的外部端点，可以访问到Web应用。
   vi. 修改访问URL为 name.html ，可以看到部署应用的容器的 hostname 。

24.3. 使用Logtail收集Windows容器日志

您可以通过使用Logtail采集Windows容器上的应用信息。本文介绍如何使用Logtail收集Windows容器日志。

前提条件

- 已创建Kubernetes集群，并且在创建集群时选中**使用日志服务**。更多信息，请参见创建Kubernetes托管版集群。
- 已创建Windows节点池。更多信息，请参见创建Windows节点池。
- 已通过kubectl连接ACK集群。具体操作，请参见步骤二：选择集群凭证类型。

使用限制

目前Logtail仅限于支持采集Stdout的输出至日志服务，后续会支持对日志文件的采集。

为Windows节点安装Logtail v1.0.x

执行以下YAML模板将Logtail以DaemonSet方式部署至Windows节点上。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 labels:
  k8s-app: logtail-ds-windows
 name: logtail-ds-windows
 namespace: kube-system
spec:
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   app: logtail-ds-windows
 template:
  metadata:
   annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
   labels:
    app: logtail-ds-windows
  spec:
   affinity:
```

```
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: type
          operator: NotIn
          values:
          - virtual-kubelet
        - key: beta.kubernetes.io/os
          operator: In
          values:
          - windows
        - key: windows.alibabacloud.com/deployment-topology
          operator: DoesNotExist
      - matchExpressions:
        - key: type
          operator: NotIn
          values:
          - virtual-kubelet
        - key: kubernetes.io/os
          operator: In
          values:
          - windows
        - key: windows.alibabacloud.com/deployment-topology
          operator: DoesNotExist
  containers:
  - name: logtail
    command:
    - powershell.exe
    - -NoLogo
    - -NonInteractive
    - -File
    - entrypoint.ps1
    env:
    # --- configure the logtail file configuration generation --- #
    ## "ALICLOUD_LOGTAIL_CONFIG_PATH" specifies the configuration path of logtail,
    ## if the path is blank, the configuration will generate
    ## via "ALICLOUD_LOGTAIL_CONFIG_ITEM__" prefix environment variables.
    ## "ALICLOUD_LOGTAIL_CONFIG_ITEM__" needs to be combined with a type indicator to
    ## display the type of the configuration item, the following indicators are optional.
    ## -     INT__ : 64-bit signed integer.
    ## -    UINT__ : 64-bit unsigned integer.
    ## -    DOUB__ : 64-bit floating-point number.
    ## -    BOOL__ : boolean.
    ## -     STR__ : string, default type.
    ## - {<TYPE>}S__ : array in <TYPE> with vertical bar separated format.
    ## P.S: don't treate "ALICLOUD_LOGTAIL_CONFIG_ITEM__" as a silver bullet,
    ## when the configuration is too complicated,
    ## please mount a detailed configuration file on "ALICLOUD_LOGTAIL_CONFIG_PATH".
    - name: ALICLOUD_LOGTAIL_CONFIG_PATH
      valueFrom:
        configMapKeyRef:
          key: log-config-path
          name: alibaba-log-configuration
    - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__DOUB__CPU_USAGE_LIMIT
      valueFrom:
        configMapKeyRef:
          key: cpu_core_limit
```

```
      key: cpu-core-limit
      name: alibaba-log-configuration
  - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__UINT__MEM_USAGE_LIMIT
    valueFrom:
      configMapKeyRef:
        key: mem-limit
        name: alibaba-log-configuration
  - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__UINT__MAX_BYTES_PER_SEC
    valueFrom:
      configMapKeyRef:
        key: max-bytes-per-sec
        name: alibaba-log-configuration
  - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__UINT__SEND_REQUESTS_CONCURRENCY
    valueFrom:
      configMapKeyRef:
        key: send-requests-concurrency
        name: alibaba-log-configuration
  # --- configure the logtail configuration --- #
  ## "ALICLOUD_LOG_REGION" specifies the region of the Alibaba Cloud,
  ## it will discover the region automatically if blank.
  ## "ALICLOUD_LOG_USER_ID" is the same as "ALIYUN_LOGTAIL_USER_ID",
  ## which is to specify the uid of Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_PROJECT" is the same as "ALICLOUD_LOG_DEFAULT_PROJECT",
  ## which is to specify the project of Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_MACHINE_GROUP" is the same as "ALIYUN_LOGTAIL_USER_DEFINED_ID"
  ## and "ALICLOUD_LOG_DEFAULT_MACHINE_GROUP",
  ## which is to specify the machine group of Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_ENDPOINT" specifies the endpoint of the Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_ECS_FLAG" specifies whether to log the ECS flags.
  ## "ALICLOUD_LOG_DOCKER_ENV_CONFIG"
  ## "ALICLOUD_LOG_ENV_TAGS" is the same as "ALIYUN_LOG_ENV_TAGS",
  ## which is to specify the environment variables to be recorded,
  ## it is in form of vertical bar separated list.
  - name: ALICLOUD_LOG_REGION
    value: ""
  - name: ALICLOUD_LOG_USER_ID
    valueFrom:
      configMapKeyRef:
        key: log-ali-uid
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_PROJECT
    valueFrom:
      configMapKeyRef:
        key: log-project
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_MACHINE_GROUP
    valueFrom:
      configMapKeyRef:
        key: log-machine-group
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_ENDPOINT
    valueFrom:
      configMapKeyRef:
        key: log-endpoint
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_ECS_FLAG
    value: "true"
  - name: ALICLOUD_LOG_DOCKER_ENV_CONFIG
```

```
      value: "true"
    - name: ALICLOUD_LOG_ENV_TAGS
      value: _node_name_|_node_ip_|working_ip
    - name: _node_name_
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: spec.nodeName
    - name: working_ip
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: status.hostIP
    - name: _node_ip_
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: status.hostIP
    # 根据不同集群的地域，您需修改以下镜像地址中的地域<cn-hangzhou>信息。
    image: registry-vpc.cn-hangzhou.aliyuncs.com/acs/logtail-windows:v1.0.20
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        cpu: 1
        memory: 1Gi
      requests:
        cpu: 100m
        memory: 256Mi
    securityContext:
      privileged: false
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - name: docker-pipe
      mountPath: \\.\pipe\docker_engine
    - name: docker-data
      mountPath: c:/ProgramData/docker
      readOnly: true
    - mountPath: c:/logtail_host
      name: root
      readOnly: true
  terminationGracePeriodSeconds: 30
  priorityClassName: system-node-critical
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
  tolerations:
  - operator: Exists
  volumes:
  - name: docker-pipe
    hostPath:
      path: \\.\pipe\docker_engine
  - name: docker-data
    hostPath:
      path: c:/ProgramData/docker
      type: Directory
  - name: root
    hostPath:
      path: c:/
```

```
      type: Directory
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 10%
    type: RollingUpdate
```

为Windows节点安装Logtail v1.1.x

> 注意 Logtail v1.1.x依赖Windows工作节点上的特权托管进程，仅适用于集群内节点池默认含有 `windows.alibabacloud.com/deployment-topology=2.0` 标签的Windows工作节点。

使用以下YAML模板将Logtail以DaemonSet方式部署至Windows节点上。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    k8s-app: logtail-ds-windows
  name: logtail-ds-windows
  namespace: kube-system
spec:
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: logtail-ds-windows
  template:
    metadata:
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ""
      labels:
        app: logtail-ds-windows
    spec:
      #since 1.18, we can specify `hostNetwork: true` for Windows workloads, so we can deploy an application witho
ut NetworkReady.
      hostNetwork: true
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: type
                operator: NotIn
                values:
                - virtual-kubelet
              - key: beta.kubernetes.io/os
                operator: In
                values:
                - windows
              - key: windows.alibabacloud.com/deployment-topology
                operator: In
                values:
                - "2.0"
            - matchExpressions:
              - key: type
                operator: NotIn
                values:
```

```
        - virtual-kubelet
      - key: kubernetes.io/os
        operator: In
        values:
        - windows
      - key: windows.alibabacloud.com/deployment-topology
        operator: In
        values:
        - "2.0"
  containers:
  - name: logtail
    # use the below volume configuration since v1.1+.
    command:
    - pwsh.exe
    - -NoLogo
    - -NonInteractive
    - -File
    - entrypoint.ps1
    env:
    # --- configure the logtail file configuration generation --- #
    ## "ALICLOUD_LOGTAIL_CONFIG_PATH" specifies the configuration path of logtail,
    ## if the path is blank, the configuration will generate
    ## via "ALICLOUD_LOGTAIL_CONFIG_ITEM__" prefix environment variables.
    ## "ALICLOUD_LOGTAIL_CONFIG_ITEM__" needs to be combined with a type indicator to
    ## display the type of the configuration item, the following indicators are optional.
    ## -     INT__ : 64-bit signed integer.
    ## -    UINT__ : 64-bit unsigned integer.
    ## -    DOUB__ : 64-bit floating-point number.
    ## -    BOOL__ : boolean.
    ## -     STR__ : string, default type.
    ## - {<TYPE>}S__ : array in <TYPE> with vertical bar separated format.
    ## P.S: don't treate "ALICLOUD_LOGTAIL_CONFIG_ITEM__" as a silver bullet,
    ## when the configuration is too complicated,
    ## please mount a detailed configuration file on "ALICLOUD_LOGTAIL_CONFIG_PATH".
    - name: ALICLOUD_LOGTAIL_CONFIG_PATH
      valueFrom:
        configMapKeyRef:
          key: log-config-path
          name: alibaba-log-configuration
    - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__DOUB__CPU_USAGE_LIMIT
      valueFrom:
        configMapKeyRef:
          key: cpu-core-limit
          name: alibaba-log-configuration
    - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__UINT__MEM_USAGE_LIMIT
      valueFrom:
        configMapKeyRef:
          key: mem-limit
          name: alibaba-log-configuration
    - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__UINT__MAX_BYTES_PER_SEC
      valueFrom:
        configMapKeyRef:
          key: max-bytes-per-sec
          name: alibaba-log-configuration
    - name: ALICLOUD_LOGTAIL_CONFIG_ITEM__UINT__SEND_REQUESTS_CONCURRENCY
      valueFrom:
        configMapKeyRef:
          key: send-requests-concurrency
```

```
      key: send-requests-concurrency
      name: alibaba-log-configuration
  # --- configure the logtail configuration --- #
  ## "ALICLOUD_LOG_REGION" specifies the region of the Alibaba Cloud,
  ## it will discover the region automatically if blank.
  ## "ALICLOUD_LOG_USER_ID" is the same as "ALIYUN_LOGTAIL_USER_ID",
  ## which is to specify the uid of Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_PROJECT" is the same as "ALICLOUD_LOG_DEFAULT_PROJECT",
  ## which is to specify the project of Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_MACHINE_GROUP" is the same as "ALIYUN_LOGTAIL_USER_DEFINED_ID"
  ## and "ALICLOUD_LOG_DEFAULT_MACHINE_GROUP",
  ## which is to specify the machine group of Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_ENDPOINT" specifies the endpoint of the Alibaba Cloud SLS service.
  ## "ALICLOUD_LOG_ECS_FLAG" specifies whether to log the ECS flags.
  ## "ALICLOUD_LOG_DOCKER_ENV_CONFIG"
  ## "ALICLOUD_LOG_ENV_TAGS" is the same as "ALIYUN_LOG_ENV_TAGS",
  ## which is to specify the environment variables to be recorded,
  ## it is in form of vertical bar separated list.
  - name: ALICLOUD_LOG_REGION
    value: ""
  - name: ALICLOUD_LOG_USER_ID
    valueFrom:
      configMapKeyRef:
        key: log-ali-uid
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_PROJECT
    valueFrom:
      configMapKeyRef:
        key: log-project
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_MACHINE_GROUP
    valueFrom:
      configMapKeyRef:
        key: log-machine-group
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_ENDPOINT
    valueFrom:
      configMapKeyRef:
        key: log-endpoint
        name: alibaba-log-configuration
  - name: ALICLOUD_LOG_ECS_FLAG
    value: "true"
  - name: ALICLOUD_LOG_DOCKER_ENV_CONFIG
    value: "true"
  - name: ALICLOUD_LOG_ENV_TAGS
    value: _node_name_|_node_ip_|working_ip
  - name: _node_name_
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: spec.nodeName
  - name: _node_ip_
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: status.hostIP
  - name: working_ip
    valueFrom:
```

```
        fieldRef:
          apiVersion: v1
          fieldPath: status.hostIP
      # 根据不同集群的地域，您需修改以下镜像地址中的地域<cn-hangzhou>信息。
      image: registry-vpc.cn-hangzhou.aliyuncs.com/acs/logtail-windows:v1.1.18-aliyun.1
      imagePullPolicy: IfNotPresent
      resources:
        limits:
          cpu: 1
          memory: 1Gi
        requests:
          cpu: 100m
          memory: 256Mi
      securityContext:
        privileged: false
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - name: wins-pipe
        mountPath: \\.\pipe\rancher_engine
      - name: host-binary
        mountPath: c:/host/opt/ilogtail/bin
    terminationGracePeriodSeconds: 30
    priorityClassName: system-node-critical
    restartPolicy: Always
    tolerations:
    - operator: Exists
    volumes:
    - name: wins-pipe
      hostPath:
        path: \\.\pipe\rancher_wins
    - name: host-binary
      hostPath:
        path: c:/opt/ilogtail/bin
        type: DirectoryOrCreate
  updateStrategy:
    type: RollingUpdate
```

部署添加Logtail的应用

在Windows节点上添加Logtail后，使用以下应用模板检验添加的Logtail。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  app: logtail-test
 name: logtail-test
spec:
 replicas: 1
 template:
  metadata:
   labels:
    app: logtail-test
   name: logtail-test
  spec:
   containers:
   - name: nanoserver
    image: mcr.microsoft.com/windows/servercore:1809
    command: ["powershell.exe"]
    args: ["ping -t 127.0.0.1 -w 10000"]
    env:
   ######### 配置环境变量 ############
    - name: aliyun_logs_logtail-stdout
     value: stdout
    - name: aliyun_logs_logttail-tags
     value: tag1=v1
   ##################################
   nodeSelector:
    beta.kubernetes.io/os: windows
   tolerations:
   - effect: NoSchedule
    key: os
    operator: Equal
    value: windows
```

成功部署示例应用后，您可查看日志信息。具体操作，请参见查询和分析日志。

24.4. 在Windows容器中使用云盘及基于SMB的文件存储

您可以在ACK Windows容器中使用云盘动态存储卷及SMB静态存储资源。本文介绍如何在Windows容器中为Windows容器配置阿里云云盘及基于SMB的文件存储。

前提条件

- 已开通Flexvolume存储插件的使用权限。
  若未开通Flexvolume存储插件的使用权限，请提交工单申请配置白名单。
- 已创建Kubernetes集群，且创建集群时使用的存储插件为Flexvolume。具体操作，请参见创建Kubernetes托管版集群。
- 已创建Windows节点池。具体操作，请参见创建Windows节点池。
- 已通过kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

为Windows容器配置阿里云云盘

阿里云云盘是为阿里云服务器（ECS）提供的一种块存储服务，它具有低时延，高性能，耐用性好及高可靠等特点。

您可以通过以下步骤为Windows容器配置阿里云云盘。

步骤一：创建存储类StorageClass

云盘控制器（Cloud Disk Controller）根据存储类（StorageClass）的参数fstype的值判断创建的云盘适用于Windows文件系统或Linux文件系统。

- 当参数fstype的值为ext3、ext4或xfs时，表示该StorageClass适用于Linux文件系统。
- 当参数fstype的值为ntfs时，表示该StorageClass适用于Windows文件系统。

默认情况下，若存储插件选择为Flexvolume，则在ACK集群创建时会定义Linux文件系统适用的存储类（StorageClass），而Windows文件系统适用的存储类需要自定义。

1. 使用以下YAML示例自定义Windows文件系统适用的存储类（StorageClass），并创建*storageclass.yaml*文件。

   以下为一些StorageClass的配置示例说明：

YAML示例	说明
kind: StorageClass apiVersion: storage.k8s.io/v1 metadata: name: alicloud-disk-ssd-windows provisioner: alicloud/disk parameters: type: cloud_ssd fstype: ntfs + allowVolumeExpansion: true # useless	目前Windows不支持盘区扩展。例如，配置的 allowVolumeExpansion: true 并不能起到预期扩展云盘的作用。
kind: StorageClass apiVersion: storage.k8s.io/v1 metadata: name: alicloud-disk-ssd-windows provisioner: alicloud/disk parameters: type: cloud_ssd fstype: ntfs + reclaimPolicy: Retain # only unmount	根据需要使用保留回收策略 reclaimPolicy: Retain 可以减少由于错误操作导致数据删除的问题。

YAML示例	说明

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: alicloud-disk-ssd-windows
provisioner: alicloud/disk
parameters:
  type: cloud_ssd
  fstype: ntfs
+ regionId: cn-hangzhou
+ zoneId: cn-hangzhou-b
	若缺少地域及可用区的定义，云盘可能会挂载失败。使用参数 regionId 及 zoneId 指定地域及可用区可以避免挂载失败。

YAML示例 ❯

2. 执行以下命令创建Windows环境下的StorageClass。

```
kubectl create -f storageclass.yaml
```

步骤二：为Windows节点安装Flexvolume插件

1. 使用以下YAML示例创建*flexvolume-windows.yaml*文件。

YAML示例 ❯

2. 执行以下命令为所有的Windows工作节点安装Flexvolume插件。

```
kubectl create -f flexvolume-windows.yaml
```

3. DaemonSet部署成功后，执行以下命令可以在每个WIndows节点上查看FlexVolume插件alicloud-disk-controller。

```
ls C:\usr\libexec\kubernetes\kubelet-plugins\volume\exec\alicloud~disk.exe\
```

预期输出：

```
Directory: C:\usr\libexec\kubernetes\kubelet-plugins\volume\exec\alicloud~disk.exe
Mode          LastWriteTime      Length Name
----          -------------      ------ ----
-a----     8/12/2021  2:45 PM     5636096 disk.exe
```

步骤三：创建云盘并挂载到Windows容器中

以有状态工作负载（StatefulSet）为例，您可以在volumeClaimTemplates中定义需要的存储类（StroageClass），Windows工作节点上的FlexVolume插件会自动完成所有操作：验证您的账户信息，创建阿里云云盘，并挂载到阿里云服务器上，最后挂载到Windows容器中。

YAML示例 ❯

StatefulSet部署完成后，执行以下命令可查看到持久化存储卷（PersitentVolume）资源对象。

```
kubectl get pv
```

预期输出：

```
NAME              STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS          AGE
ssd-web-windows-disk-0  Bound   d-wz9dihqdkusysc56****  50Gi    RWO       alicloud-disk-ssd-windows  5s
```

步骤四：验证云盘的持久化存储

在持久化存储卷（PersitentVolume）的支持下，部署的工作负载在被删除后，其访问日志仍然可以被保留。

1. 执行以下命令访问部署的Windows应用**web-windows-disk**。

```
for i in {1..2}; do; curl $(kubectl get svc web-windows-disk --template '{{ (index .status.loadBalancer.ingress 0).ip }}'); done
```

2. 执行以下命令查看**web-windows-disk-0**应用的日志。

```
kubectl logs web-windows-disk-0
```

预期输出：

```
openresty started...
x.x.x.x - - [14/Aug/2021:18:28:28 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:28:32 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
```

3. 执行以下命令删除**web-windows-disk-0**应用。

```
kubectl delete pod web-windows-disk-0
```

4. 执行以下命令访问重新部署的Windows应用**web-windows-disk**。

```
for i in {1..3}; do; curl $(kubectl get svc web-windows-disk --template '{{ (index .status.loadBalancer.ingress 0).ip }}'); done
```

5. 执行以下命令查看**web-windows-disk-0**应用的日志。

```
kubectl logs web-windows-disk-0
```

预期输出：

```
openresty started...
x.x.x.x - - [14/Aug/2021:18:28:28 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:28:32 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
openresty started...
x.x.x.x - - [14/Aug/2021:18:56:40 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:56:40 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:56:41 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
```

从前后两次**web-windows-disk-0**应用日志的对比可得，部署的工作负载**web-windows-disk**在被删除后，其访问日志仍然被保留。

为Windows容器配置基于SMB的文件存储

阿里云为ECS提供阿里云文件存储NAS的分布式文件系统。阿里云文件存储NAS基于POSIX文件接口，天然适配原生操作系统，提供共享访问，同时保证数据一致性和锁互斥。

SMB（Server Message Block）是阿里云文件存储NAS实现的一种协议，在Windows平台上工作良好。和阿里云云盘FlexVolume插件不同的是，使用SMB Flexvolume插件并不依赖Linux工作节点或Kubernetes控制器。

没有Kubernetes控制器意味着SMB FlexVolume插件仅能处理静态的持久化存储卷（PersitentVolume）资源对象，而无法利用存储类（StorageClass）动态分配卷。

步骤一：为Windows节点安装Flexvolume插件

为所有的Windows工作节点安装Flexvolume插件。具体操作，请参见为Windows容器配置阿里云云盘的步骤二：为Windows节点安装Flexvolume插件。

步骤二：使用SMB创建持久卷并挂载到Windows容器中

1. 创建持久化存储卷。

   使用以下YAML示例创建PV及PVC，向阿里云文件系统NAS申请10 GiB的文件存储。
   配置项的部分参数说明如下所示：

参数	说明
driver	挂载存储卷的驱动类型，必须为 `alicloud/smb.exe` 。
server	NAS文件系统中挂载点的挂载地址，必须与Windows节点的VPC在同一地域。
path	NAS在容器中挂载的位置，必须以*/myshare/*为父目录并创建子目录。例如，*/myshare/web-windows-logs*。
user	登录ECS的用户名，建议使用**workgroup/administrator**。
password	登录ECS的用户密码。

YAML示例 ❯

2. 使用以下YAML示例创建Service及Deployment，并挂载已创建的持久化存储卷。

   YAML示例 ❯

3. Deployment部署完成后，执行以下命令查看应用及持久化存储卷（PersitentVolume）资源对象。
   - 执行以下命令查看部署的工作负载Deployment。

     ```
     kubectl get pod
     ```

     预期输出：

     ```
     NAME                    READY  STATUS       RESTARTS  AGE
     web-windows-smb-584676df65-t4mmh  1/1   Running      0     26d
     ```

   - 执行以下命令查看持久化存储卷（PersitentVolume）资源对象。

     ```
     kubectl get pv
     ```

     预期输出：

     ```
     NAME     CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM            STORAGECLASS    REASO
     N  AGE
     pv-smb-10g  10Gi    ReadWriteMany  Retain      Bound    default/web-windows-smb                10d
     ```

步骤四：验证NAS的持久化存储

在持久化存储卷（PersitentVolume）的支持下，部署的工作负载在被删除后，其访问日志仍然可以被保留。

1. 执行以下命令访问部署的Windows应用**web-windows-smb**。

   ```
   for i in {1..2}; do; curl $(kubectl get svc web-windows-smb --template '{{ (index .status.loadBalancer.ingress 0).i
   p }}'); done
   ```

2. 执行以下命令查看**web-windows-smb-584676df65-t4mmh**应用的日志。

   ```
   kubectl logs web-windows-smb-584676df65-t4mmh
   ```

   预期输出：

   ```
   openresty started...
   x.x.x.x - - [14/Aug/2021:18:28:28 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
   x.x.x.x - - [14/Aug/2021:18:28:32 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
   ```

3. 执行以下命令删除**web-windows-smb-584676df65-t4mmh**应用。

   ```
   kubectl delete pod web-windows-smb-584676df65-t4mmh
   ```

4. 执行以下命令访问重新部署的Windows应用**web-windows-smb**。

   ```
   for i in {1..3}; do; curl $(kubectl get svc web-windows-smb --template '{{ (index .status.loadBalancer.ingress 0).i
   p }}'); done
   ```

5. 执行以下命令查看应用。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME                    READY  STATUS       RESTARTS  AGE
   web-windows-smb-584676df65-87bpz  1/1   Running      0     26d
   ```

6. 执行以下命令查看**web-windows-smb-584676df65-87bpz**应用的日志。

```
kubectl logs web-windows-smb-584676df65-87bpz
```

预期输出：

```
openresty started...
x.x.x.x - - [14/Aug/2021:18:28:28 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:28:32 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
openresty started...
x.x.x.x - - [14/Aug/2021:18:56:40 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:56:40 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
x.x.x.x - - [14/Aug/2021:18:56:41 +0800] "GET / HTTP/1.1" 200 674 "-" "curl/7.64.1"
```

从前后两次**web-windows-smb-******应用日志的对比可得，部署的工作负载**web-windows-smb-584676df65-t4mmh**在被删除后，其访问日志仍然被保留。

24.5. 在Windows容器中使用基于DirectX的GPU加速

对于Windows节点的工作负载，GPU相比于CPU可提供更大规模的并行计算能力，且能够将操作速度提高几个数量级，从而降低成本并提高吞吐量。Windows容器支持对基于DirectX构建的所有框架进行GPU加速。本文介绍在Windows节点如何安装DirectX设备插件以及在Windows容器中如何使用基于DirectX构建的GPU加速功能。

前提条件

- 已创建Kubernetes集群，且集群版本不低于v1.20.4。具体操作，请参见创建Kubernetes托管版集群。
  - Windows节点仅支持部署Docker及Containerd两种容器运行时。更多信息，请参见如何选择容器运行时？。
  - Windows节点的实例规格必须是GPU计算型或GPU虚拟化型。更多信息，请参见GPU计算型或GPU虚拟化型。
  - Windows节点必须安装已激活License的GRID驱动，可通过以下两种方式获取GRID驱动：
    - 如果您已经是NVIDIA的企业用户，可以通过NVIDIA的企业许可站点下载并安装对应的GRID驱动。
    - 如果您尚未是NVIDIA的企业用户，可以使用阿里云提供的Windows Server 2019 GRID 9驱动预装镜像。
- 已创建Windows节点池。具体操作，请参见创建Windows节点池。
- 已通过kubectl连接ACK集群。具体操作，请参见步骤二：选择集群凭证类型。

背景信息

DirectX（Direct eXtension，简称DX）是一种应用程序接口（API）。DirectX可以使以Windows为平台的游戏和多媒体程序获得更高的执行效率，加强3D图形和声音效果，并向设计人员提供一个共同的硬件驱动标准，降低安装及设置硬件的复杂度。DirectX可以允许GPU从事更多的通用计算工作，同时减轻过载，鼓励开发人员更好地将GPU作为并行处理器使用。

步骤一：为Windows节点安装DirectX设备插件

将DirectX设备插件以DaemonSet方式部署到Windows节点上。

1. 使用以下内容创建*directx-device-plugin-windows.yaml*文件。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 labels:
  k8s-app: directx-device-plugin-windows
 name: directx-device-plugin-windows
 namespace: kube-system
spec:
 revisionHistoryLimit: 10
 selector:
  matchLabels:
```

```
  matchLabels:
    k8s-app: directx-device-plugin-windows
  template:
    metadata:
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ""
      labels:
        k8s-app: directx-device-plugin-windows
    spec:
      tolerations:
        - operator: Exists
      # since 1.18, we can specify "hostNetwork: true" for Windows workloads, so we can deploy an application wi
thout NetworkReady.
      hostNetwork: true
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: type
                    operator: NotIn
                    values:
                      - virtual-kubelet
                  - key: beta.kubernetes.io/os
                    operator: In
                    values:
                      - windows
                  - key: windows.alibabacloud.com/deployment-topology
                    operator: In
                    values:
                      - "2.0"
                  - key: windows.alibabacloud.com/directx-supported
                    operator: In
                    values:
                      - "true"
              - matchExpressions:
                  - key: type
                    operator: NotIn
                    values:
                      - virtual-kubelet
                  - key: kubernetes.io/os
                    operator: In
                    values:
                      - windows
                  - key: windows.alibabacloud.com/deployment-topology
                    operator: In
                    values:
                      - "2.0"
                  - key: windows.alibabacloud.com/directx-supported
                    operator: In
                    values:
                      - "true"
      containers:
        - name: directx
          command:
            - pwsh.exe
            - -NoLogo
            - -NonInteractive
```

```
        - -File
        - entrypoint.ps1
      # 根据不同集群的地域，您需修改以下镜像地址中的地域<cn-hangzhou>信息。
      image: registry-vpc.cn-hangzhou.aliyuncs.com/acs/directx-device-plugin-windows:v1.0.0
      imagePullPolicy: IfNotPresent
      volumeMounts:
       - name: host-binary
         mountPath: c:/host/opt/bin
       - name: wins-pipe
         mountPath: \\.\pipe\rancher_wins
    volumes:
     - name: host-binary
       hostPath:
         path: c:/opt/bin
         type: DirectoryOrCreate
     - name: wins-pipe
       hostPath:
         path: \\.\pipe\rancher_wins
```

2. 执行以下命令安装DirectX设备插件。

```
kubectl create -f directx-device-plugin-windows.yaml
```

步骤二：部署使用基于DirectX的GPU加速的Windows工作负载

DirectX设备插件可以为Windows容器自动添加 class/<interface class GUID> 设备，以支持调用ECS实例主机的DirectX服务。更多信息，请参见Windows上的容器中的设备。在需要使用GPU加速的Windows工作负载内添加以下resources资源信息并重新部署：

```
spec:
 ...
 template:
  ...
  spec:
   ...
   containers:
    - name: gpu-user
      ...
+     resources:
+       limits:
+         windows.alibabacloud.com/directx: "1"
+       requests:
+         windows.alibabacloud.com/directx: "1"
```

> **注意** 上述配置不会将整个ECS实例主机的GPU资源专门分配给容器，也不会阻止ECS实例主机上的GPU被别的应用访问。相反，GPU资源在ECS实例主机和容器之间动态调度，这表示您可以在主机上运行多个Windows容器，且每个容器都可以使用支持硬件加速的DirectX功能。

关于Windows容器中的GPU加速的更多信息，请参见Windows容器中的GPU加速。

验证在Windows工作负载是否成功使用GPU加速功能

在Windows节点上添加DirectX设备插件后，使用以下示例应用验证DirectX设备插件是否成功部署到Windows节点上。

1. 使用以下内容创建*gpu-job-windows.yaml*文件。

```
apiVersion: batch/v1
kind: Job
metadata:
 labels:
  k8s-app: gpu-job-windows
 name: gpu-job-windows
 namespace: default
spec:
 parallelism: 1
 completions: 1
 backoffLimit: 3
 manualSelector: true
 selector:
  matchLabels:
   k8s-app: gpu-job-windows
 template:
  metadata:
   labels:
    k8s-app: gpu-job-windows
  spec:
   restartPolicy: Never
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
       - matchExpressions:
         - key: type
           operator: NotIn
           values:
            - virtual-kubelet
         - key: beta.kubernetes.io/os
           operator: In
           values:
            - windows
       - matchExpressions:
         - key: type
           operator: NotIn
           values:
            - virtual-kubelet
         - key: kubernetes.io/os
           operator: In
           values:
            - windows
   tolerations:
    - key: os
      value: windows
   containers:
    - name: gpu
      # 根据不同集群的地域，您需修改以下镜像地址中的地域<cn-hangzhou>信息。
      image: registry-vpc.cn-hangzhou.aliyuncs.com/acs/sample-gpu-windows:v1.0.0
      imagePullPolicy: IfNotPresent
      resources:
       limits:
        windows.alibabacloud.com/directx: "1"
       requests:
        windows.alibabacloud.com/directx: "1"
```

> **说明**
>
> - 镜像 `registry-vpc.{region}.aliyuncs.com/acs/sample-gpu-windows` 是阿里云容器服务提供的Windows GPU加速容器示例镜像（该镜像基于microsoft-windows制作。更多信息，请参见mcr.microsoft.com/windows）。
> - 因镜像文件较大（文件大小为15.3 GB），所以在部署应用时拉取镜像需要较长时间，请耐心等待。
> - 该示例通过WinMLRunner产生模拟输入数据，对 `gpu-job-windows` 任务使用GPU加速后，通过Tiny YOLOv2模型进行100次评估，最终输出相应的性能测量数据。

2. 执行以下命令创建示例应用。

```
kubectl create -f gpu-job-windows.yaml
```

3. 执行以下命令查看示例应用gpu-job-windows的日志信息。

```
kubectl logs -f gpu-job-windows
```

预期输出：

```
INFO: Executing model of "tinyyolov2-7" 100 times within GPU driver ...
Created LearningModelDevice with GPU: NVIDIA GRID T4-8Q
Loading model (path = c:\data\tinyyolov2-7\model.onnx)...
================================================================
Name: Example Model
Author: OnnxMLTools
Version: 0
Domain: onnxconverter-common
Description: The Tiny YOLO network from the paper 'YOLO9000: Better, Faster, Stronger' (2016), arXiv:1612.08
242
Path: c:\data\tinyyolov2-7\model.onnx
Support FP16: false
Input Feature Info:
Name: image
Feature Kind: Image (Height: 416, Width:  416)
Output Feature Info:
Name: grid
Feature Kind: Float
```

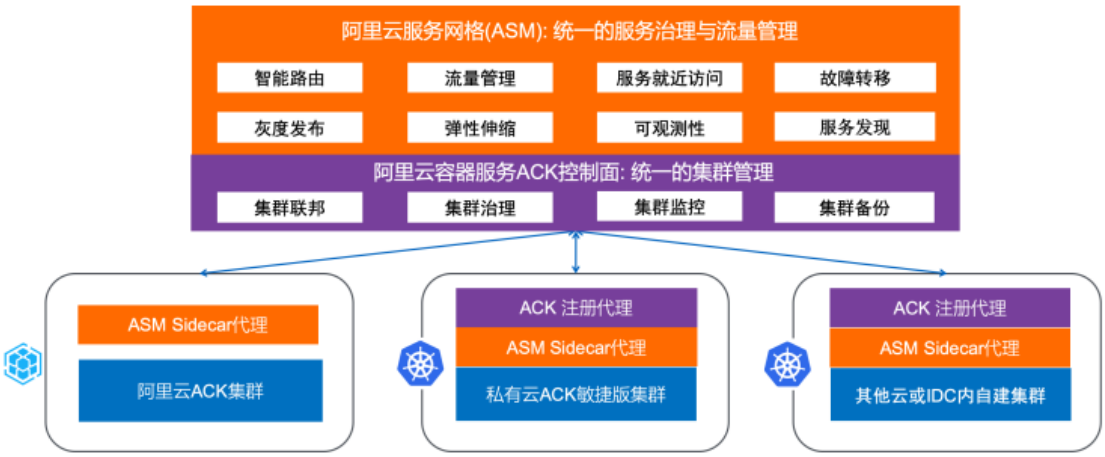
从预期输出可得，在示例应用gpu-job-windows中已成功使用GPU加速功能。

25.注册集群

25.1. 注册集群概述

ACK ONE（Open Nexus Everywhere）注册集群帮助您将云下Kubernetes集群接入云端，快速搭建混合云集群，可以将本地数据中心Kubernetes集群或其他云厂商Kubernetes集群接入阿里云容器服务管理平台统一管理的集群形态。本文介绍注册集群支持的功能和关联的云资源。

为什么需要注册的集群



在日常运维过程中，您可能同时运维着云上ACK集群、本地数据中心自建Kubernetes集群甚至其他云厂商的容器集群，在多云或混合云多集群的云架构下，如果您有以下需求，可以考虑使用阿里云注册集群。

- 通过阿里云容器服务控制台统一管理云上ACK集群和本地数据中心自建Kubernetes集群，包括一致的运维体验和安全治理能力。例如，使用容器服务控制台统一管理集群和应用，统一的日志、监控及告警体系，统一的阿里云账号及RAM用户的授权管理等。
- 扩展本地数据中心自建Kubernetes集群使用云上计算资源进行弹性扩缩容。例如，手动或自动扩缩容云上ECS或神龙裸金属，并添加为本地数据中心自建Kubernetes集群的云上计算节点等。
- 使用阿里云容器服务应用中心功能统一管理云上云下Kubernetes集群中的应用生命周期和GitOps交付，需要先将云下Kubernetes集群接入到注册集群，成为与云上ACK集群处于同一控制平面的容器集群。
- 使用阿里云服务网格产品对云上云下Kubernetes集群进行统一的服务治理，需要先将云下Kubernetes集群接入到注册集群，成为与云上ACK集群处于同一控制平面的容器集群。

安全特性

在注册外部Kubernetes集群时，ACK Register Agent组件提供了多集群的管理能力，同时提供了在外部集群上应用的部署和生命周期管理能力。

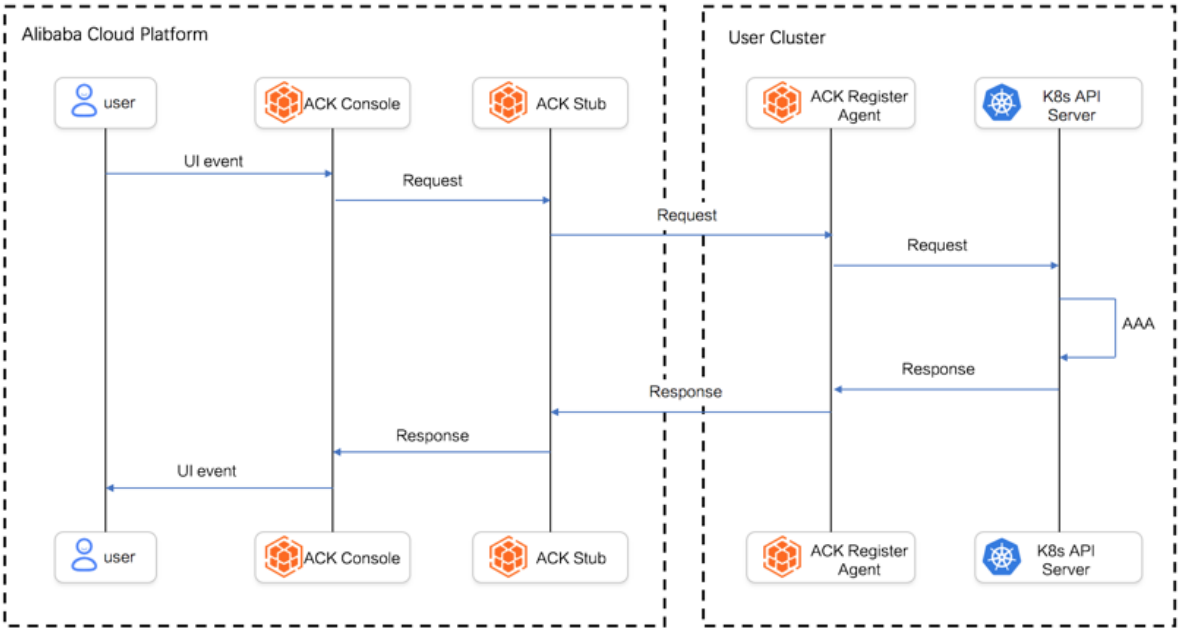
组件说明

组件名称	组件功能
ACK Console	ACK容器服务管理控制台。
ACK Register Agent	ACK集群注册Agent，以Deployment形式部署在指定外部集群中，用于接收ACK集群注册代理组件的请求并转发给对应的K8s API Server，同时接收K8s API Server的返回并发送给ACK集群注册代理组件。

组件名称	组件功能
ACK Stub	ACK集群注册代理组件，部署在容器服务管控侧，每一个注册外部集群都会部署一个与之对应的Stub代理组件，用于代理转发ACK管控组件和指定外部集群中注册Agent组件之间的请求。
K8s API Server	注册集群中运行的K8s API Server组件。

连接架构

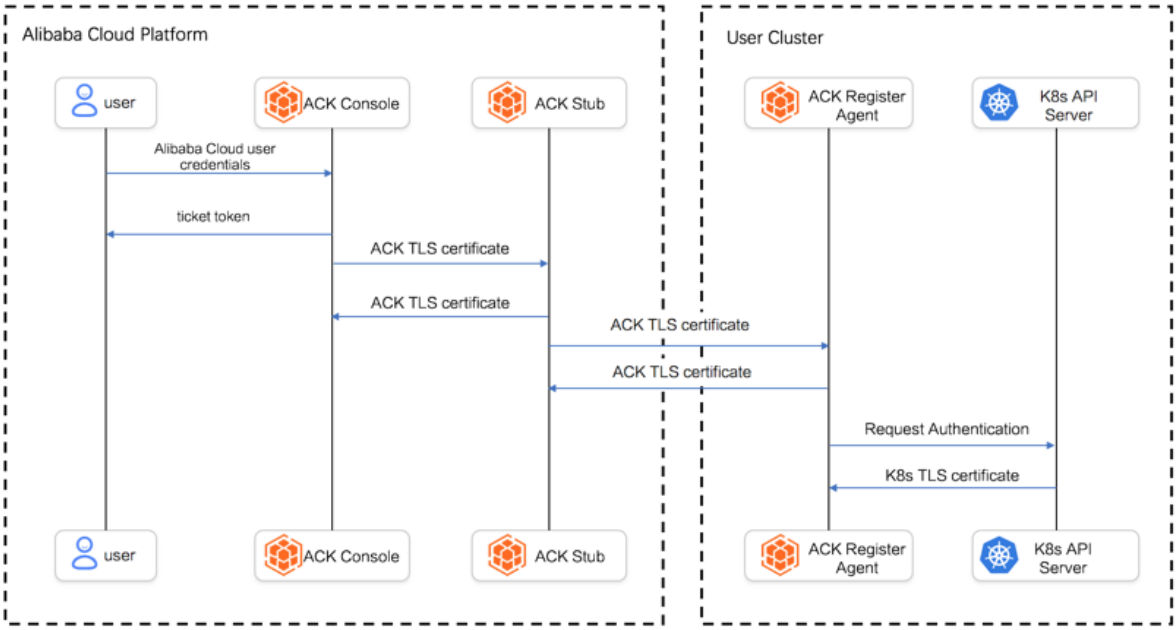
通过ACK Stub和ACK Register Agent组件，可以将您在ACK控制台上的请求下发到指定的外部集群API Server，其中Register Agent组件会以Deployment双实例的形式部署在指定的外部集群中，并且和ACK管控平台中的Stub代理组件连通。Agent组件通过下图中的流程将请求发送到您指定的API Server。



当用户完成外部集群注册流程后，ACK Register Agent组件会完成部署并与ACK管控侧建立TLS 1.2的长链接。对于已授权用户或授权后的ACK管控服务，ACK Console组件通过链路向指定的外部集群发送请求，请求会首先发送到Stub代理组件，然后请求被转发到对应的Agent，并最终由Agent组件将请求发送到外部API Server。API Server在收到请求后，会经过认证（Authentication）、鉴权（Authorization）和准入控制（Admission Control）流程，在完成审计后返回请求。请求返回的路径同样经由Agent，Stub代理组件并最终返回给ACK管控服务。所有请求在链路上都经过了认证鉴权流程，保证了集群访问的安全可控。

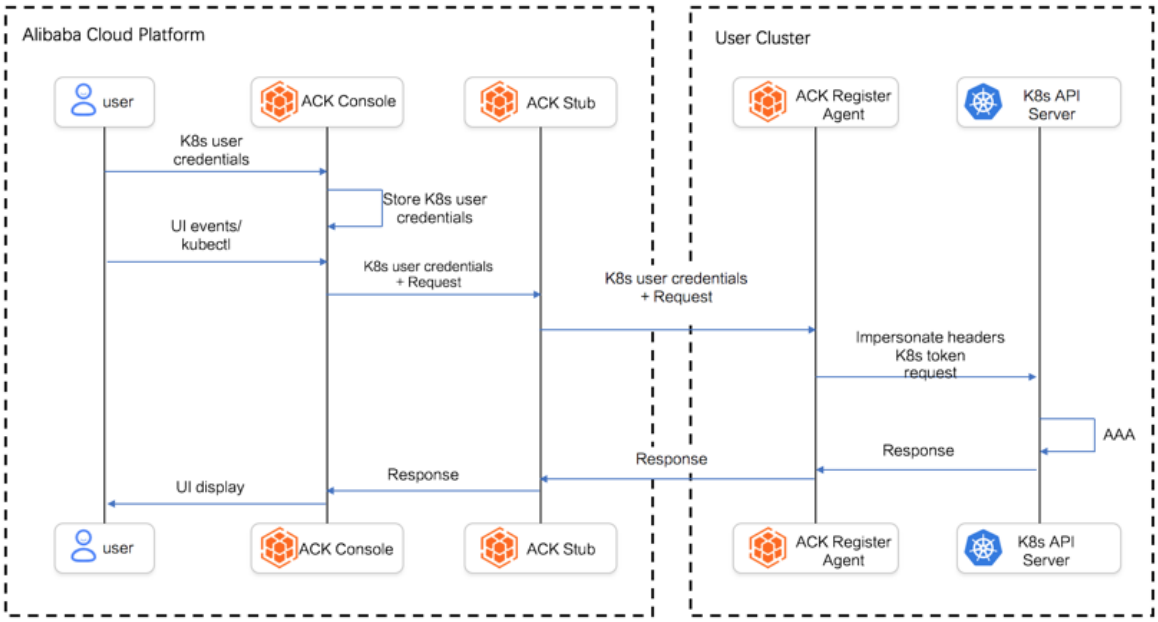
组件访问安全

架构所有组件之间的访问均需要经过认证授权，所有访问目标都是经过安全校验后的合法对象，保证数据只在可信的对象之间传输。



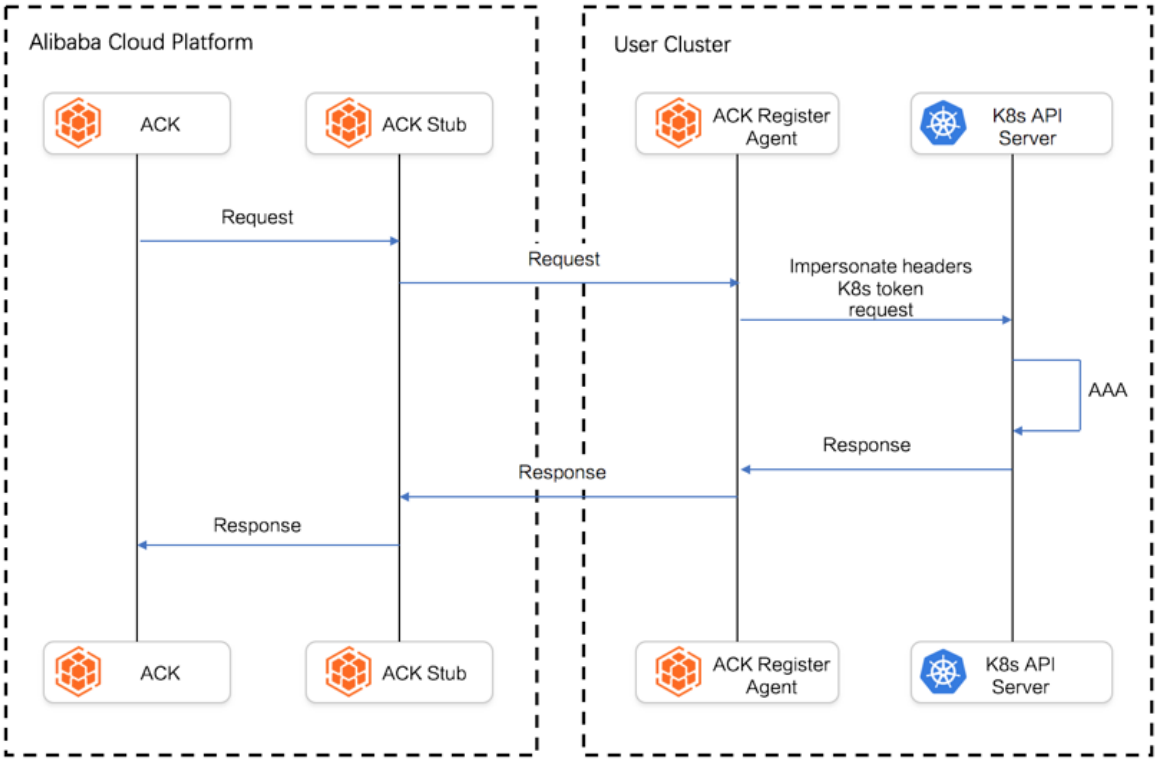
在认证上，访问组件之间通过RAM账号的登录认证和组件之间的双向TLS认证，保证所有访问链接数据在传输过程中的TLS加密。而鉴权采用基于阿里云RAM服务的访问控制和x509证书校验中的白名单机制。

- 请求链路安全
  在多云混合云管理请求链路上的所有访问凭证均包含了请求者的身份信息，包括用户使用阿里云容器服务下发的访问凭证和容器服务内部组件访问凭证，以保证所有发送到外部集群API Server的请求都能够经过鉴权和审计处理。
- 终端用户请求链路
  所有终端用户的请求都需要使用ACK管控签发的带有用户身份的集群访问凭证，对于外部集群的访问流程如下图所示。



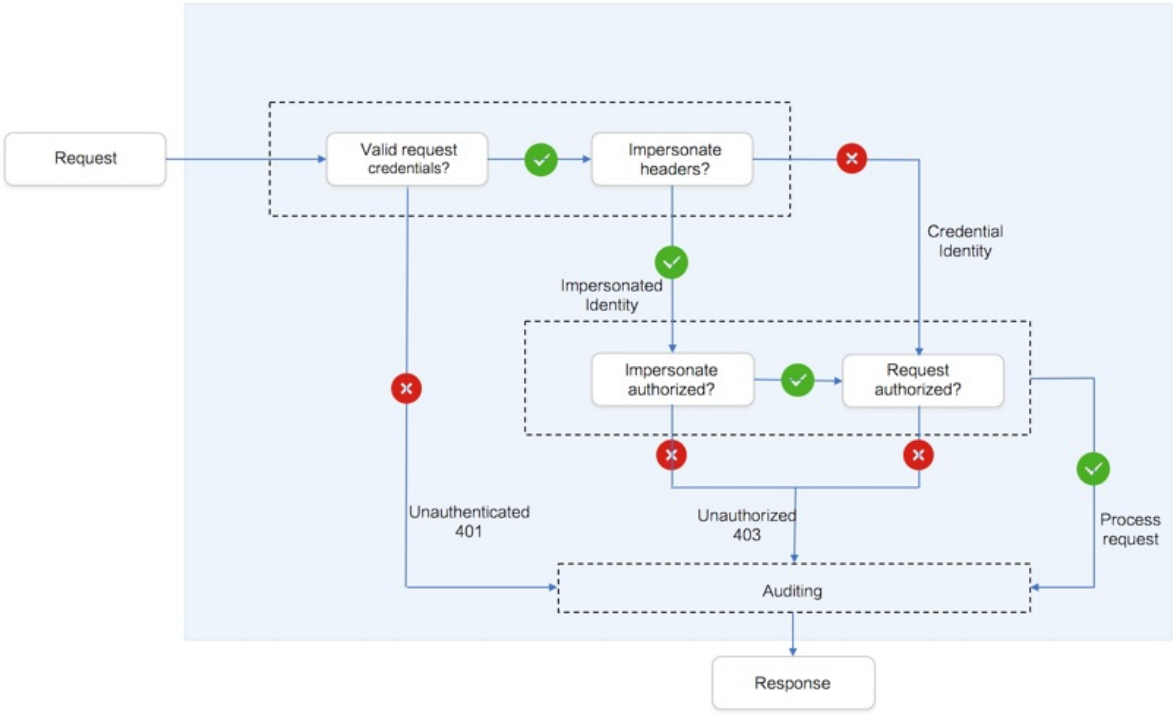
在阿里云平台侧，集群的管理人员可以通过RAM访问控制策略配置指定终端用户对外部集群的控制权限。经过RAM鉴权的用户可以在容器服务控制台获取指定外部集群的访问凭证，该访问凭证会作为认证凭据在Stub和Agent组件处认证，且组件间所有的访问链路都经过TLS加密。在Agent组件完成认证后，会将用户访问凭证中的身份信息作为请求目标API Server的身份扮演header，在API Server中会根据接收到的用户身份完成认证，鉴权和请求审计等操作。

- 服务组件请求链路
  ACK和ASM服务内部组件访问外部API Server的请求链路与终端用户访问链路类似，首先管控组件会使用集群注册时下发的Agent组件访问证书发起请求，在Stub和Agent组件完成请求认证后，经由Agent以身份扮演的方式向目标API Server作7层代理转发，最终在API Server完成请求的RBAC鉴权和审计。整个请求链路均经过了加密保证数据传输的安全性。



集群内部安全

Agent组件请求在目标集群API Server中的请求流程如下图所示。



在集群API Server内部，首先会在认证阶段校验请求凭据的合法性。

- 如果请求凭据不是一个合法的凭据，则API Server会直接返回401的认证失败错误。
- 如果认证通过，API Server会查看请求是否有指定格式的身份扮演header，如果存在的话会以指定的身份扮演身份去进行下一段的鉴权流程。

在鉴权流程中，API Server会判断传入的用户身份是否有对目标资源的指定操作权限。

- 如果鉴权失败，则API Server返回403的鉴权失败错误。
- 如果鉴权成功，则经过审计流程后返回请求结果。

Agent组件是Go语言编写的代理转发和集群注册组件，代码的编写和发布均经过了阿里云相关团队的安全审核。与此同时，指定集群的管理员也需要根据Kubernetes相关的最佳安全实践规范保护集群节点的安全性，通过对基础设施和应用相关的安全配置保障Agent组件和集群的安全性。

注册集群与ACK集群的能力对比

分类	能力	ACK集群	注册集群
集群访问	使用kubectl和kubeconfig文件访问集群	✔	✔
	使用控制台访问	✔	✔
	管理名空间	✔	✔
	管理Pod	✔	✔
	管理工作负载	✔	✔
	管理持久化卷和Storage Classes	✔ 支持云盘、本地盘、NAS、CPFS、OSS。	✔ 支持的存储类型取决于注册集群的实际环境。

分类	能力	ACK集群	注册集群
K8s核心概念	管理Service	✔	✔
	管理Ingress	✔	✔
	管理网络策略	✔ 支持Terway网络模式。	✔ 取决于注册集群的实际环境。
	管理Configmap	✔	✔
	管理Secret	✔	✔
	管理RBAC	✔	✔
	管理Pod Security Policy	✔	✔
	管理Resource Quotas	✔	✔
	管理CRD	✔	✔
	支持水平弹性伸缩 (HPA)	✔	✔
	支持Pod RuntimeClass	✔ 仅支持阿里云安全沙箱容器。	无
	支持弹性容器实例（ECI）	✔	✔ 配置方法请参见扩容弹性容器实例ECI。
应用管理	部署、管理Helm Charts应用	✔	✔
	支持GitOps应用交付链	✔	无
	支持Knative Addon	✔	✔
	支持Istio Addon	✔	✔
	支持托管服务网格ASM	✔	✔ 配置方法请参见通过ASM管理外部注册Kubernetes集群应用。
安全治理	支持RAM统一身份认证和RBAC权限管理	✔	✔
	支持基于日志服务（SLS）的集群审计	✔	✔
	支持云安全中心	✔	无
	配置巡检	✔	✔ 配置方法请参见使用配置巡检功能检查注册集群Workload安全隐患。

分类	能力	ACK集群	注册集群
可观测性	支持事件中心（NPD）	✔	✔ 配置方法请参见创建并使用Kubernetes事件中心。
	支持Ingress图表	✔	✔ 配置方法请参见Ingress访问日志分析与监控。
	支持日志采集	✔	✔ 配置方法请参见将日志服务接入注册集群。
	支持ARMS-应用监控	✔	✔ 配置方法请参见将应用实时监控服务ARMS接入注册集群。
	支持ARMS-Prometheus	✔	✔ 配置方法请参见将arms-prometheus接入注册集群。
	支持AHAS-架构感知	✔	✔ 配置方法请参见将AHAS-架构感知接入注册集群。
	支持AHAS-应用限流	✔	✔ 配置方法请参见将AHAS-应用限流接入注册集群。
	支持Node Problem Detector（NPD）	✔	✔ 配置方法请参见将事件中心接入注册集群。
	支持Metrics Adpater	✔	✔ 配置方法请参见将alibaba-cloud-metrics-adapter接入注册集群。
	支持云监控集成	✔	无
	支持KMS集成	✔	无
集群生命周期管理	管理节点	✔	✔
	管理节点池	✔	无
	支持集群弹性伸缩	✔	✔ 取决于注册集群的实际环境。
	支持证书轮转	✔	无
	支持集群升级	✔	无

分类	能力	ACK集群	注册集群
	支持系统组件管理	✔	无
	支持集群检查	✔	无

注册集群的关联云资源或云服务

创建一个注册集群，用户侧将会涉及到以下云资源或云服务：

- 专有网络VPC和VSwitch。要求专有网络VPC有访问公网能力，您需要为其配置绑定公网EIP或者设置SNAT，产生费用。
- 一个弹性网卡ENI，不产生费用。
- 一个私网负载均衡实例，产生实例规格费用。关于实例规格的费用，请参见实例规格。
- （可选）一个弹性公网IP，产生流量费用。关于流量费用，请参见按量付费。

此外，如果您希望扩展本地数据中心自建Kubernetes使用云上计算资源进行弹性扩缩容，还涉及到以下云资源或云服务：

- 云上弹性ECS、裸金属、ECI等计算资源，根据实际使用计算资源收费。
- 开通ESS云服务，免费。

其余按使用的功能收取费用。

25.2. 混合集群部署

25.2.1. 创建注册集群并接入本地数据中心集群

注册集群是用于将本地数据中心Kubernetes集群或其他云厂商Kubernetes集群接入阿里云容器服务管理平台统一管理的集群形态。本文将介绍如何创建一个注册集群并接入本地数据中心Kubernetes集群。

前提条件

登录RAM管理控制台和弹性伸缩控制台开通相应的服务。

操作步骤

1. 创建接入集群。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击页面右上角的**创建集群**。
   iv. 在**注册集群**页签，完成创建集群配置项。

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。

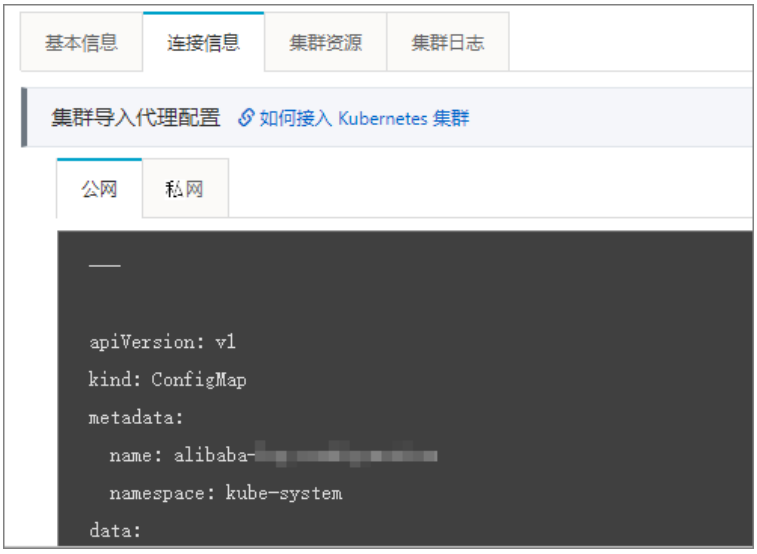
配置项	描述
资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
地域	选择集群所在的地域。
专有网络	您可以在已有VPC列表中选择所需的VPC和交换机。
虚拟交换机	您可以在选定的VPC下选择所需的节点虚拟交换机。
网络插件	选择**不开启**时，表示不使用节点池扩容云上计算资源则无需配置网络插件。 选择**Terway**并选中**IPvlan**时，表示采用IPvlan+ebpf作为网卡共享模式虚拟化技术，只能使用Alibaba Cloud Linux2系统。
Pod虚拟交换机	您可以在选定的VPC下选择所需的Pod虚拟交换机。
Service CIDR	如无特殊需求，使用默认的172.21.0.0/20即可。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 **注意** 删除默认创建的SLB实例将会导致无法访问API Server。
绑定EIP	设置是否启用绑定EIP。选中此选项，会在集群中自动绑定EIP，用于建立集群链接。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 **说明** ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。
日志服务	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。更多详情，请参见通过日志服务采集Kubernetes容器日志。

配置项	描述
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。
标签	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、acs:、http://或https://开头的字符串，不区分大小写，最多64个字符。 ■ *值*不能是aliyun、acs:、http://或https://开头的字符串，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。
服务协议	创建集群前，需阅读并同意*服务条款*及*免责申明*。

v. 在页面右侧，单击**创建集群**，完成集群创建。
您可以在集群列表，看到您创建的集群。



2. 将目标集群接入注册集群中。
   i. 在新集群（本例中为test-external-cluster1）右侧单击**操作**列下的**详情**。
   ii. 在**集群信息**页面单击**连接信息**页签。
   iii. 在**集群导入代理配置**区域根据需要选择**公网**或者**私网**，然后单击右侧的**复制**，将公网或私网页签的内容复制到一个文件中，并执行kubectl命令，将目标集群注册至新集群中。
   例如，您可以新建*agent.yaml*文件，将以下内容复制到*agent.yaml*文件中，并在目标集群中执行`kubectl apply -f agent.yaml`命令。



iv. 在目标集群中执行以下命令，查看代理运行状况。

```
kubectl -n kube-system get pod |grep ack-cluster-agent
```

预期输出：

```
ack-cluster-agent-5f7d568f6-6fc4k       1/1   Running  0     9s
ack-cluster-agent-5f7d568f6-tf6fp       1/1   Running  0     9s
```

注册成功后，您可以在容器服务管理控制台的Kubernetes**集群列表**页面，看到该集群的状态为**运行中**。

执行结果

在**集群列表**页面中，找到注册集群，单击**操作**列下的**详情**，查看新集群的**基本信息**和**连接信息**。

您可以使用该kubeconfig连接目标集群，进行应用负载的部署。连接集群的具体操作步骤，请参见通过kubectl连接Kubernetes集群。

相关文档

- 注册集群概述

25.2.2. 创建混合集群

混合集群是通过阿里云注册集群接入本地数据中心自建Kubernetes集群后，为其扩容云上计算节点，成为同时管理云上云下计算资源的容器集群。本文介绍如何创建混合集群。

前提条件

- 本地数据中心自建Kubernetes集群的网络与云上注册集群使用的专有网络VPC互联互通，网络互联互通包括计算节点网络和容器网络互联互通。您可以通过云企业网产品服务实现网络互通。更多信息，请参见教程概览。
- 目标集群必须使用注册集群提供的私网集群导入代理配置接入注册集群。
- 通过注册集群扩容的云上计算节点能够访问本地数据中心自建Kubernetes集群的API Server。

操作步骤

1. 规划Terway场景的容器网络。具体操作，请参见网络规划。
2. 本地数据中心Kubernetes网络与云上专有网络VPC互联互通。具体操作，请参见方案介绍。
3. 创建注册集群并接入本地数据中心自建Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
4. 配置容器网络插件。具体操作，请参见配置混合集群网络。
5. 配置自定义节点添加脚本。具体操作，请参见使用自定义节点添加脚本。
6. 创建节点池并扩容云上计算节点。具体操作，请参见创建节点池并扩容。

   如果需要设置自动弹性伸缩。具体操作，请参见配置自动弹性伸缩。

25.2.3. 通过SAG vCPE实现云上ACK与云下K8s集群互通

智能接入网关SAG（Smart Access Gateway）vCPE是智能接入网关软件镜像版，通过将SAG vCPE镜像部署在您网络的宿主机中，使宿主机作为一个CPE（Customer-premises equipment）设备为您提供上云服务。本文为您介绍如何通过SAG vCPE将已经部署在用户云下网络环境中的K8s集群接入阿里云ACK，实现云上云下网络互通。

前提条件

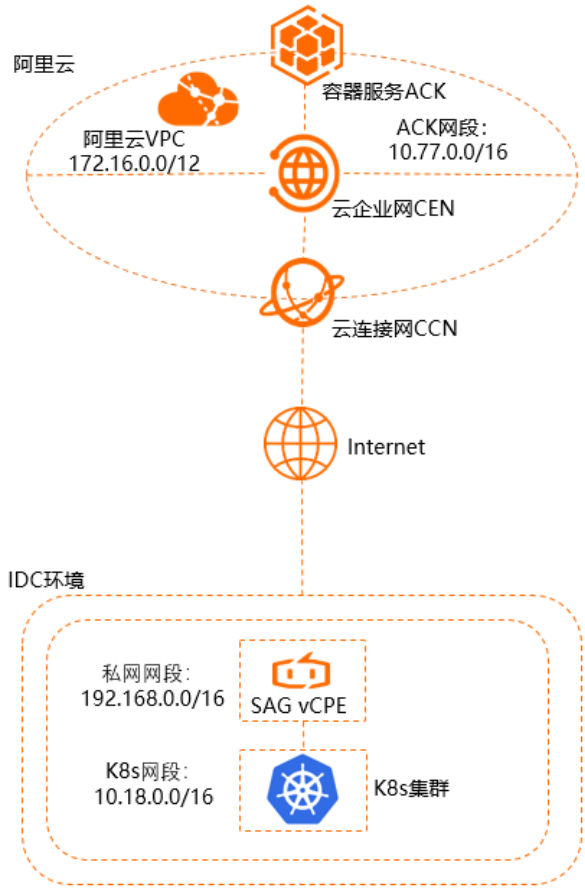
- 您已具备操作IDC网络管理和配置的权限。具体可以咨询IDC管理员。
- 您已经在阿里云平台创建了专有网络VPC（Virtual Private Cloud）并部署了相关云服务。具体操作，请参见搭建IPv4专有网络。
- 您已经了解阿里云VPC中云服务所应用的安全组规则，并确保安全组规则允许云下网络访问阿里云VPC内的资源。具体操作，请参见查询安全组规则和添加安全组规则。

- 您已了解阿里云ACK集群创建以及网络规划的全部内容，并确保ACK集群的网络与云下网络环境不存在网段重叠的情况。具体操作，请参见创建Kubernetes托管版集群和Kubernetes集群网络规划。

示例场景

某企业在自己的专有IDC环境部署了K8s集群，在阿里云平台某地域部署了ACK。该企业计划使用SAG vCPE产品将云上ACK和云下的K8s服务连通，实现资源互访。

您可以在IDC内的一台实例（物理机或者虚拟机）中部署SAG vCPE镜像，使该实例作为一台SAG vCPE设备连接阿里云。SAG vCPE设备连接至阿里云后，可通过阿里云云连接网和云企业网实现IDC内资源与阿里云VPC内资源的互访。本文场景示例图如下所示。



从上图可以得出云上和云下的网络情况如下表所示。

类型	私网网段（VPC网段）	容器网段
阿里云云上ACK集群	172.16.0.0/12	10.77.0.0/16
IDC中K8s集群	192.168.0.0/16	10.18.0.0/16

部署流程



1. 创建SAG vCEP实例：在智能接入网关管理控制台创建SAG vCPE实例，创建后您可以通过SAG vCPE实例管理SAG vCPE设备。
2. 部署SAG vCEP镜像：在IDC机房中选择一台机器，部署SAG vCPE镜像，该机器可作为SAG vCPE设备连接IDC网络至阿里云。

3. 配置阿里云侧网络：在智能接入网关管理控制台对SAG vCPE设备进行网络配置，以便SAG vCPE设备能正常接入阿里云。
4. 配置IDC侧网络：在IDC中进行网络配置，实现IDC和阿里云ACK资源互通。
5. 测试连通性：分别测试云上云下主机间的网络连通性、云上云下容器Pod间的网络连通性。

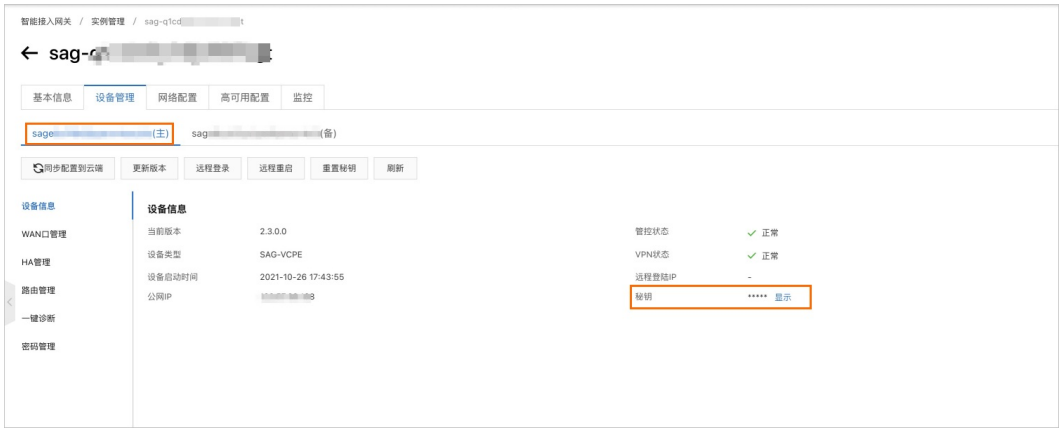
步骤一：创建SAG vCEP实例

您需要在智能接入网关管理控制台创建SAG vCPE实例，创建后您可以通过SAG vCPE实例管理SAG vCPE设备。

1. 登录智能接入网关管理控制台。
2. 在**智能接入网关**页面，选择**购买智能接入网关 > 创建智能接入网关（VCPE）**。
3. 在购买页面，完成以下配置，然后单击**立即购买**并完成支付。

配置	说明	本文示例
区域	选择SAG vCPE实例所属的区域。	中国内地
实例名称	输入SAG vCPE实例的名称。 实例名称可为空。 长度为2~128个字符，以大小写字母或中文开头，可包含数字，半角句号（.），下划线（_）或短划线（-）。	Demo
实例类型	选择智能接入网关vCPE的实例类型。	SAG-vCPE
版本	选择智能接入网关vCPE的版本。	基础版
使用方式	选择SAG vCPE的使用方式。默认为**双机**。 双机方式下一个SAG vCPE实例中默认可以连接两台SAG vCPE设备。您可以配置两台SAG vCPE设备为主备模式，共同将本端网络接入阿里云，提高您网络的可用性。本文中只使用主设备。	双机
带宽峰值	网络通信的带宽峰值。单位：Mbps。	50 Mbps
购买数量	选择需要创建的SAG vCPE实例的数量。	1
购买时长	选择购买时长。 您可以选中下方的**到期自动续费**进行自动续费。	1个月
资源组	选择SAG vCPE实例所属的资源组。	无

4. 返回智能接入网关管理控制台，在顶部菜单栏，选择已创建实例的区域。
5. 在左侧导航栏，选择**智能接入网关**。
6. 在**智能接入网关**页面，单击已创建的实例ID。
7. 在实例详情页面，单击**设备管理**页签，查看并记录当前SAG vCPE主设备的序列号和密钥，用于后续SAG vCPE实例和SAG vCPE设备的绑定。



步骤二：部署SAG vCEP镜像

为实现IDC和阿里云ACK集群的网络互通，您需要在IDC机房中选择一台机器，用于部署SAG vCPE镜像。部署完成后，该机器可作为SAG vCPE设备为您提供服务，连接IDC网络至阿里云。

1. 在IDC环境中选择一台机器。

   为了保证SAG vCEP镜像可以正常运行，选择的机器需要满足以下条件：

   - 支持安装以下类型的操作系统：
     - （推荐）64位，CentOS 7.6或以上规格。
     - 64位，Ubuntu 18.04或以上规格。
   - 支持安装3.10.0-957.21.3.el7.x86_64或以上规格的内核版本。
   - 有单独的可连接公网的网卡。
   - 支持远程登录。
   - 未运行业务系统。

2. 登录IDC机器，执行以下命令，下载脚本至实例的*/root*目录下。

   > **说明**
   > - 您可以将脚本下载至自定义路径内，请注意后续执行脚本时，路径需修改为您的自定义路径。
   > - 下载脚本后，请勿修改脚本内容以及脚本名称。

   根据您的宿主机是否部署在中国内地区域，选择不同的下载脚本命令。

   宿主机部署在中国内地区域 | 宿主机部署在非中国内地区域

   ```
   wget -O /root/sag_vcpe_v2.3.0_deployment.sh https://sdwan-oss-shanghai.oss-cn-shanghai.aliyuncs.com/vcpe_vm/sag_vcpe_v2.3.0_deployment.sh
   ```

3. 执行以下命令，为脚本赋予可执行权限。

   ```
   chmod +x /root/sag_vcpe_v2.3.0_deployment.sh
   ```

4. 执行脚本。

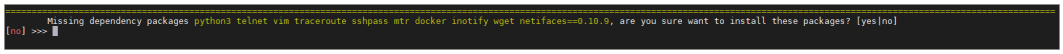
   ```
   /root/sag_vcpe_v2.3.0_deployment.sh -n sag**** -k X8==**** -t idc  -w eth0
   ```

   部分参数说明如下表。关于脚本参数的更多信息，请参见脚本参数说明。

参数	描述
-n	SAG vCPE设备的序列号。
-k	SAG vCPE设备的密钥。
-t	安装SAG vCPE镜像的宿主机所在的平台。取值： • aliyun（默认值）：表示SAG vCPE镜像部署在阿里云云服务器ECS（Elastic Compute Service）中。 • aws：表示SAG vCPE镜像部署在AWS EC2中。 • ens：表示SAG vCPE镜像部署在阿里云边缘节点服务ENS（Edge Node Service）的实例中。 • 如果您的SAG vCPE镜像部署在本地网络的服务器中，则本参数的取值可为 aliyun 、 aws 、 ens 之外的任意英文字符。
-w	WAN口的网卡名称。您可以通过 ifconfig 命令查看宿主机的网卡名称。

执行脚本时，系统会自动检测部署环境是否满足需求：

- 如果部署环境相关的组件安装不完整，系统会出现如下图的提示。您可以输入*yes*，系统将自动帮您安装相关组件。安装组件完成后，开始部署SAG vCPE镜像。



- 如果检测到部署环境已经满足需求，则会直接开始部署SAG vCPE镜像，镜像部署完成后系统会出现如下图的提示。



5. 部署完成后，执行 docker ps 命令，查看系统中是否已有以下两个容器。



- 如果系统已包含vsag-core和vsag-manager-base两个容器，则证明部署成功。
- 如果系统未包含上述两个容器，则说明部署失败，请提交工单处理。

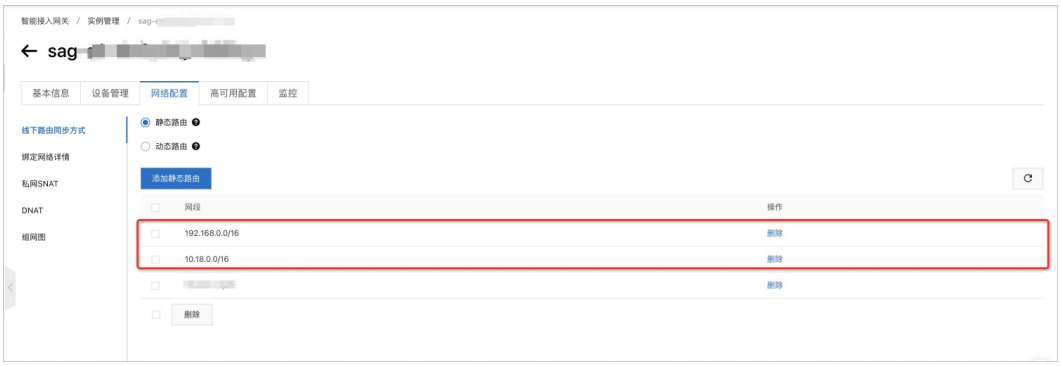
步骤三：配置阿里云侧网络

SAG vCPE镜像部署完成后，您还需要在智能接入网关管理控制台对SAG vCPE设备进行网络配置，以便SAG vCPE设备能正常接入阿里云。

1. 配置线下路由同步方式。
   i. 登录智能接入网关管理控制台，在顶部菜单栏，选择目标区域。
   ii. 在**智能接入网关**页面，找到目标实例，在**操作**列单击**网络配置**。
   iii. 在**网络配置**页签的**线下路由同步方式**区域，单击**添加静态路由**。
   iv. 在**添加静态路由**对话框中，输入IDC所在的私网网段，然后单击**确定**。

v. 再次单击**添加静态路由**，在**添加静态路由**对话框中，输入IDC中的K8s集群容器网段，然后单击**确定**。

两个网段都添加完成后，界面如下图所示。



2. 绑定云连接网。

云连接网是SAG的重要组成部分，SAG通过云连接网将您的网络接入阿里云。

i. 创建云连接网。具体操作，请参见创建云连接网。

> **说明** 云连接网所在区域需和SAG vCPE实例所在区域相同。

ii. 在智能接入网关管理控制台的左侧导航栏，选择**智能接入网关**。

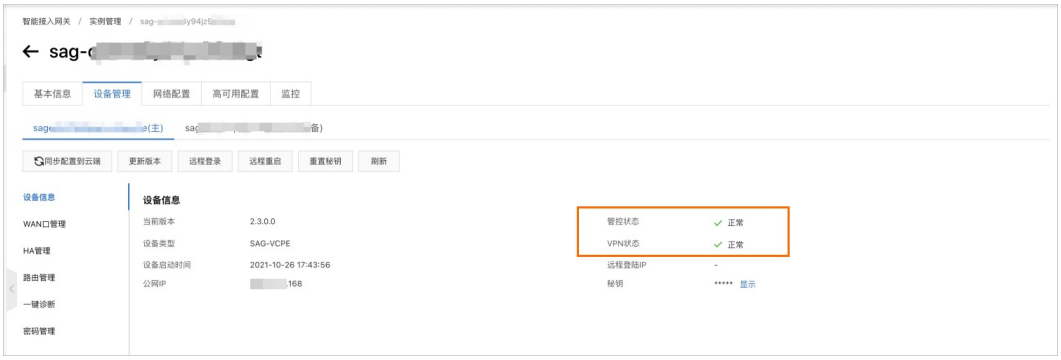
iii. 在**智能接入网关**页面，找到目标实例，在**操作**列单击**网络配置**。

iv. 在**网络配置**页签，选择**绑定网络详情**。

v. 在**已绑定同账号实例**区域下，单击**添加网络**，选择已创建的云连接网实例，然后单击**确定**。

vi. 单击**设备管理**页签，查看SAG vCPE设备的VPN状态和管控状态。

在您绑定云连接网后，SAG vCPE设备的VPN状态和管控状态均为正常，则表示SAG vCPE设备已接入阿里云。



3. 配置云企业网。

您需要通过以下操作将SAG vCPE实例连接到云企业网，并在云企业网中加载已创建的VPC实例。操作完成后，SAG vCPE实例和阿里云上VPC实例可学习到对方的路由，SAG vCPE设备可与阿里云VPC内的资源互通。

i. 在智能接入网关管理控制台的左侧导航栏，选择**云连接网**。

ii. 在**云连接网**页面，找到目标云连接网实例，在**操作**列单击**绑定云企业网**。

iii. 在**绑定云企业网**面板，选择要绑定的云企业网实例，然后单击**确定**。

您可以通过以下两种方式选择目标云企业网实例，本文选择**新建CEN**。

- **选择现有CEN**：如果您已经创建了云企业网，可以在下拉列表中，选择已创建的云企业网实例进行绑定。
- **新建CEN**：如果您未创建过云企业网，您可以在下方文本框中，输入云企业网实例名称，系统会为您新建云企业网实例并自动进行绑定。

> **说明** 云企业网实例名称长度为2~100个字符，以大小写字母或中文开头，可包含数字、下划线（_）或短划线（-）。

iv. 将已经创建的阿里云VPC实例，绑定到此云企业网中。具体操作，请参见加载网络实例。

最终VPC和云连接网加载到云企业网后，如下图所示。



步骤四：配置IDC侧网络

为实现IDC和阿里云资源互通，您还需要在IDC中进行网络配置，配置需要的具体命令请咨询IDC网络管理员。

1. 配置IDC静态路由示例。

在IDC网络配置路由，将要访问的阿里云ACK集群网段（如果需要访问云上的容器网段，也需要添加）下一跳指向安装了SAG vCEP的机器IP地址，由该实例完成和阿里云云上服务的互通。

```
ip route add 10.77.0.0/16 via 192.168.11.210
```

> **说明** 静态路由示例仅供参考，不同厂商的不同设备可能会不同。

2. 配置IDC环境的防火墙。

允许阿里云ACK集群和IDC在私网网段可相互通信。

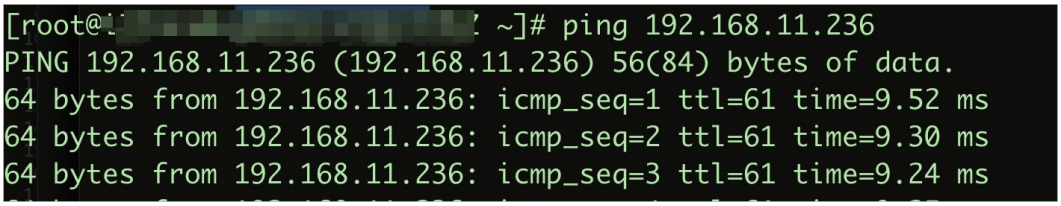
步骤五：测试连通性

1. 测试主机连通性。

i. 登录阿里云VPC内的一个ECS实例。具体操作，请参见连接方式概述。

ii. 执行 `ping` 命令，访问云下IDC环境中的某台主机实例，验证云上和云下主机网络是否正常。

经验证，阿里云VPC中的资源与云下IDC中的资源可正常通信，如下图所示。



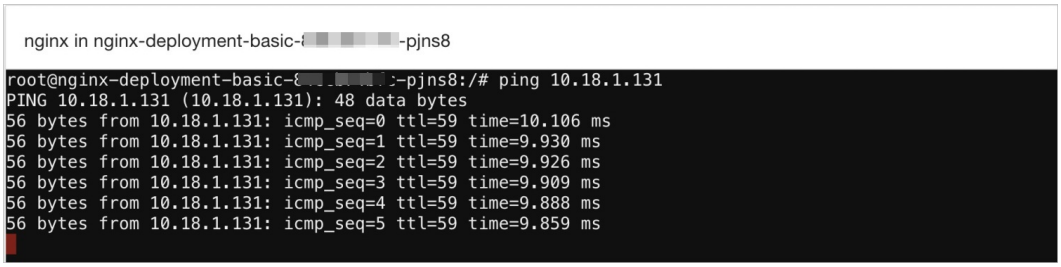
2. 测试容器Pod网络连通性。

   i. 分别在阿里云云上的ACK集群和云下IDC的K8s集群部署测试容器，YAML示例如下。

```
apiVersion: apps/v1 # 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:
  #  nodeSelector:
  #   env: test-team
   containers:
   - name: nginx
     image: nginx:1.7.9 #替换成您实际的<image_name:tags>。
     ports:
     - containerPort: 80
     resources:
      limits:
       cpu: "500m"
```

   ii. 进入到ACK集群容器内，执行 `Ping` 命令，访问云下IDC的K8s集群中容器Pod的IP，验证云上ACK和云下K8s的Pod网络是否正常。

   经验证，阿里云ACK集群中的Pod与云下IDC中K8s集群的Pod可正常通信，如下图所示。



总结

通过SAG vCEP结合云连接网络以及云企业网，您可以快速地完成云上和云下环境的快速组网，实现云上云下的网络互通。在这个过程中，为了降低可能的隐患，建议您提前做好网络规划，避免在生产环境中，由于网络冲突而带来的影响。

相关文档

- 什么是智能接入网关
- SAG vCPE介绍
- 什么是云企业网
- Kubernetes集群网络规划

25.3. 配置混合集群网络

混合集群中的容器网络插件包括本地数据中心中运行的容器网络插件和云上计算节点上运行的容器网络插件两部分。本文将介绍如何配置混合集群的网络。

前提条件

- 对于场景二：本地数据中心容器网络模式为BGP网络及场景三：本地数据中心容器网络为Host网络，在创建注册集群时需要配置Terway网络的参数。
  - 根据您的需要选择是否选中IPvlan。
  - 配置Pod虚拟交换机。
  - 配置Service CIDR。

  具体操作，请参见创建注册集群并接入本地数据中心集群。
- 已完成创建混合集群中步骤4之前的操作。

场景一：本地数据中心容器网络模式为覆盖（Overlay）网络

若本地数据中心内的容器网络模式为覆盖网络，则云上计算节点也可以复用此网络模式，只需要保证云上计算节点能够正常拉取容器网络插件守护进程集所使用的容器镜像即可。

例如，常见的覆盖网络模式有：

- Flannel VXLAN模式。
- Calico IPIP模式。
- Cilium VXLAN模式。

场景二：本地数据中心容器网络模式为BGP网络

若本地数据中心内的容器网络模式为BGP网络，那么云上计算节点需要使用Terway网络。关于云上云下容器网络互通，请参见云上边界路由器的BGP配置。

在此场景下，需要保证以下条件：

- 云下容器网络插件（如Calico BGP路由反射模式）的守护进程集不要被调度到云上计算节点。
- 同时也要保证Terway网络插件的守护进程集不要被调度到云下计算节点。

在注册集群中，通过注册集群节点池扩容的计算节点都会添加一个固定的节点标签：`alibabacloud.com/external=true`，可以作为区分云上云下节点的过滤条件。

例如，针对云下Calico网络插件，可以使用`nodeAffinity`设置其不被调度到拥有标签`alibabacloud.com/external=true`的节点上。其他任何运行于云下且不希望被调度到云上的工作负载，都可以使用这种方式设置。执行以下命令更新Calico网络插件。

```
cat <<EOF > calico-ds.pactch
spec:
 template:
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
       - key: alibabacloud.com/external
         operator: NotIn
         values:
         - "true"
EOF
kubectl -n kube-system patch ds calico-node -p "$(cat calico-ds.pactch)"
```

云上Terway网络插件默认只被调度到拥有标签`alibabacloud.com/external=true`的节点上。

场景三：本地数据中心容器网络为Host网络

由于本地数据中心内的容器网络使用的是Host网络，所以只需保证云上Terwav网络插件的守护进程集不被调度到云下即可。云上Terway网络插件默认只调度到拥有节点标签为 `alibabacloud.com/external=true` 的云上节点。

安装和配置Terway网络插件

在场景二和场景三下，需要为混合集群的云上节点安装和配置Terway网络插件。

步骤一：安装Terway插件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 在**组件管理**页面单击**网络**页签，在terway-eniip组件区域单击**安装**。



您可以使用kubeconfig连接注册集群并查看Terway网络插件守护进程集，在混合集群扩容云上节点之前，它将不会被调度到任何一个云下节点上。
执行以下命令查看Terway网络。

```
kubectl -nkube-system get ds |grep terway
```

预期输出：

```
terway-eniip  0     0     0    0      0      alibabacloud.com/external=true    16s
```

步骤二：配置Terway插件

在注册集群中，所有Addon组件都使用自定义的AccessKey来获取云资源的操作权限。

1. 创建RAM账号并为账号授予以下RAM Policy。

   具体操作，请参见自定义RAM授权策略。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:AttachNetworkInterface",
        "ecs:DetachNetworkInterface",
        "ecs:DeleteNetworkInterface",
        "ecs:DescribeInstanceAttribute",
        "ecs:AssignPrivateIpAddresses",
        "ecs:UnassignPrivateIpAddresses",
        "ecs:DescribeInstances",
        "ecs:ModifyNetworkInterfaceAttribute"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "vpc:DescribeVSwitches"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

2. 执行以下命令编辑ConfigMap *eni-config*并配置 `eni_conf.access_key` 和 `eni_conf.access_secret` 。

```
kubectl -n kube-system edit cm eni-config
```

*eni-config*的示例如下所示。

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: eni-config
 namespace: kube-system
data:
 eni_conf: |
 {
  "version": "1",
  "max_pool_size": 5,
  "min_pool_size": 0,
  "vswitches": {{.PodVswitchId}},
  "eni_tags": {"ack.aliyun.com":"{{.ClusterID}}"},
  "service_cidr": "{{.ServiceCIDR}}",
  "security_group": "{{.SecurityGroupId}}",
  "access_key": "",
  "access_secret": "",
  "vswitch_selection_policy": "ordered"
 }
 10-terway.conf: |
 {
  "cniVersion": "0.3.0",
  "name": "terway",
  "type": "terway"
 }
```

开启Terway组件的NetworkPolicy

在注册集群中Terway组件默认关闭NetworkPolicy。若您不需要开启NetworkPolicy，请跳过该步骤。更多信息，请参见使用网络策略Network Policy。

> **注意** Terway组件的NetworkPolicy功能依赖Calico CRD。若您的集群已经使用了Calico容器网络插件，开启Terway的NetworkPolicy功能时可能导致已有Calico容器网络异常。若有问题，请提交工单咨询。

1. 使用以下示例部署CRD资源。

```
kubectl apply -f - <<EOF ---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 name: felixconfigurations.crd.projectcalico.org
spec:
 scope: Cluster
 group: crd.projectcalico.org
 version: v1
 names:
  kind: FelixConfiguration
  plural: felixconfigurations
  singular: felixconfiguration
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 name: bgpconfigurations.crd.projectcalico.org
spec:
 scope: Cluster
 group: crd.projectcalico.org
```

```
  version: v1
  names:
    kind: BGPConfiguration
    plural: bgpconfigurations
    singular: bgpconfiguration
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: ippools.crd.projectcalico.org
spec:
  scope: Cluster
  group: crd.projectcalico.org
  version: v1
  names:
    kind: IPPool
    plural: ippools
    singular: ippool
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: hostendpoints.crd.projectcalico.org
spec:
  scope: Cluster
  group: crd.projectcalico.org
  version: v1
  names:
    kind: HostEndpoint
    plural: hostendpoints
    singular: hostendpoint
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: clusterinformations.crd.projectcalico.org
spec:
  scope: Cluster
  group: crd.projectcalico.org
  version: v1
  names:
    kind: ClusterInformation
    plural: clusterinformations
    singular: clusterinformation
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: globalnetworkpolicies.crd.projectcalico.org
spec:
  scope: Cluster
  group: crd.projectcalico.org
  version: v1
  names:
    kind: GlobalNetworkPolicy
    plural: globalnetworkpolicies
    singular: globalnetworkpolicy
---
```

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 name: globalnetworksets.crd.projectcalico.org
spec:
 scope: Cluster
 group: crd.projectcalico.org
 version: v1
 names:
  kind: GlobalNetworkSet
  plural: globalnetworksets
  singular: globalnetworkset
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 name: networkpolicies.crd.projectcalico.org
spec:
 scope: Namespaced
 group: crd.projectcalico.org
 version: v1
 names:
  kind: NetworkPolicy
  plural: networkpolicies
  singular: networkpolicy
EOF
```

2. 执行以下命令编辑ConfigMap *eni-config*，添加networkpolicy配置。

```
kubectl -n kube-system edit cm eni-config
```

*eni-config*的示例如下所示：

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: eni-config
 namespace: kube-system
data:
 eni_conf: |
  {
    ...
    ...
  }
 10-terway.conf: |
  {
    ...
    ...
  }
 disable_network_policy: "false"
```

25.4. 容器镜像

25.4.1. 通过自建Kubernetes集群的免密组件拉取容器镜像

通过aliyun-acr-credential-helper组件可以在自建Kubernetes集群中免密拉取ACR个人版或企业版私有镜像。本文列举两个场景介绍如何使用免密组件拉取私有镜像。

前提条件

- 您需要创建一个注册集群，并将自建Kubernetes集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
- 通过kubectl连接注册集群。具体操作，请参见通过kubectl工具连接集群。

使用限制

使用免密组件涉及的镜像及集群限制如下：

- 镜像
  - 支持拉取容器镜像服务企业版实例和个人版实例中的私有镜像。
  - 支持拉取集群当前用户容器镜像服务中的私有镜像，通过跨账号授权或AccessKey ID和AccessKey Secret配置可以拉取其他用户的私有镜像。
  - 支持跨地域拉取容器镜像服务中的私有镜像。
- 集群
  - 支持集群多命名空间免密拉取。
  - 集群版本不低于1.11.2。否则，请您手动升级。具体操作，请参见升级ACK集群K8s版本。

注意事项

- 在Kubernetes资源（例如无状态应用Deployment）模板中配置拉取凭证（imagePullSecret）会导致免密组件失效，如果需使用免密组件，请避免手工配置拉取凭证（imagePullSecret）。
- 如果部署的Kubernetes资源（例如无状态应用Deployment）使用了自定义的ServiceAccount，需先调整免密组件配置文件中Service-Account字段，使其作用于自定义的ServiceAccount，再进行部署资源操作。
- 在集群中创建新的Service Account一段时间后，免密插件根据ACK集群默认权限生成的ACR私有镜像拉取Token才会更新到应用使用到的Service Account中， 使用Service Account的应用才会使用Token去拉取镜像。如果创建完Service Account之后立即创建应用则会出现因鉴权失败无法拉取的情况。
- 免密插件默认覆盖ACK中所有命名空间中默认的ServiceAccount中的imagePullSecret字段。被覆盖的ServiceAccount会随着对应kube-system命名空间中**acr-configuration**配置项中的service-account字段变动而变动。
- 在修改kube-system命名空间中的**acr-configuration**配置项时，请确认缩进是否与给出的场景的例子相同。建议直接复制对应场景的YAML内容到编辑器中，修改对应的值然后直接应用到集群，以保证YAML格式的正确性。

在注册集群中配置免密组件

步骤一：为免密组件配置RAM权限

在注册集群中安装组件前，您需要在接入集群中设置AccessKey用来访问云服务的权限。设置AccessKey前，您需要创建RAM用户并为其添加访问相关云资源的权限。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 创建权限策略。具体操作，请参见创建自定义策略。

   权限策略模板如下所示：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "cr:GetAuthorizationToken",
        "cr:ListInstanceEndpoint",
        "cr:PullRepository"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。
4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。
5. 使用AccessKey在注册集群中创建名为alibaba-addon-secret的Secret资源。

   安装免密组件时将自动引用此AccessKey访问对应的云服务资源。

   ```
   kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=<your access key id>' --from-literal='access-key-secret=<your access key secret>'
   ```

   > **说明** <your access key id> 及 <your access key secret> 为上一步获取的AccessKey信息。

步骤二：升级并配置免密组件

在使用免密组件拉取镜像前，您可能需要升级并配置组件。

1. 升级aliyun-acr-credential-helper组件。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面，单击目标集群操作列下的**更多 > 系统组件管理**。
   iv. 在**安全**区域，找到**aliyun-acr-credential-helper**，单击**升级**。
2. 设置acr-configuration配置项。

   方式一：通过控制台方式设置acr-configuration配置项。

   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面，单击目标集群下的**详情**。
   iv. 在**集群信息**页面左侧导航栏，选择**配置管理 > 配置项**。

v. 在**配置项**页面的上方命名空间下拉框中，选择**kube-system**，然后找到配置项**acr-configuration**，然后通过以下两种方式配置acr-configuration。

- 方式一：单击其右侧的**编辑**，设置**配置项**键和值。
  如果您没有**acr-configuration**配置项，请参见创建配置项。关于如何更新配置项，请参见修改配置项。
- 方式二：单击其右侧的**YAML编辑**，设置**配置项**键和值。

acr-configuration配置项的键和值说明如下。

配置项键	配置项键说明	配置项值
service-account	使免密组件作用于指定的服务账号。	默认为**default**。 **说明** 如果要配置多个请以逗号分隔，如果设置为“*”，表示支持所有命名空间下的所有ServiceAccount。

acr-registry-info	容器镜像的实例信息数组，YAML多行字符串格式，每个实例以三元组方式配置。
	说明 实例信息三元组：
	■ instanceId：实例ID，企业版实例必须配置此项。
	■ regionId：可选，默认为本地地域。
	■ domains：可选，默认为相应实例的所有域名。若要指定个别域名，多个以逗号分隔。
		默认值为空，表示免密拉取本地地域的默认容器镜像实例仓库。
		针对企业版容器镜像实例，配置示例如下：
		`- instanceId: "cri-xxx"`
		`regionId: "cn-hangzhou"`
		`domains: "xxx.com,yyy.com"`
		针对个人版容器镜像实例，配置示例如下：
		`- instanceId: ""`
		`regionId: "cn-hangzhou"`
		`domains: "xxx.com,yyy.com"`

watch-namespace	期望能免密拉取镜像的Namespace。	默认值为**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过期阈值。	默认值为**15m**（建议使用15 min）。

方式二：通过kubectl命令行的方式设置acr-configuration配置项。

i. 执行以下命令打开acr-configuration配置项的编辑页。

```
kubectl edit cm acr-configuration -n kube-system
```

ii. 根据实际情况设置acr-configuration配置项的值。

企业版和个人版容器镜像实例下设置acr-configuration配置项的示例如下：

- 企业版

```
apiVersion: v1
data:
 acr-api-version: "2018-12-01"
 acr-registry-info: |-
  - instanceId: "cri-xxx"
    regionId: "cn-hangzhou"
 expiring-threshold: 15m
 service-account: default
 watch-namespace: all
kind: ConfigMap
metadata:
 name: acr-configuration
 namespace: kube-system
 selfLink: /api/v1/namespaces/kube-system/configmaps/acr-configuration
```

- 个人版

```
apiVersion: v1
data:
 acr-api-version: "2018-12-01"
 acr-registry-info: |-
  - instanceId: ""
    regionId: "cn-hangzhou"
 expiring-threshold: 15m
 service-account: default
 watch-namespace: all
kind: ConfigMap
metadata:
 name: acr-configuration
 namespace: kube-system
 selfLink: /api/v1/namespaces/kube-system/configmaps/acr-configuration
```

场景一：拉取个人版实例和企业实例的私有镜像

注册集群支持同时拉取企业版和个人版的私有镜像，只拉取企业版的私有镜像，只拉取个人版的私有镜像。根据您的使用场景，按照以下方式修改配置项**acr-configuration**中的configMap。具体操作，请参见配置组件。配置内容如下：

- 配置拉取企业版的私有镜像。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: "cri-xxx"
     regionId: "cn-hangzhou"
     domains: "xxx.com,yyy.com"
```

- 配置拉取个人版的私有镜像。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
     regionId: "cn-hangzhou"
     domains: "xxx.com,yyy.com"
```

- 配置同时拉取个人版和企业版的私有镜像。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
   - instanceId: "cri-xxxx"
```

场景二：配置跨地域拉取镜像权限

如果需要拉取的镜像与当前注册集群不属于同一地域的时候，需要修改配置项acr-configuration中的configMap。

例如，个人版的镜像仓库同时拉取北京地域与杭州地域的镜像，配置如下。具体操作，请参见配置组件。

```
data:
  service-account: "default"
  watch-namespace: "all"
  expiring-threshold: "15m"
  notify-email: "cs@aliyuncs.com"
  acr-registry-info: |
   - instanceId: ""
     regionId: cn-beijing
   - instanceId: ""
     regionId: cn-hangzhou
```

25.4.2. 通过image-syncer工具迁移容器镜像

在Kubernetes集群迁移场景中，镜像仓库之间进行镜像迁移和同步是基本需求，image-syncer工具可以解决通用的容器镜像批量迁移和镜像同步复制的问题，将已有的容器镜像平滑的迁移到阿里云镜像服务ACR上。本文主要为您介绍如何通过image-syncer工具迁移容器镜像。

背景信息

阿里云上的容器服务ACK在使用成本、运维成本、方便性、长期稳定性上大大超过云厂商自建自维护Kubernetes集群，有不少云厂商纷纷想把之前自己维护Kubernetes负载迁移到阿里云ACK服务上。在迁移过程中，当镜像个数较少时，可以通过docker pull或docker push命令完成镜像迁移，如果涉及到成千上百个镜像，甚至几T的镜像仓库数据时，迁移过程就变得非常漫长，并且可能丢失数据。此时，用户在各种容器镜像仓库之间迁移时，期望有镜像同步复制的能力。阿里云开源image-syncer工具具备此能力并已经帮助多家云厂商成功迁移镜像，其中最大镜像仓库的总量达到3TB以上。同步任务时能跑满机器带宽，并且对进行同步任务的机器磁盘容量没有要求。

image-syncer简介

在Kubernetes集群迁移场景中，镜像仓库之间进行镜像迁移和镜像同步复制是基本需求，而使用docker pull或docker push结合脚本的传统方式进行镜像同步，有如下几个局限性：

- 依赖磁盘存储，需要及时进行本地镜像的清理，并且落盘造成多余的时间开销，难以胜任生产场景中大量镜像的迁移。
- 依赖Docker程序，Docker Daemon对Pull和Push的并发数进行了严格的限制，无法进行高并发同步。
- 一些功能只能通过HTTP API进行操作，仅使用Docker CLI无法做到，使脚本变得复杂。

image-syncer的定位是一个简单、易用的批量镜像迁移和镜像同步复制工具，支持几乎所有目前主流的基于Docker Registry V2搭建的镜像存储服务，例如ACR、Docker、Hub、Quay、自建Harbor等，目前已经初步经过了TB级别的生产环境镜像迁移验证，详情请参见image-syncer。

工具特点

image-syncer提供了以下几项支持：

- 支持多对多镜像仓库同步。
- 支持基于Docker Registry V2搭建的Docker镜像仓库服务
  例如，Docker Hub、Quay、 阿里云镜像服务ACR、Harbor等。
- 镜像同步复制只经过内存和网络，不依赖磁盘存储，同步速度快。
- 支持增量同步。
  通过对同步过的镜像blob信息落盘，不会对已同步的镜像进行重复同步。
- 支持并发同步。
  可以通过配置文件调整并发数。
- 支持自动重试失败的同步任务，解决大部分镜像同步中的网络抖动问题。
- 不依赖Docker以及其他程序。

通过使用image-syncer，只需要保证image-syncer的运行环境与需要同步的registry网络连通，您可以快速地完成从镜像仓库的迁移、复制以及增量同步，并且image-syncer对硬件资源几乎没有要求（因为image-syncer严格控制网络连接数目=并发数，所以只有在单个镜像层过大的情况下，并发数目过大可能会打满内存，内存占用 <= 并发数x最大镜像层大小）。除了使用重传机制规避同步过程中可能出现的偶发问题之外，image-syncer会在运行结束时统计最后同步失败的镜像个数，并且打印出详细的日志，帮助使用者定位同步过程中出现的问题。

准备工作

使用image-syncer时，您只需要提供一个配置文件，内容示例如下：

```
{
  "auth": {            // 认证字段，其中每个对象为一个registry的一个账号和
                  // 密码；通常，同步源需要具有pull以及访问tags权限，
                  // 同步目标需要拥有push以及创建仓库权限，如果没有提供，则默认匿名访问
    "quay.io": {       // registry的url，需要和下面images中对应registry的url相同
      "username": "xxx",          // 用户名，可选
      "password": "xxxxxxxxx",     // 密码，可选
      "insecure": true          // registry是否是http服务，如果是，insecure字段需要为true，默认是false，可选，支持这个选项需要image-syncer版本 > v1.0.1
    },
    "registry.cn-beijing.aliyuncs.com": {
      "username": "xxx",
      "password": "xxxxxxxxx"
    },
    "registry.hub.docker.com": {
      "username": "xxx",
      "password": "xxxxxxxxxx"
    }
  },
  "images": {
    // 同步镜像规则字段，其中一条规则包括一个源仓库（键）和一个目标仓库（值）
    // 同步的最大单位是仓库（repo），不支持通过一条规则同步整个namespace以及registry
    // 源仓库和目标仓库的格式与docker pull/push命令使用的镜像url类似（registry/namespace/repository:tag）
    // 源仓库和目标仓库（如果目标仓库不为空字符串）都至少包含registry/namespace/repository
    // 源仓库字段不能为空，如果需要将一个源仓库同步到多个目标仓库需要配置多条规则
    // 目标仓库名可以和源仓库名不同（tag也可以不同），此时同步功能类似于：docker pull + docker tag + docker push
    "quay.io/coreos/kube-rbac-proxy": "quay.io/ruohe/kube-rbac-proxy",
    "xxxx":"xxxxx",
    "xxx/xxx/xx:tag1,tag2,tag3":"xxx/xxx/xx"
    // 当源仓库字段中不包含tag时，表示将该仓库所有tag同步到目标仓库，此时目标仓库不能包含tag
    // 当源仓库字段中包含tag时，表示只同步源仓库中的一个tag到目标仓库，如果目标仓库中不包含tag，则默认使用源tag
    // 源仓库字段中的tag可以同时包含多个（比如"a/b/c:1,2,3"），tag之间通过","隔开，此时目标仓库不能包含tag，并且默认使用原来的tag
    // 当目标仓库为空字符串时，会将源镜像同步到默认registry的默认namespace下，并且repo以及tag与源仓库相同，默认registry和默认namespace可以通过命令行参数以及环境变量配置，参考下面的描述
  }
}
```

使用示例

- 从自建Harbor同步镜像到ACR默认实例版
- 从自建Harbor同步镜像到ACR企业版

25.5. 弹性伸缩

25.5.1. 创建节点池并扩容

注册集群弹性节点池用于管理一组线上节点资源，您可以通过节点池为您的线下集群扩容线上ECS节点。本文主要介绍如何创建注册集群弹性节点池。

前提条件

已完成创建混合集群中步骤6之前的操作。

步骤一：创建节点池

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面右上角，单击**创建节点池**。
6. 在**创建节点池**对话框，设置创建节点池的配置项。

   关于部分配置项说明如下所示。

参数	说明
数量	设置节点池初始节点数量。如不需要创建节点，可以填写为0。
操作系统	选择节点的操作系统，仅支持CentOS及Alibaba Cloud Linux 2.1903。
节点标签	您可以为集群节点添加标签。
ECS标签	您可以为ECS实例添加标签。
污点	您可以为集群节点添加污点。
安全组	选择节点所在安全组。

   关于更多配置项说明，请参见节点池配置。

7. 单击**确认配置**。

   在**节点池**页面，如果节点池**状态**显示为**初始化中**，则说明节点池正在创建中。创建完成后，节点池**状态**显示为**已激活**。



步骤二：扩容云上节点

在您扩容ACK注册集群前，请注意以下两点。

- 默认情况下，每个集群中最多可包含100个节点。如果您需要添加更多节点，请提交工单申请。
- 添加已有云服务器时，请确保您的专有网络中的ECS实例已绑定一个弹性公网IP（EIP），或者相应VPC已经配置了NAT网关。您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。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在目标节点池右侧的**操作**列中，单击**扩容**。
6. 在节点池对话框的**扩容数量**调整框中，输入要增加的节点数，然后单击**提交**。

   > **说明** 如果您需要修改节点池配置，展开**修改节点池配置**。

   在**节点池**页面，节点池**状态**刚开始显示为**伸缩中**。扩容完成后，**状态**会显示为**已激活**。

相关操作

1. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。

2. 在节点池列表中，单击目标节点池右侧的**详情**。
3. 在节点池详情页面下方的**节点管理**页签中，查看节点池中扩容节点的详细信息。

25.5.2. 配置自动弹性伸缩

本文将为您介绍如何使用混合集群的自动弹性伸缩能力。

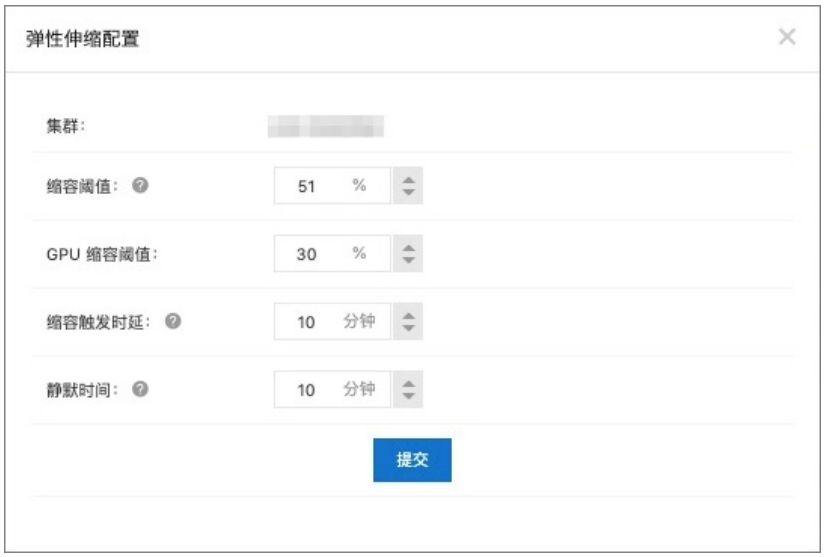
前提条件

创建自动弹性伸缩配置时，cluster-autoscaler组件将以Deployment形式自动部署在混合集群中。由于云上节点会在自动缩容后销毁，不利于Deployment类型部署的系统组件提供稳定的服务，所以这种提供云上服务的组件需要避免被调度到自动扩容出来的云上节点，同时也要避免被调度到云下节点。所以需要满足以下条件：

- 创建和扩容普通节点池。具体操作，请参见创建节点池并扩容。
- 为节点池中的节点配置节点标签 `alibabacloud.com/cloud-worker-nodes=true` 。具体操作，请参见添加节点标签。
  cluster-autoscaler组件将会自动调度到拥有节点标签的云上节点上。

步骤一：创建自动弹性伸缩配置

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，执行以下操作进入**自动弹性伸缩配置**页面。
   您可以通过以下两种路径配置自动弹性伸缩。
   - 路径一：在目标集群右侧的**操作**列下，选择**更多 > 自动伸缩**。
   - 路径二：
     a. 选择目标集群，然后单击**操作**列的**详情**。
     b. 在目标集群的左侧导航栏，选择**节点管理 > 节点池**。
     c. 在**节点池**页面右上方，单击**自动弹性伸缩配置**。
4. 在**自动弹性伸缩配置**页面，完成弹性伸缩配置并提交。



5. 配置cluster-autoscaler组件。
   在成功完成自动弹性伸缩配置后，集群中就会自动部署一个Deployment。

i. 执行以下命令查看Deployment。

```
kubectl -n kube-system get deploy |grep cluster-autoscaler
```

预期输出：

```
cluster-autoscaler      1/1   1      1      5s
```

ii. 配置cluster-autoscaler组件相关云资源的RAM Policy，代码示例如下所示。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ess:DescribeScalingGroups",
        "ess:DescribeScalingInstances",
        "ess:DescribeScalingActivities",
        "ess:DescribeScalingConfigurations",
        "ess:DescribeScalingRules",
        "ess:DescribeScheduledTasks",
        "ess:DescribeLifecycleHooks",
        "ess:DescribeNotificationConfigurations",
        "ess:DescribeNotificationTypes",
        "ess:DescribeRegions",
        "ess:CreateScalingRule",
        "ess:ModifyScalingGroup",
        "ess:RemoveInstances",
        "ess:ExecuteScalingRule",
        "ess:ModifyScalingRule",
        "ess:DeleteScalingRule",
        "ecs:DescribeInstanceTypes",
        "ess:DetachInstances",
        "vpc:DescribeVSwitches"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

iii. 执行以下命令授权RAM Policy的AccessKey信息。

```
export ACCESS_KEY_ID=<ACCESS KEY ID>
export ACCESS_KEY_SECRET=<ACCESS KEY SECRET>
```

iv. 执行以下命令创建一个名为*alibaba-addon-secret*的Secret资源。

```
kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=${ACCESS_KEY_ID}' --from-literal='access-key-secret=${ACCESS_KEY_SECRET}'
```

步骤二：创建自动弹性伸缩节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，执行以下操作进入**自动弹性伸缩配置**页面。

您可以通过以下两种路径配置自动弹性伸缩。

- 路径一：在目标集群右侧的**操作**列下，选择**更多 > 自动伸缩**。
- 路径二：
  a. 选择目标集群，然后单击**操作**列的**详情**。
  b. 在目标集群的左侧导航栏，选择**节点管理 > 节点池**。
  c. 在**节点池**页面右上方，单击**自动弹性伸缩配置**。

4. 在**自动弹性伸缩配置**页面，单击**创建节点池**。
5. 在**创建节点池**对话框，设置创建节点池的配置项。

   部分配置项说明如下所示。

参数	说明
数量	设置节点池初始节点数量。如不需要创建节点，可以填写为0。
操作系统	选择节点的操作系统，包括CentOS、Alibaba Cloud Linux 2.1903。
节点标签	您可以为集群节点添加标签。例如，`workload=auto`。
ECS标签	您可以为ECS实例添加标签。
污点	您可以为集群节点添加污点。
安全组	选择节点所在安全组。

   关于更多配置项说明，请参见创建Kubernetes专有版集群。

6. 单击**确认配置**。

执行结果

执行以下命令根据设置的**节点标签** `workload=auto` 验证节点池是否可以正确弹出节点。

```
kubectl run nginx --image nginx -l workload=auto
```

25.5.3. 使用自定义节点添加脚本

混合集群的添加云上节点的方式依赖本地数据中心自建Kubernetes集群的搭建方式，例如使用Kubeadm搭建、使用Kubernetes二进制文件搭建或者使用Rancher平台搭建等。本文将介绍如何使用自定义节点添加脚本。

前提条件

已完成创建混合集群中步骤5之前的操作。

步骤一：编写自定义节点添加脚本

kubelet配置的示例代码如下所示。

```
cat >/usr/lib/systemd/system/kubelet.service <<EOF
[Unit]
Description=Kubernetes Kubelet
After=docker.service
Requires=docker.service
[Service]
ExecStart=/data0/kubernetes/bin/kubelet \\
 --node-ip=${ALIBABA_CLOUD_NODE_NAME} \\
 --hostname-override=${ALIBABA_CLOUD_NODE_NAME} \\
 --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf \\
 --config=/var/lib/kubelet/config.yaml \\
 --kubeconfig=/etc/kubernetes/kubelet.conf \\
 --cert-dir=/etc/kubernetes/pki/ \\
 --cni-bin-dir=/opt/cni/bin \\
 --cni-cache-dir=/opt/cni/cache \\
 --cni-conf-dir=/etc/cni/net.d \\
 --logtostderr=false \\
 --log-dir=/var/log/kubernetes/logs \\
 --log-file=/var/log/kubernetes/logs/kubelet.log \\
 --node-labels=${ALIBABA_CLOUD_LABELS} \\
 --root-dir=/var/lib/kubelet \\
 --provider-id=${ALIBABA_CLOUD_PROVIDE_ID} \\
 --register-with-taints=${ALIBABA_CLOUD_TAINTS} \\
 --v=4
Restart=on-failure
RestartSec=5
[Install]
WantedBy=multi-user.target
EOF
```

在编写自定义节点添加脚本时需要注意必须接收阿里云注册集群下发的系统环境变量，如下所示。

系统环境变量	说明	示例
ALIBABA_CLOUD_PROVIDE_ID	自定义节点添加脚本中必须接收并进行配置，否则会影响注册集群管理异常。	`ALIBABA_CLOUD_PROVIDE_ID=cn-shenzhen.i-wz92ewt14n9wx9mol2cd`
ALIBABA_CLOUD_NODE_NAME	自定义节点添加脚本中必须接收并进行配置，否则会导致注册集群节点池中节点状态异常。	`ALIBABA_CLOUD_NODE_NAME=cn-shenzhen.192.168.1.113`
ALIBABA_CLOUD_LABELS	自定义节点添加脚本中必须接收并进行配置，否则会影响节点池管理状态异常以及后续工作负载在云上云下的调度。	`ALIBABA_CLOUD_LABELS=alibabacloud.com/nodepool-id=np0e2031e952c4492bab32f512ce1422f6,ack.aliyun.com=cc3df6d939b0d4463b493b82d0d670c66,alibabacloud.com/instance-id=i-wz960ockeekr3dok06kr,alibabacloud.com/external=true,workload=cpu` 其中workload=cpu是用户在节点池中配置的用户自定义节点标签，其他都为系统下发的节点标签。

系统环境变量	说明	示例
ALIBABA_CLOUD_TAINTS	自定义节点添加脚本中必须接收并进行配置，否则节点池中的污点配置将不会生效。	`ALIBABA_CLOUD_TAINTS=workload=ack:NoSchedule`

步骤二：保存自定义脚本

将自定义脚本保存在HTTP文件服务器上，例如存放在OSS Bucket上。示例地址为 `https://kubelet-****.oss-ap-southeast-3-internal.aliyuncs.com/attachnode.sh` 。

步骤三：使用自定义脚本

1. 将本地数据中心集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

   注册集群的Agent组件会自动在kube-system下创建名为ack-agent-config的ConfigMap，初始化配置如下所示。

   ```
   apiVersion: v1
   data:
    addNodeScriptPath: ""
    enableNodepool: "true"
    isInit: "true"
   kind: ConfigMap
   metadata:
    name: ack-agent-config
    namespace: kube-system
   ```

2. 将自定义节点添加脚本的路径 `https://kubelet-****.oss-ap-southeast-3-internal.aliyuncs.com/attachnode.sh` 配置到 `addNodeScriptPath` 字段区域并保存。

   命令示例如下所示。

   ```
   apiVersion: v1
   data:
    addNodeScriptPath: https://kubelet-****.oss-ap-southeast-3-internal.aliyuncs.com/attachnode.sh
    enableNodepool: "true"
    isInit: "true"
   kind: ConfigMap
   metadata:
    name: ack-agent-config
    namespace: kube-system
   ```

25.5.4. 扩容弹性容器实例ECI

虚拟节点（Virtual Node）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。您可以灵活动态的按需创建ECI Pod，免去集群容量规划的麻烦。本文主要介绍虚拟节点和弹性容器实例ECI，以及通过ack-virtual-node组件如何创建ECI Pod。

前提条件

- 已创建注册集群，并将自建Kubernetes集群（高于1.14版本）接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
- 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。
- 您需要确认集群所在区域在ECI支持的地域列表内。登录弹性容器实例控制台查看已经支持的地域和可用区。
- 自建Kubernetes集群网络与注册集群所在的VPC网络内网互通，即使用内网接入端点接入注册集群。更多信息，请参见方案介绍。

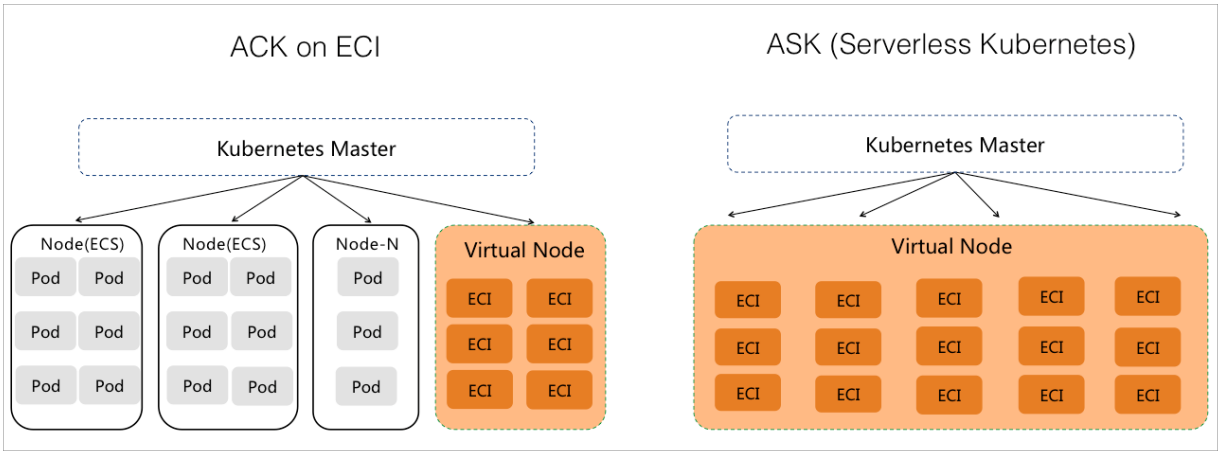
虚拟节点和弹性容器实例ECI

阿里云弹性容器实例ECI（Elastic Container Instance）是面向容器的无服务器弹性计算服务，提供免运维、强隔离、快速启动的容器运行环境。使用ECI无需购买和管理底层ECS服务器，让您更加关注在容器应用而非底层基础设施的维护工作。您可按需创建ECI，仅为容器配置的资源付费（按量按秒计费）。

虚拟节点Virtual Node非常适合运行在如下多个场景，帮助用户极大降低计算成本，提升计算弹性效率：

- 在线业务的波峰波谷弹性伸缩：如在线教育、电商等行业有着明显的波峰波谷计算特征。使用虚拟节点可以显著减少固定资源池的维护，降低计算成本。
- 数据计算：使用虚拟节点承载Spark、Presto等计算场景，有效降低计算成本。
- CI/CD Pipeline：Jenkins、Gitlab-Runner。
- Job任务：定时任务、AI。

阿里云容器服务基于虚拟节点和ECI提供了多种Serverless Container产品形态，包括Serverless Kubernetes（ASK）和ACK on ECI，充分支撑各种弹性和免节点运维场景的用户诉求。



使用说明

注册集群支持托管的弹性容器实例ECI和非托管弹性容器实例ECI两种方式：

- “托管”是指负责管理ECI Pods生命周期的组件将托管在管控侧，可以降低组件维护成本、节省集群资源。
- “非托管”是指负责管理ECI Pods生命周期的组件将直接部署在集群。

> **注意** 对于托管的弹性容器实例ECI，不需要进行步骤一的操作，直接执行步骤二：在注册集群中部署ack-virtual-node组件即可。

步骤一：在注册集群中配置ack-virtual-node组件的RAM权限

若您在注册集群的**集群导入代理配置**区域使用私网接入IDC集群，则可以跳过此步骤，ack-virtual-node组件将托管到ACK集群。更多信息，请参见将目标集群接入注册集群中。

在注册集群安装组件前，您需要在接入集群中设置AccessKey用来访问云服务的权限。设置AccessKey前，您需要创建RAM用户并为其添加访问相关云资源的权限。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. （可选）创建自定义权限策略。

   具体操作，请参见创建自定义策略。
   自定义策略模板如下所示：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "eci:CreateContainerGroup",
        "eci:DeleteContainerGroup",
        "eci:DescribeContainerGroups",
        "eci:DescribeContainerLog",
        "eci:UpdateContainerGroup",
        "eci:UpdateContainerGroupByTemplate",
        "eci:CreateContainerGroupFromTemplate",
        "eci:RestartContainerGroup",
        "eci:ExportContainerGroupTemplate",
        "eci:DescribeContainerGroupMetric",
        "eci:DescribeMultiContainerGroupMetric",
        "eci:ExecContainerCommand",
        "eci:CreateImageCache",
        "eci:DescribeImageCaches",
        "eci:DeleteImageCache"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。

   可选择自定义权限策略或AliyunECIFullAccess策略为RAM用户授权。

4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。

5. 使用AccessKey在注册集群中创建名为*alibaba-addon-secret*的Secret资源。

   安装ack-virtual-node组件时将自动引用此AccessKey访问对应的云服务资源。

   ```
   kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=<your access key id>' --from-literal='access-key-secret=<your access key secret>'
   ```

   > **说明** <your access key id> 及 <your access key secret> 为上一步获取的AccessKey信息。

步骤二：在注册集群中部署ack-virtual-node组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 组件管理**。
5. 单击**其他**页签，在ack-virtual-node组件区域单击**安装**。

   在**提示**对话框单击**确定**。这时会以集群默认虚拟交换机和安全组作为ack-virtual-node的初始ECI配置参数。具体操作，请参见相关操作。

步骤三：创建ECI Pod

创建ECI Pod前，需要确保Virtual node的状态为ready。

执行以下命令，查看Virtual node状态。

```
kubectl  get no |grep virtual-kubelet
```

预期输出：

```
virtual-kubelet-cn-hangzhou-b  Ready  agent         18d  v1.20.11-aliyun.1
```

从预期输出可得，当前Virtual node状态为ready。

您可以通过以下两种方法创建ECI Pod。

- **方式一：配置Pod标签。**
  将Pod添加标签 `alibabacloud.com/eci=true` ，Pod将以ECI方式运行，其节点是虚拟节点，示例如下：
  i. 执行以下命令给Pod添加标签。
  ```
  kubectl run nginx --image nginx -l alibabacloud.com/eci=true
  ```
  ii. 执行以下命令查看Pod。
  ```
  kubectl get pod -o wide|grep virtual-kubelet
  ```
  预期输出：
  ```
  nginx-7fc9f746b6-r4xgx  0/1  ContainerCreating  0  20s  192.168.XX.XX  virtual-kubelet  <none>  <none>
  ```
- **方式二：配置命名空间标签。**
  将Pod所在的命名空间添加标签 `alibabacloud.com/eci=true` ，Pod将以ECI方式运行，其节点是虚拟节点，示例如下：
  i. 执行以下命令创建虚拟节点。
  ```
  kubectl create ns vk
  ```
  ii. 执行以下命令将Pod所在的命名空间添加标签。
  ```
  kubectl label namespace vk alibabacloud.com/eci=true
  ```
  iii. 执行以下命令将命名空间中的Pod调度到虚拟节点上。
  ```
  kubectl -n vk run nginx --image nginx
  ```
  iv. 执行以下命令查看Pod。
  ```
  kubectl -n vk get pod -o wide|grep virtual-kubelet
  ```
  预期输出：
  ```
  nginx-6f489b847d-vgj4d  1/1  Running  0  1m  192.168.XX.XX  virtual-kubelet  <none>  <none>
  ```

相关操作

修改虚拟节点配置

虚拟节点Controller的配置决定了其调度ECI Pod的行为和ECI运行环境配置，包括虚拟交换机和安全组配置等。您可以根据需要灵活的修改Controller配置，修改配置后不会影响已经运行的ECI Pod，会立即生效于新建的ECI Pod。

执行以下命令修改虚拟节点Controller的配置。

```
kubectl -n kube-system edit deployment ack-virtual-node-controller
```

常用的变更操作如下：

- **更新virtual-node controller版本**
  当您使用更新虚拟节点功能时，需要更新Virtual Node Controller镜像至最新版本。
- **修改安全组配置ECI_SECURITY_GROUP**
  您可以修改此环境变量，改变ECI Pod的安全组。
- **修改虚拟交换机配置ECI_VSWITCH**
  您可以修改此环境变量，改变ECI Pod所在的虚拟交换机。建议配置多个虚拟交换机支持多可用区，当单可用区库存不足时，Controller会选择另外一个可用区创建ECI Pod。
- **修改kube-proxy配置ECI_KUBE_PROXY**
  此环境变量默认值为true，表示ECI Pod默认可以访问集群中的ClusterIP Service。如果ECI Pod无需访问ClusterIP Service时，例如Job计算场景，您可以设置此环境变量为**false**关闭kube-proxy功能。另外在一些规模化场景，例如集群中需要启动大量ECI Pod时，ECI中的kube-proxy和Kubernetes Api Server之间的并发连接数也会大量增加，您同样可以选择关闭kube-proxy功能，减少对API Server的压力提升可扩展性。
- **修改kube-system/eci-profile configmap**
  您可以修改此ConfigMap配置更多ECI相关参数，例如虚拟交换机及安全组等。

删除虚拟节点

1. 卸载ack-virtual-node组件。
   在注册集群中，删除所有ECI Pod后，在**组件管理**页面卸载ack-virtual-node组件即可。
2. 通过命令 `kubectl delete no` 删除相关虚拟节点。

> **说明** 当集群中存在ECI Pod时，卸载ack-virtual-node组件会导致ECI实例的残留。

相关文档

- 注册集群概述
- 通过部署ACK虚拟节点组件创建ECI Pod
- 基于ECI运行Job任务

25.6. 运维

25.6.1. 可观测性

25.6.1.1. 将日志服务接入注册集群

注册集群接入日志服务可以为分布在各处的Kubernetes集群提供统一的管理方式。本文介绍如何通过阿里云容器服务Kubernetes版中的应用将日志服务接入注册集群。

前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

步骤一：为日志服务组件配置RAM权限

在注册集群中安装组件前，您需要在接入集群中设置AccessKey用来访问云服务的权限。设置AccessKey前，您需要创建RAM用户并为其添加访问相关云资源的权限。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 创建权限策略。具体操作，请参见创建自定义策略。

   Logtail组件需要的权限策略信息如下：

```
{
  "Version": "1",
```

```
  "Statement": [
    {
      "Action": [
        "log:CreateProject",
        "log:GetProject",
        "log:DeleteProject",
        "log:CreateLogStore",
        "log:GetLogStore",
        "log:UpdateLogStore",
        "log:DeleteLogStore",
        "log:CreateConfig",
        "log:UpdateConfig",
        "log:GetConfig",
        "log:DeleteConfig",
        "log:CreateMachineGroup",
        "log:UpdateMachineGroup",
        "log:GetMachineGroup",
        "log:DeleteMachineGroup",
        "log:ApplyConfigToGroup",
        "log:GetAppliedMachineGroups",
        "log:GetAppliedConfigs",
        "log:RemoveConfigFromMachineGroup",
        "log:CreateIndex",
        "log:GetIndex",
        "log:UpdateIndex",
        "log:DeleteIndex",
        "log:CreateSavedSearch",
        "log:GetSavedSearch",
        "log:UpdateSavedSearch",
        "log:DeleteSavedSearch",
        "log:CreateDashboard",
        "log:GetDashboard",
        "log:UpdateDashboard",
        "log:DeleteDashboard",
        "log:CreateJob",
        "log:GetJob",
        "log:DeleteJob",
        "log:UpdateJob",
        "log:PostLogStoreLogs",
        "log:CreateSortedSubStore",
        "log:GetSortedSubStore",
        "log:ListSortedSubStore",
        "log:UpdateSortedSubStore",
        "log:DeleteSortedSubStore",
        "log:CreateApp",
        "log:UpdateApp",
        "log:GetApp",
        "log:DeleteApp",
        "cs:DescribeTemplates",
        "cs:DescribeTemplateAttribute"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
```

```
}
```

3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。
4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。
5. 使用AccessKey在注册集群中创建名为 alibaba-addon-secret 的Secret资源。

   执行以下命令创建Logtail组件使用的Secret。

```
kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=<your Access Key ID>' --from-literal='access-key-secret=<your AccessKey Secret>'
```

> **说明** <your AccessKey ID> 及 <your AccessKey Secret> 为上一步获取的AccessKey信息。

步骤二：安装logtail-ds组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 系统组件管理**。
4. 单击**日志与监控**页签，在**logtail-ds**组件区域单击**安装**。
5. 在**提示**对话框单击**确定**。

步骤三：配置日志服务

关于创建应用时配置日志服务的具体操作，请参见步骤二：创建应用时配置日志服务。

相关文档

- 注册集群概述
- Logtail发布历史

25.6.1.2. 将事件中心接入注册集群

注册集群接入事件中心功能可以记录Kubernetes集群的状态变更，包括配置Pod及组件异常等。Kubernetes事件中心功能实时汇聚Kubernetes中的所有事件并提供存储、查询、分析、可视化、告警等能力。本文介绍如何通过容器服务Kubernetes版中的应用将事件中心接入至注册的Kubernetes集群。

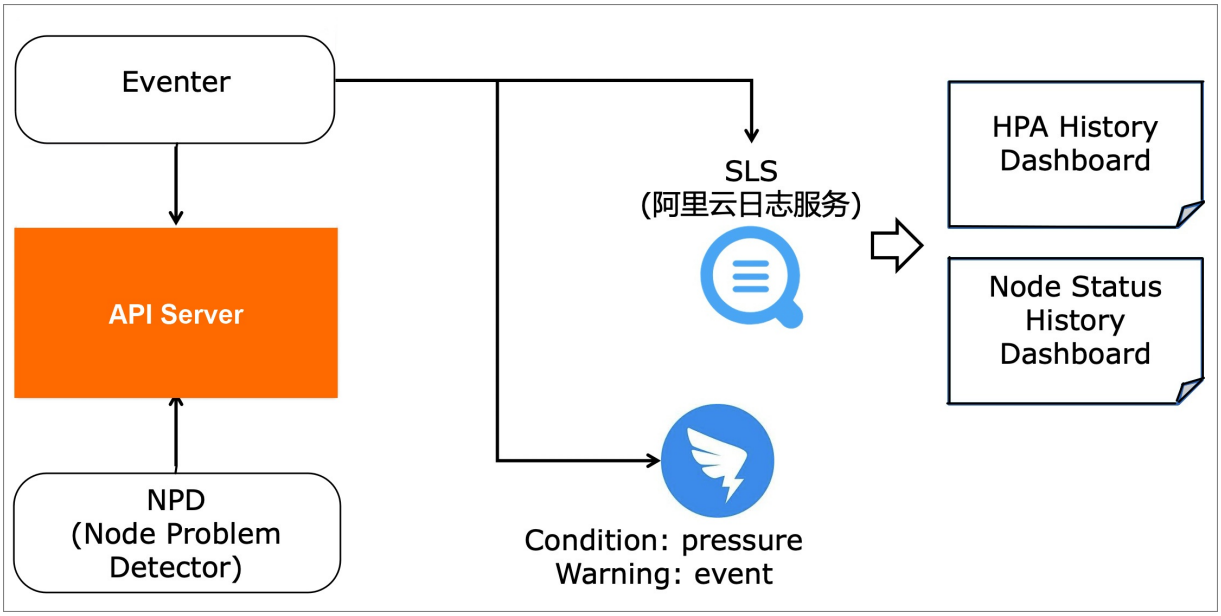
前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

背景信息

Kubernetes的架构设计基于状态机，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。

ACK提供开箱即用的容器场景事件监控方案，通过ACK维护的NPD（node-problem-detector）以及包含在NPD中的kube-eventer提供容器事件监控能力。



- NPD（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，结合kube-eventer可以实现节点事件告警的闭环。更多信息，请参见NPD。
- kube-eventer是ACK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、EventBridge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。更多信息，请参见kube-eventer。

步骤一：为事件中心组件配置RAM权限

在注册集群中安装组件前，您需要在接入集群中设置AccessKey用来访问云服务的权限。设置AccessKey前，您需要创建RAM用户并为其添加访问相关云资源的权限。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 创建权限策略。具体操作，请参见创建自定义策略。

   Logtail组件需要的权限策略信息如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:CreateProject",
        "log:GetProject",
        "log:DeleteProject",
        "log:CreateLogStore",
        "log:GetLogStore",
        "log:UpdateLogStore",
        "log:DeleteLogStore",
        "log:CreateConfig",
        "log:UpdateConfig",
        "log:GetConfig",
        "log:DeleteConfig",
        "log:CreateMachineGroup",
        "log:UpdateMachineGroup",
        "log:GetMachineGroup",
```

```
            "log:DeleteMachineGroup",
            "log:ApplyConfigToGroup",
            "log:GetAppliedMachineGroups",
            "log:GetAppliedConfigs",
            "log:RemoveConfigFromMachineGroup",
            "log:CreateIndex",
            "log:GetIndex",
            "log:UpdateIndex",
            "log:DeleteIndex",
            "log:CreateSavedSearch",
            "log:GetSavedSearch",
            "log:UpdateSavedSearch",
            "log:DeleteSavedSearch",
            "log:CreateDashboard",
            "log:GetDashboard",
            "log:UpdateDashboard",
            "log:DeleteDashboard",
            "log:CreateJob",
            "log:GetJob",
            "log:DeleteJob",
            "log:UpdateJob",
            "log:PostLogStoreLogs",
            "log:CreateSortedSubStore",
            "log:GetSortedSubStore",
            "log:ListSortedSubStore",
            "log:UpdateSortedSubStore",
            "log:DeleteSortedSubStore",
            "log:CreateApp",
            "log:UpdateApp",
            "log:GetApp",
            "log:DeleteApp",
            "cs:DescribeTemplates",
            "cs:DescribeTemplateAttribute"
        ],
        "Resource": [
            "*"
        ],
        "Effect": "Allow"
    }
  ]
}
```

3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。

   可选择自定义权限策略或AliyunECIFullAccess策略为RAM用户授权。

4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。

5. 使用AccessKey在注册集群中创建名为 alibaba-addon-secret 的Secret资源。

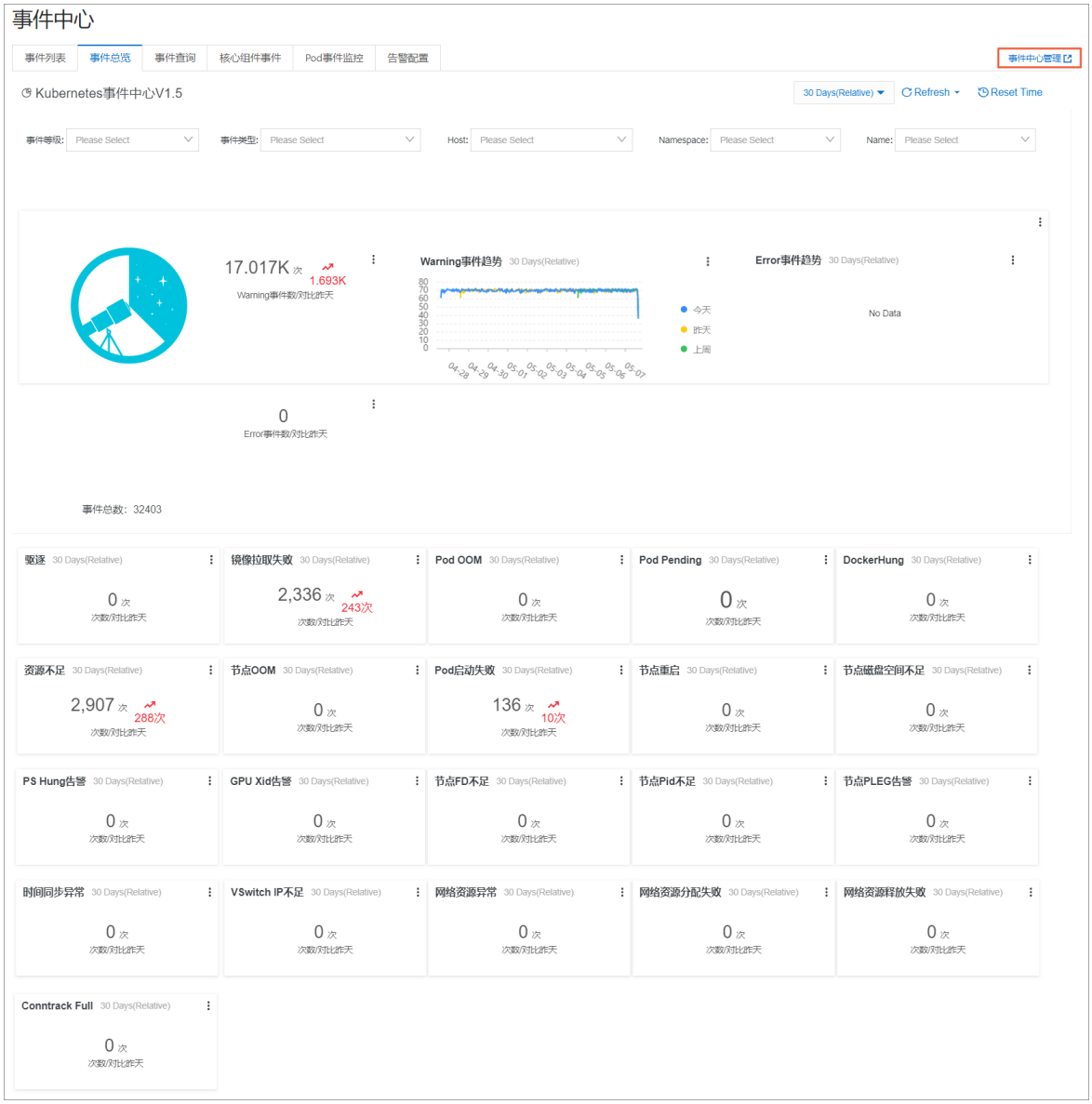
   安装事件中心相关组件时将自动引用此AccessKey访问对应的云服务资源。
   执行以下命令创建Logtail组件使用的Secret。

   ```
   kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=<your Access Key ID>' --from-literal='access-key-secret=<your AccessKey Secret>'
   ```

   > **说明** <your AccessKey ID> 及 <your AccessKey Secret> 为上一步获取的AccessKey信息。

步骤二：安装事件中心

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 选择**运维管理 > 事件中心**。
5. （可选）若提示**未安装组件ack-node-problem-detector**，请单击**创建事件中心**。
6. 单击右上角的**事件中心管理**，在**K8s事件中心**页面左侧导航栏，单击目标集群名称左边的展开图标 ，查看Kubernetes事件中心相关信息。
   您可以查看Kubernetes事件的总览及详情、Pod生命周期，还可以进行自定义查询及告警配置。



执行结果

配置成功后，即可使用事件中心功能。更多信息，请参见创建并使用Kubernetes事件中心。
将事件中心接入注册的Kubernetes集群后，即可使用K8s事件中心，包括查看事件总览、查询事件详情、查看Pod生命周期、配置告警和自定义查询等操作。

相关文档

- 注册集群概述

- 事件监控

25.6.1.3. 将报警配置功能接入注册集群

报警配置功能提供容器场景报警能力的统一管理，包括容器服务异常事件报警，以及容器场景指标报警。您可以通过集群内部署CRD的方式配置容器服务的报警规则。本文介绍在注册集群中如何接入报警配置功能及如何配置报警规则。

前提条件

- 已创建注册集群，并将自建Kubernetes集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
- 已通过kubectl连接注册集群。具体操作，请参见通过kubectl工具连接集群。

功能使用场景

容器服务报警配置功能集合容器场景的监控报警能力，提供报警的统一配置管理，有如下几个典型的使用场景：

- **集群运维**
  可以通过监控报警第一时间了解集群管控、存储、网络、弹性扩缩容等异常事件。例如：
  - 通过**集群资源异常报警规则集**第一时间感知集群基础资源的关键指标是否异常。例如，CPU、Memory、网络等关键指标是否出现高水位情况，避免影响集群稳定性。
  - 通过配置并查看**集群异常事件报警规则集**感知集群节点或容器节点异常。例如，集群节点Docker进程异常、集群节点进程异常及集群容器副本启动失败等异常。
  - 通过配置并查看**集群存储异常事件报警规则集**感知集群存储的变更与异常。
  - 通过配置并查看**集群网络异常事件报警规则集**感知集群网络的变更与异常。
  - 通过配置并查看**集群管控运维异常报警规则集**感知集群管控的变更与异常等。
- **应用开发**
  可以通过监控报警第一时间了解在集群中运行应用的异常事件、指标是否异常。例如，集群容器副本异常或者应用Deployment的CPU、内存水位指标是否超过阈值等。可通过开启报警配置功能中的默认报警规则模板，即可快速接受集群内应用容器副本的异常事件报警通知。例如，通过配置并订阅关注**集群容器副本异常报警规则集**感知所属应用的Pod是否异常。
- **应用管理**
  关注运行在集群上的应用健康、容量规划、集群运行稳定性及异常甚至是错误报警等贯穿应用生命周期的一系列问题。例如，通过配置并订阅关注**集群重要事件报警规则集**感知集群内所有Warning、Error等异常报警；关注**集群资源异常报警规则集**感知集群的资源情况，从而更好地做容量规划等。
- **多集群管理**
  当您有多个集群需要管理，为集群配置报警规则往往会是一个重复繁琐且难以同步的操作。容器服务报警配置功能，支持通过集群内部署**CRD配置的方式管理报警规则**。可通过维护多个集群中同样配置的CRD资源，来方便快捷地实现多集群中报警规则的同步配置。

在注册集群中配置云监控组件

步骤一：为云监控组件配置RAM权限

在注册集群中安装组件前，您需要在接入集群中设置AccessKey用来访问云服务的权限。设置AccessKey前，您需要创建RAM用户并为其添加访问相关云资源的权限。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 创建权限策略。具体操作，请参见创建自定义策略。

   权限策略模板如下所示：

```
{
      "Action": [
        "log:*",
        "arms:*",
        "cs:UpdateContactGroup"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
}
```

3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。
4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。
5. 使用AccessKey在注册集群中创建名为alibaba-addon-secret的Secret资源。

   安装云监控组件时将自动引用此AccessKey访问对应的云服务资源。

   ```
   kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=<your access key id>' --from-literal='access-key-secret=<your access key secret>'
   ```

   > **说明** <your access key id> 及 <your access key secret> 为上一步获取的AccessKey信息。

步骤二：安装与升级云监控组件

控制台会自动检测报警配置环境是否符合要求，并会引导进行开通、安装或升级组件。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。
5. 在**报警配置**页面控制台会自动检查以下条件。

   若不符合条件，请按以下提示完成操作。

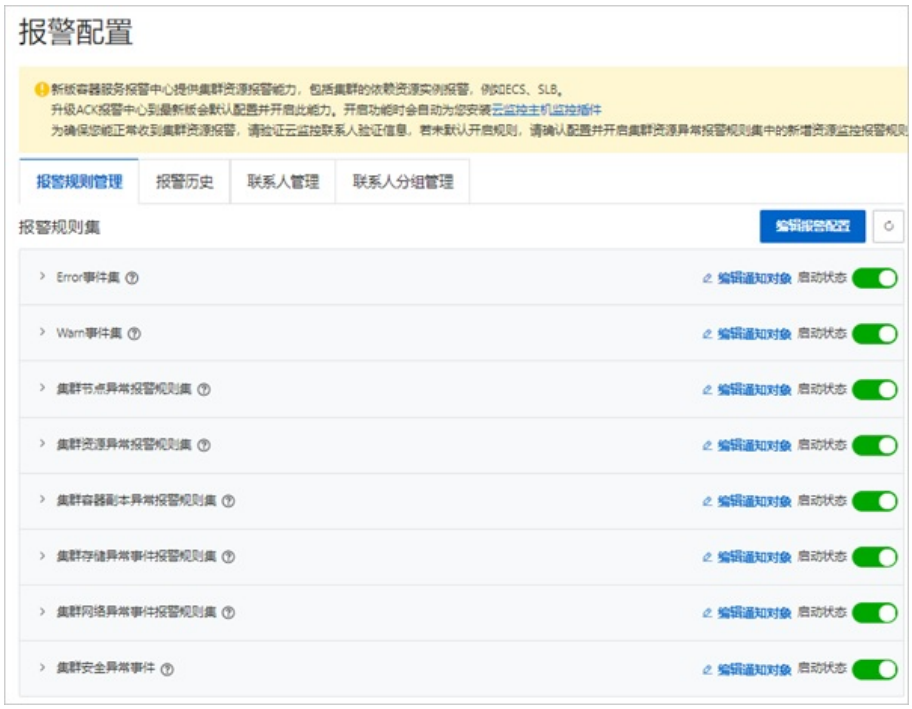
   - 已开通SLS日志服务云产品。当您首次使用日志服务时，需要登录日志服务控制台，根据页面提示开通日志服务。
   - 已安装**事件中心**。具体操作，请参见事件监控。
   - 集群托管组件alicloud-monitor-controller升级到最新版本。更多信息，请参见alicloud-monitor-controller。



接入报警配置功能

步骤一：开启默认报警规则

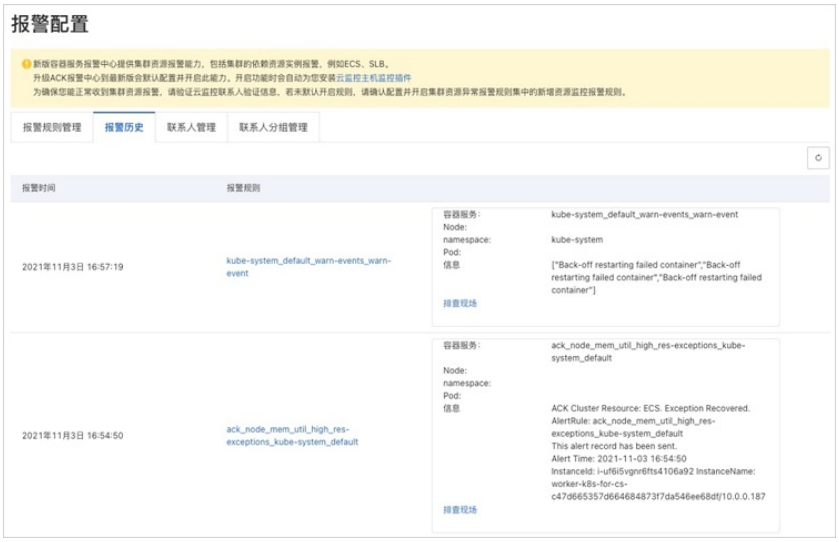
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。
5. 在**报警规则管理**页签，开启对应报警规则集。



步骤二：手动配置报警规则

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。

功能特性	说明
报警规则管理	○ 容器服务报警规则功能会默认生成容器场景下的报警模板（包含异常事件报警、异常指标报警）。 ○ 报警规则被分类为若干个报警规则集，可为报警规则集关联多个联系人分组，并启动或关闭报警规则集。 ○ 报警规则集中包含多个报警规则，一个报警规则对应单个异常的检查项。多个报警规则集可以通过一个YAML资源配置到对应集群中，修改YAML会同步生成报警规则。 ○ 关于报警规则YAML配置，请参见如何通过CRD配置报警规则。 ○ 关于默认报警规则模板，请参见默认报警规则模板。
报警历史	目前可查看最近发送的近100条历史记录。单击对应报警**报警规则类型**的链接，可跳转到对应监控系统中查看详细规则配置；单击对应报警**排查现场**的链接可快速定位到异常发生的资源页面（异常事件、指标异常的资源）。 
联系人管理	对联系人进行管理，可创建、编辑或删除联系人。
联系人分组管理	对联系人分组进行管理，可创建、编辑或删除联系人分组。当无联系人分组时，控制台会从您的阿里云账号注册信息中同步创建一个默认联系人分组。

5. 在**报警规则管理**页签，单击**编辑通知对象**可设置关联的通知对象；打开**启动状态**可开启对应报警规则集。

如何通过CRD配置报警规则

报警配置功能开启时，会默认在**kube-system**命名空间下创建一个AckAlertRule类型的资源配置，包含默认报警规则模板。容器服务报警规则集可通过此资源配置在集群中。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏，选择**运维管理 > 报警配置**。
5. 在**报警规则管理**页签中，单击右上角**编辑报警配置**，可查看当前集群中的AckAlertRule资源配置，并可通过YAML文件修改。

报警规则配置的YAML文件示例如下：

```
apiVersion: alert.alibabacloud.com/v1beta1
kind: AckAlertRule
metadata:
 name: default
spec:
 groups:
  #以下是一个集群事件报警规则配置样例。
  - name: pod-exceptions                #报警规则分组名，对应报警模板中的Group_Name字段。
    rules:
     - name: pod-oom                    #报警规则名。
       type: event                      #报警规则类型(Rule_Type)，枚举值为event（事件类型）、metric-cms（云监控指标类型）。
       expression: sls.app.ack.pod.oom  #报警规则表达式，当规则类型为event时，表达式的值为本文默认报警规则模板中Rule_Expression_Id值。
       enable: enable                   #报警规则开启状态，枚举值为enable、disable。
     - name: pod-failed
       type: event
       expression: sls.app.ack.pod.failed
       enable: enable
  #以下是一个集群基础资源报警规则配置样例。
  - name: res-exceptions                #报警规则分组名，对应报警模板中的Group_Name字段。
    rules:
     - name: node_cpu_util_high         #报警规则名。
       type: metric-cms                 #报警规则类型(Rule_Type)，枚举值为event（事件类型）、metric-cms（云监控指标类型）。
       expression: cms.host.cpu.utilization  #报警规则表达式，当规则类型为metric-cms时，表达式的值为本文默认报警规则模板中Rule_Expression_Id值。
       contactGroups:                   #报警规则映射的联系人分组配置，由ACK控制台生成，同一个账号下联系人相同，可在多集群中复用。
       enable: enable                   #报警规则开启状态，枚举值为enable、disable。
       thresholds:                      #报警规则阈值，详情见文档如何更改报警规则阈值部分。
        - key: CMS_ESCALATIONS_CRITICAL_Threshold
          unit: percent
          value: '1'
```

默认报警规则模板

在以下情况下注册集群会默认创建相应报警规则：

- 开启默认报警规则功能。
- 未开启默认报警规则，首次进入报警规则页面。

默认创建的报警规则如下表所示。

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
critical-events集群重要事件报警规则集	集群Error事件	集群中所有Error Level异常事件触发该报警。	event	error-event	sls.app.ack.error
	集群Warn事件	集群中关键Warn Level异常事件触发该报警，排除部分可忽略事件。	event	warn-event	sls.app.ack.warn

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-error集群异常事件报警规则集	集群节点Docker进程异常	集群中节点Dockerd或Containerd 运行时异常。	event	docker-hang	sls.app.ack.docker.hang
	集群驱逐事件	集群中发生驱逐事件。	event	eviction-event	sls.app.ack.eviction
	集群GPU的XID错误事件	集群中GPU XID异常事件。	event	gpu-xid-error	sls.app.ack.gpu.xid_error
	集群节点下线	集群中节点下线。	event	node-down	sls.app.ack.node.down
	集群节点重启	集群中节点重启。	event	node-restart	sls.app.ack.node.restart
	集群节点时间服务异常	集群中节点时间同步系统服务异常。	event	node-ntp-down	sls.app.ack.ntp.down
	集群节点PLEG异常	集群中节点PLEG异常。	event	node-pleg-error	sls.app.ack.node.pleg_error
	集群节点进程异常	集群中节点进程数异常。	event	ps-hang	sls.app.ack.ps.hang
	集群节点-CPU使用率≥85%	集群中节点实例CPU使用率超过水位。默认值85%。 在剩余资源不足15%时，Kubelet会发生强制驱逐行为。 关于如何调整阈值，请参见。	metric-cms	node_cpu_util_high	cms.host.cpu.utilization
	集群节点-内存使用率≥85%	集群中节点实例内存使用率超过水位。默认值85%。 在剩余资源不足15%时，Kubelet会发生强制驱逐行为。 关于如何调整阈值，请参见。	metric-cms	node_mem_util_high	cms.host.memory.utilization
	集群节点-磁盘使用率≥85%	集群中节点实例磁盘使用率超过水位。默认值85%。 关于如何调整阈值，请参见。	metric-cms	node_disk_util_high	cms.host.disk.utilization

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
res-exceptions 集群资源异常报警规则集	集群节点-公网流出带宽使用率≥85%	集群中节点实例公网流出带宽使用率超过水位。默认值85%。 关于如何调整阈值，请参见。	metric-cms	node_public_net_util_high	cms.host.public.network.utilization
	集群节点-inode使用率≥85%	集群中节点实例inode使用率超过水位。默认值85%。 关于如何调整阈值，请参见。	metric-cms	node_fs_inode_util_high	cms.host.fs.inode.utilization
	集群资源-负载均衡最大连接数使用率≥85%	集群中负载均衡实例最大连接数超过水位。默认值85%。 说明 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见。	metric-cms	slb_qps_util_high	cms.slb.qps.utilization
	集群资源-负载均衡网络流出带宽使用率≥85%	集群中负载均衡实例网络流出带宽使用率超过水位。默认值85%。 说明 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见。	metric-cms	slb_traff_tx_util_high	cms.slb.traffic.tx.utilization

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
	集群资源-负载均衡最大连接数使用率≥85%	集群中负载均衡实例最大连接数使用率超过水位。默认值85%。 说明 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见。	metric-cms	slb_max_con_util_high	cms.slb.max.connection.utilization
	集群资源-负载均衡监听每秒丢失连接数持续≥1	集群中负载均衡实例每秒丢失连接数持续超过水位。默认值1次。 说明 负载均衡实例，即API-Server、Ingress所关联的SLB负载均衡实例。 关于如何调整阈值，请参见。	metric-cms	slb_drop_con_high	cms.slb.drop.connection
	集群节点文件句柄过多	集群中节点文件句柄数过多异常。	event	node-fd-pressure	sls.app.ack.node.fd_pressure
	集群节点磁盘空间不足	集群中节点磁盘空间不足异常事件。	event	node-disk-pressure	sls.app.ack.node.disk_pressure
	集群节点进程数过多	集群中节点进程数过多异常事件。	event	node-pid-pressure	sls.app.ack.node.pid_pressure
	集群节点调度资源不足	集群中无调度资源异常事件。	event	node-res-insufficient	sls.app.ack.resource.insufficient
	集群节点IP资源不足	集群中IP资源不足异常事件。	event	node-ip-pressure	sls.app.ack.ip.not_enough

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
pod-exceptions 集群容器副本异常报警规则集	集群容器副本OOM	集群容器副本Pod或其中进程出现OOM（Out of Memory）。	event	pod-oom	sls.app.ack.pod.oom
	集群容器副本启动失败	集群容器副本Pod启动失败事件（Pod Start Failed）。	event	pod-failed	sls.app.ack.pod.failed
	集群镜像拉取失败事件	集群容器副本Pod出现镜像拉取失败事件。	event	image-pull-back-off	sls.app.ack.image.pull_back_off
	无可用LoadBalancer	集群无法创建LoadBalancer事件。请提交工单联系容器服务团队。	event	slb-no-ava	sls.app.ack.ccm.no_ava_slb
	同步LoadBalancer失败	集群创建LoadBalancer同步失败事件。请提交工单联系容器服务团队。	event	slb-sync-err	sls.app.ack.ccm.sync_slb_failed
	删除LoadBalancer失败	集群删除LoadBalancer失败事件。请提交工单联系容器服务团队。	event	slb-del-err	sls.app.ack.ccm.del_slb_failed
	删除节点失败	集群删除节点失败事件。请提交工单联系容器服务团队。	event	node-del-err	sls.app.ack.ccm.del_node_failed
	添加节点失败	集群添加节点失败事件。请提交工单联系容器服务团队。	event	node-add-err	sls.app.ack.ccm.add_node_failed
	创建VPC网络路由失败	集群创建VPC网络路由失败事件。请提交工单联系容器服务团队。	event	route-create-err	sls.app.ack.ccm.create_route_failed
	同步VPC网络路由失败	集群同步VPC网络路由失败事件。请提交工单联系容器服务团队。	event	route-sync-err	sls.app.ack.ccm.sync_route_failed

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-ops-err 集群管控运维异常报警规则集	托管节点池命令执行失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-run-cmd-err	sls.app.ack.nlc.run_command_fail
	托管节点池未提供任务的具体命令	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-empty-cmd	sls.app.ack.nlc.empty_task_cmd
	托管节点池出现未实现的任务模式	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-url-m-unimp	sls.app.ack.nlc.url_mode_unimpl
	托管节点池发生未知的修复操作	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-opt-no-found	sls.app.ack.nlc.op_not_found
	托管节点池销毁节点发生错误	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-des-node-err	sls.app.ack.nlc.destroy_node_fail
	托管节点池节点排水失败	集群托管节点池排水异常事件。请提交工单联系容器服务团队。	event	nlc-drain-node-err	sls.app.ack.nlc.drain_node_fail
	托管节点池重启ECS未达到终态	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-restart-ecs-wait	sls.app.ack.nlc.restart_ecs_wait_fail
	托管节点池重启ECS失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-restart-ecs-err	sls.app.ack.nlc.restart_ecs_fail
	托管节点池重置ECS失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-reset-ecs-err	sls.app.ack.nlc.reset_ecs_fail
	托管节点池自愈任务失败	集群托管节点池异常事件。请提交工单联系容器服务团队。	event	nlc-sel-repair-err	sls.app.ack.nlc.repair_fail
	Terway资源无效	集群Terway网络资源无效异常事件。请提交工单联系容器服务团队。	event	terway-invalid-res	sls.app.ack.terway.invalid_resource

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-network-err集群网络异常事件报警规则集	Terway分配IP失败	集群Terway网络资源分配IP失败异常事件。请提交工单联系容器服务团队。	event	terway-alloc-ip-err	sls.app.ack.terway.alloc_ip_fail
	解析Ingress带宽配置失败	集群Ingress网络解析配置异常事件。请提交工单联系容器服务团队。	event	terway-parse-err	sls.app.ack.terway.parse_fail
	Terway分配网络资源失败	集群Terway网络资源分配失败异常事件。请提交工单联系容器服务团队。	event	terway-alloc-res-err	sls.app.ack.terway.allocate_failure
	Terway回收网络资源失败	集群Terway网络资源回收失败异常事件。请提交工单联系容器服务团队。	event	terway-dispose-err	sls.app.ack.terway.dispose_failure
	Terway虚拟模式变更	集群Terway网络虚拟模式变更事件。	event	terway-virt-mod-err	sls.app.ack.terway.virtual_mode_change
	Terway触发PodIP配置检查	集群Terway网络触发PodIP配置检查事件。	event	terway-ip-check	sls.app.ack.terway.config_check
	Ingress重载配置失败	集群Ingress网络配置重载异常事件。请检查Ingress配置是否正确。	event	ingress-reload-err	sls.app.ack.ingress.err_reload_nginx
	云盘容量少于20 GiB限制	集群网盘固定限制，无法挂载小于20 GiB的磁盘。请检查所挂载云盘的容量大小。	event	csi_invalid_size	sls.app.ack.csi.invalid_disk_size
	容器数据卷暂不支持包年包月类型云盘	集群网盘固定限制，无法挂载包年包月类型的云盘。请检查所挂载云盘的售卖方式。	event	csi_not_portable	sls.app.ack.csi.disk_not_portable

规则集类型	规则名	规则说明	Rule_Type	ACK_CR_Rule_Name	SLS_Event_ID
cluster-storage-err集群存储异常事件报警规则集	挂载点正在被进程占用，卸载挂载点失败	集群存储挂载点正在被进程占用，卸载挂载点失败。	event	csi_device_busy	sls.app.ack.csi.deivce_busy
	无可用云盘	集群存储挂载时无可用云盘异常。请提交工单联系容器服务团队。	event	csi_no_ava_disk	sls.app.ack.csi.no_ava_disk
	云盘IOHang	集群出现IOHang异常。请提交工单联系容器服务团队。	event	csi_disk_iohang	sls.app.ack.csi.disk_iohang
	磁盘绑定的PVC发生slowIO	集群磁盘绑定的PVC发生slowIO异常。请提交工单联系容器服务团队。	event	csi_latency_high	sls.app.ack.csi.latency_too_high
	磁盘容量超过水位阈值	集群磁盘使用量超过水位值异常。请检查你的集群磁盘水位情况。	event	disk_space_press	sls.app.ack.csi.no_enough_disk_space
security-err集群安全异常事件	安全巡检发现高危风险配置	集群安全巡检发现高危风险配置事件。请提交工单联系容器服务团队。	event	si-c-a-risk	sls.app.ack.si.config_audit_high_risk

25.6.1.4. 将应用实时监控服务ARMS接入注册集群

注册的集群接入应用实时监控服务ARMS能为分布在各处的Kubernetes集群提供统一的管理方式。本文介绍如何通过容器服务Kubernetes版中的应用将应用实时监控服务ARMS接入至注册的Kubernetes集群。

前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，选中**ack-arms-pilot**应用。

   阿里云应用包含较多应用，您可在页面右上角搜索**ack-arms-pilot**，支持关键字搜索。
4. 在**应用目录 - ack-arms-pilot**页面右侧**创建**区域，选择目标集群。
5. 在**应用目录 - ack-arms-pilot**页面单击**参数**页签，设置相应的参数，然后单击右侧**创建**区域的**创建**。



参数	描述
accessKey	您的阿里云AccessKey ID。AK权限需包含访问ARMS的权限。
accessKeySecret	您的阿里云AccessKey Secret。

注意

- 如果您的集群和专有网络VPC之间有专线，专线会被自动使用。
- 如果您是通过公网注册的外部集群，需要删除镜像参数中的vpc。

后续步骤

验证安装是否成功，请参见为容器服务Kubernetes版Java应用安装探针。

相关文档

有关如何使用ARMS，请参见ARMS应用总览。

注册集群概述

25.6.1.5. 将arms-prometheus接入注册集群

注册的集群接入arms-prometheus能为分布在各处的Kubernetes集群提供统一的管理方式。本文介绍如何通过容器服务Kubernetes版中的应用将arms-prometheus接入至注册的Kubernetes集群。

前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，选中ack-arms-prometheus应用。

   阿里云应用包含较多应用，您可在页面右上角搜索**ack-arms-prometheus**，支持关键字搜索。
4. 在**应用目录 - ack-arms-prometheus**页面右侧**创建**区域，选择目标集群，并单击**创建**。



参数	描述	备注
cluster_id	您的集群ID。	根据选择的集群自动生成。
uid	您的阿里云账号用户名。 **说明** userID需要带引号。	
region_id	应用实时监控服务ARMS所在的地域。	

注意

- 如果您的集群和专有网络VPC之间有专线，专线会被自动使用。
- 如果您是通过公网注册的外部集群，需要删除镜像参数中的vpc。例如，删除vpc后的镜像参数为registry.cn-hangzhou.aliyuncs.com/arms-docker-repo/arms-prom-operator:v0.1。

执行结果

安装完成后，可查看监控数据和定义告警规则。更多信息，请参见阿里云Prometheus监控和创建Prometheus监控报警。

相关文档

- 注册集群概述

25.6.1.6. 将alibaba-cloud-metrics-adapter接入注册集群

您可以通过alibaba-cloud-metrics-adapter使用云指标进行Pod的水平伸缩（HPA）。本文介绍如何通过容器服务Kubernetes版中的应用将alibaba-cloud-metrics-adapter接入至注册的Kubernetes集群。

前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

背景信息

在Kubernetes中，监控数据除了用来展现具体的指标和数据，还有一个重要的用途就是通过HPA进行数据消费。为了满足开发者不同层次的监控需求，Kubernetes定义了三种不同的监控数据接口，分别是Resource Metric、Custom Metric以及External Metric：

- Resource Metric通常是通过Metrics-Server进行采集，提供的主要是Pod、Node、Namespcae等Kubernetes中

内置逻辑对象的监控。

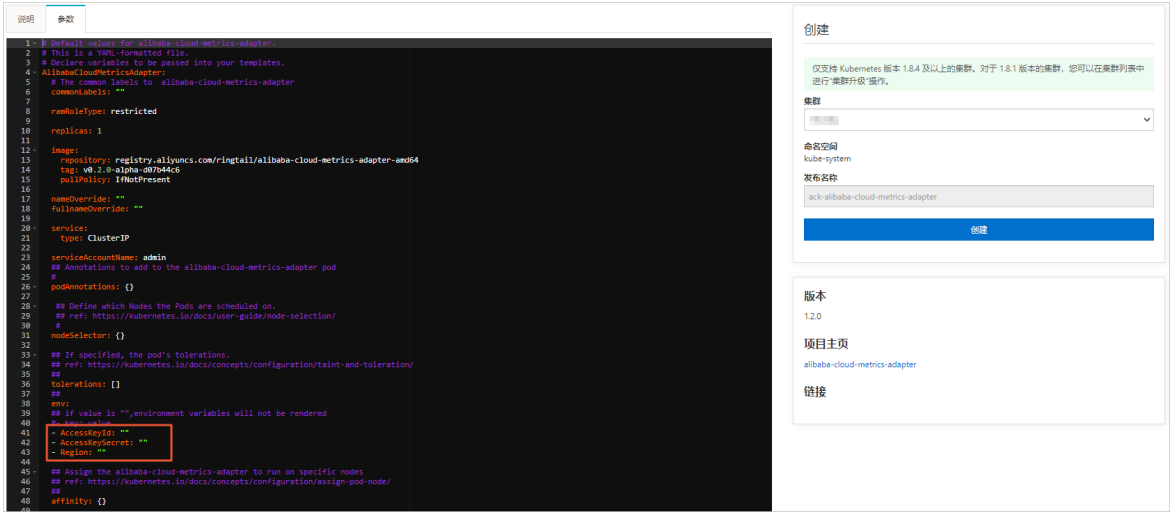
- Custom Metric是您自定义的监控指标，通常是通过Prometheus进行采集，然后再HPA进行消费。
- External Metric主要针对外部指标，通常是面向云场景，例如您可以通过云厂商的External Metric来获取Ingress的QPS（每秒查询率）作为弹性指标。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，选中**ack-alibaba-cloud-metrics-adapter**应用。

   阿里云应用包含较多应用，您可在页面右上角搜索**ack-alibaba-cloud-metrics-adapter**，支持关键字搜索。
4. 在**应用目录 - ack-alibaba-cloud-metrics-adapter**页面右侧的**创建**区域，选择目标集群。

   命名空间和**发布名称**为默认值，无需设置。
5. 在**应用目录 - ack-alibaba-cloud-metrics-adapter**单击**参数**页签，设置相应的参数，然后单击右侧的**创建**区域的**创建**。



参数	描述
AccessKeyId	您的阿里云AccessKeyId。
AccessKeySecret	您的阿里云AccessKeySecret。
Region	您集群所在的地域，例如cn-qingdao、ap-southeast-1。

> **说明** 如果您的集群和专有网络VPC之间有专线，专线会被自动使用。

相关文档

- 注册集群概述
- 使用Ingress指标进行弹性伸缩

25.6.2. 微服务

25.6.2.1. 将MSE接入注册集群

微服务引擎MSE（Microservice Engine）是一个面向业界主流开源微服务框架Spring Cloud和Dubbo一站式微服务平台，提供治理中心、托管的注册中心和托管的配置中心。本文介绍如何将MSE接入注册集群应用中。

前提条件

您已通过容器服务Kubernetes版接入一个Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

安装ack-mse-pilot组件

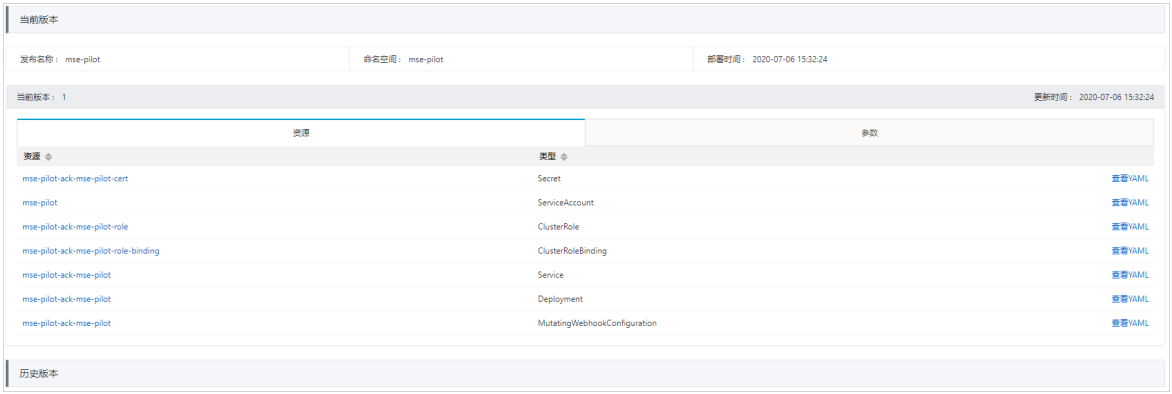
1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面搜索并单击*ack-mse-pilot*。
4. 在**应用目录- ack-mse-pilot**页面右侧区域选择目标**集群**，单击**参数**页签，设置相应的参数。

```
rbac:
  create: true

admissionRegistration:
  # valid values are "ValidatingWebhookConfiguration", and "MutatingWebhookConfiguration"
  kind: MutatingWebhookConfiguration
  path: /mutating-services
  # valid values are "Ignore" and "Fail"
  failurePolicy: Ignore
controller:
  #  namespace: .Release.Namespace
  image: registry-vpc.cn-hangzhou.aliyuncs.com/arms-docker-repo/arms-pilot:v1.41-mse

  imagePullPolicy: Always
  serviceAccount: mse-pilot
  logLevel: 1

  # Please Fill ClusterId
  cluster_id:

  # if your cluster is hosted. please fill aliyun ak/sk
  accessKey: __ACCESSKEY__
  accessKeySecret: __ACCESSKEY_SECRET__

  # if your cluster is hosted. please fill userId and regionId.
  # userId and regionId can be auto filled in ACS.
  uid: "                "

  region_id: cn-hangzhou

  accessSource: ACSK8S
  tls:
    # Admission controller server will inherit this CA from the
    # extension-apiserver-authentication ConfigMap if available.
    requestHeaderCA:
  service:
    # Type of service; valid values are "LoadBalancer" and "NodePort"
    # NodePort is useful if deploying on bare metal or hacking locally on
    # minikube
    type: ClusterIP
    port: 443
    targetPort: 8443
```

参数	说明
image	ack-mse-pilot组件镜像地址。 如果您是通过公网注册的外部集群，需要删除镜像参数中的VPC。例如，镜像参数为 `registry-vpc.cn-hangzhou.aliyuncs.com/arms-docker-repo/arms-pilot:v1.41-mse` ，删除VPC后的镜像参数为registry.cn-hangzhou.aliyuncs.com/arms-docker-repo/arms-pilot:v1.41-mse。

参数	说明
AccessKeyId	您的阿里云AccessKeyId。AccessKey需要包含MSE访问权限。具体操作，请参见为集群授予MSE治理中心的访问权限。
AccessKeySecret	您的阿里云AccessKeySecret。

5. 单击**创建**。
   安装ack-mse-pilot组件大约需要2分钟，请耐心等待。
   创建成功后，会自动跳转到目标集群的**发布**页面，检查安装结果。若下图中所有资源创建成功，则说明组件安装成功。



后续步骤

完成上述步骤后，重启应用，您就为部署在容器服务Kubernetes版中的应用开启了MSE微服务治理。

在**无状态**页面，单击目标应用**操作**列下方的**更多**，会出现**微服务治理**选项。单击**微服务治理**，可跳转至MSE应用详情页面，即可使用MSE微服务治理对您的Spring Cloud和Dubbo应用进行服务治理。相关内容，请参见使用指引。

> **说明** 若目标应用**操作**列下方的**更多**中没有出现**微服务治理**选项，请检查您的应用接入MSE后是否已经重启。

25.6.2.2. 将AHAS-架构感知接入注册集群

注册的集群接入AHAS-架构感知能为分布在各处的Kubernetes集群提供统一的管理方式。本文介绍如何通过容器服务Kubernetes版中的应用将AHAS-架构感知接入至注册的Kubernetes集群。

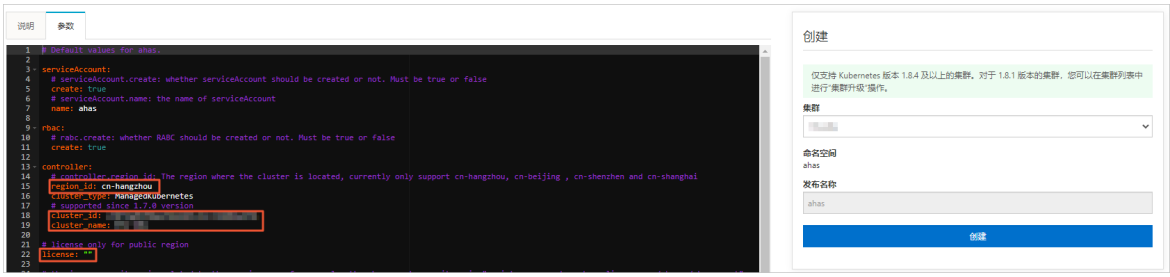
前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

操作步骤

1. 查看AHAS产品的License，备用。
   查看AHAS产品License的具体步骤，请参见查看License。
2. 登录容器服务管理控制台。
3. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
4. 在**应用目录**页面单击**阿里云应用**页签，选中**ack-ahas-pilot**应用。
   阿里云应用包含较多应用，您可在页面右上角搜索**ack-ahas-pilot**，支持关键字搜索。
5. 在**应用目录 - ack-ahas-pilot**页面右侧**创建**区域，选择目标集群。

6. 在**应用目录 - ack-ahas-pilot**页面单击**参数**页签，设置相应的参数，然后单击右侧**创建**区域的**创建**。

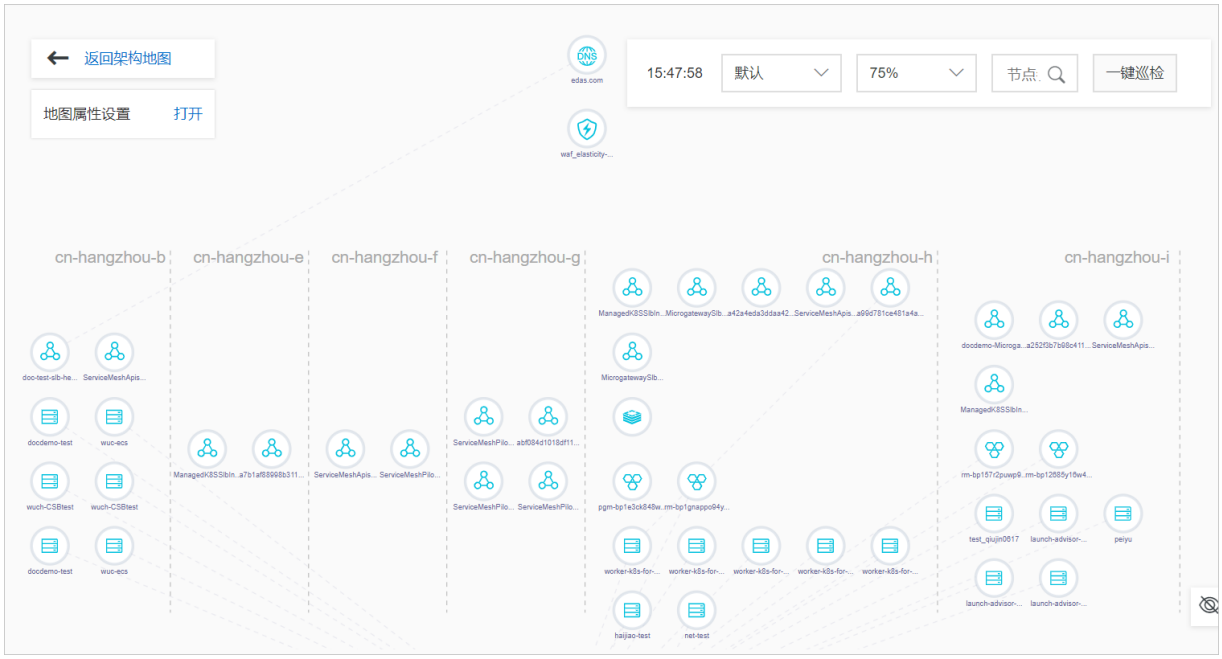


参数	描述	备注
region_id	◦ 如果集群和VPC之间有专线，该参数为专线连接的地域。 ◦ 如果集群和VPC之间没有专线，该参数填入**cn-public**	根据选择的集群自动生成。
cluster_id	您的集群ID。	
cluster_name	您的集群名称。	
license	填写您在AHAS控制台中查看到的License。	无

执行结果

部署完成后您可以在AHAS控制台中查看集群的拓扑图。
如果集群和VPC之间没有专线，即**region_id**参数填写的是**cn-pubulic**

拓扑图示例如下：



相关文档

- 注册集群概述

25.6.2.3. 将AHAS-应用限流接入注册集群

注册的集群接入AHAS-应用限流能为分布在各处的Kubernetes集群提供统一的管理方式。本文介绍如何通过容器服务Kubernetes版中的应用将AHAS-应用限流接入至注册的Kubernetes集群。

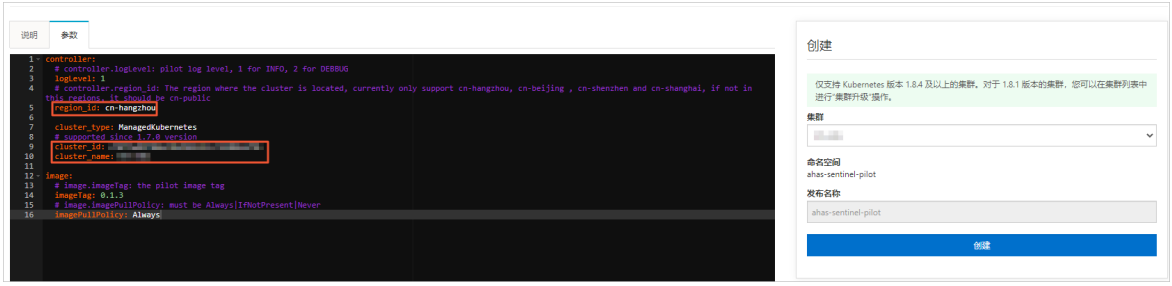
前提条件

您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面单击**阿里云应用**页签，选中**ack-ahas-sentinel-pilot**应用。

   阿里云应用包含较多应用，您可在页面上角搜索**ack-ahas-sentinel-pilot**，支持关键字搜索。
4. 在**应用目录 - ack-ahas-sentinel-pilot**页面右侧**创建**区域，选择目标集群，然后单击**创建**。



参数	描述	备注
region_id	○ 如果集群和VPC之间有专线，该参数为专线连接的region。 ○ 如果集群和VPC之间没有专线，该参数填入**cn-public**	根据所选集群自动生成。
cluster_id	您的集群ID。	
cluster_name	您的集群名称。	

结果验证

验证安装是否成功，请参见步骤二：为Java应用开启AHAS应用防护。

如果集群和VPC之间没有专线，即**region_id**参数填写的是**cn-public**，需要在应用的 `annotations` 里加入**ahasLicenseKey**。有关**ahasLicenseKey**获取方法，请参见查看License。

相关文档

- 注册集群概述

25.6.3. 集群成本分析

ACK的成本分析功能可以协助企业IT管理员了解集群资源使用量及成本分布，提升集群资源利用率，提供成本节约建议。本文介绍如何配置集群成本分析及集群成本分析功能介绍。

前提条件

- 您已创建1.18.8及以上版本的Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。
- Kubernetes集群已开启阿里云Prometheus监控。具体操作，请参见开启阿里云Prometheus监控。
- 使用成本分析功能前，请提交工单申请添加白名单。

功能介绍

容器服务成本分析主要包含以下几个重要的功能：

- 云资源费用分析
  将Kubernetes集群下所有的云资源费用进行聚合，支持查看集群整体费用的同比与环比。
- 集群费用趋势分析
  查看Kubernetes集群下所有云资源的消费趋势，分析热点云产品消费，协助成本节约决策。
- 集群成本节约推荐
  将Kubernetes集群的费用趋势进行分析，提供资源用量和成本消费的相关性分析，提供成本节约建议。
- 实时价格模拟预测
  将集群中计算资源的实时价格进行模拟，提供小时维度的成本预估，支持包年包月、按量付费、竞价付费等多种付费方式。
- 命名空间资源分账
  根据命名空间拆分集群成本比例，提供实时成本预估、成本趋势变化、集群成本占比等能力。
- 应用成本优化
  在命名空间下查看Top 10的成本消耗最多的应用以及应用成本消费的趋势，提供应用维度的成本优化分析能力。

配置成本分析

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**运维管理 > 成本分析**。
5. 在**成本分析**页面单击集群的**KubernetesWorkerRole-*****，在**RAM控制台**完成RAM授权策略。



   i. 在**RAM角色管理**页面单击**K8sWorkerRolePolicy-*****权限策略名称。
   ii. 进入**权限策略管理**页面，单击**修改策略内容**。

iii. 在**修改策略内容**面板增加以下内容后，单击**确定**。

```
{
  "Action": [
    "bssapi:QueryInstanceBill"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
{
  "Action": [
    "ecs:DescribeSpotPriceHistory",
    "ecs:DescribeInstances",
    "ecs:DescribePrice"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

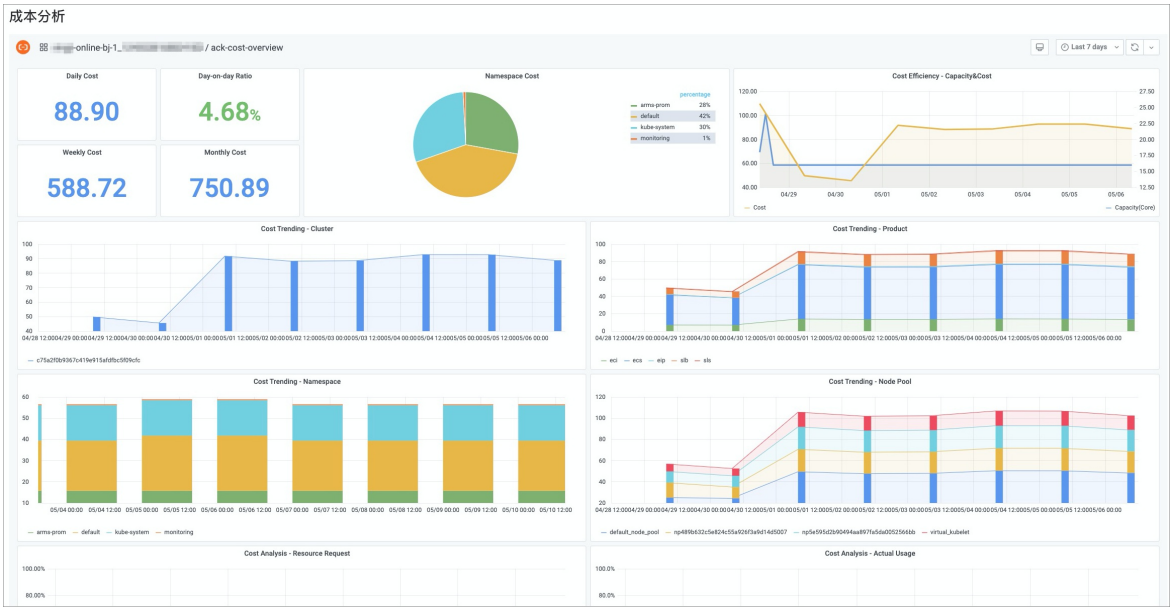
> **说明** 多个策略内容需要用半角逗号（,）分隔。

6. 在**成本分析**页面单击**开始安装**。

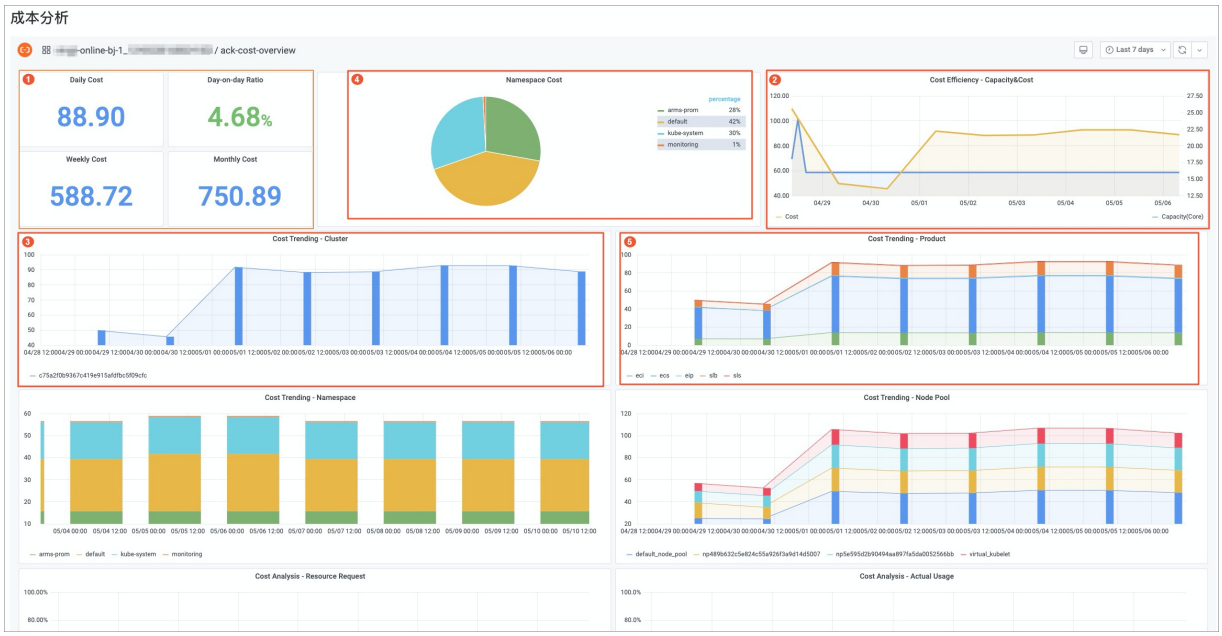


7. 在**成本分析**页面查看成本可视化大盘。

> **说明** 在开启成本分析功能后第二天08: 00自动显示账单数据。



使用成本分析能力



成本可视化大盘的相关功能介绍如下表：

序号	功能	描述
①	通过成本趋势判断集群浪费情况	成本的费用，Daily Cost为昨日费用，Day-on-day Ratio为相对昨日费用的变化比率。 当相对昨日费用的变化比率的字体颜色为绿色时，表示相比前一日的成本是降低的；当相对昨日费用的变化比率的字体颜色为红色时，表示相比前一日的成本是增涨的。
②		黄色表示成本消费，蓝色表示实际的集群容量。通常情况下，两条曲线会存在一定的相关性。 当对两者进行相关性对比后发现两者呈现不一致的趋势，表示集群的单位核成本异常，需要检查是否使用了更贵的资源。

序号	功能	描述
③		一段时间内的成本变化趋势。
④	通过命名空间进行部门成本分账	在一个集群中，可能存在各种不同的规格、付费方式的节点类型，当您通过命名空间进行分账时，不能够单一的通过每个命名空间的资源申请值进行判断，同时需要考虑命名空间下Pod所在节点情况。 成本分析通过alibaba-cloud-price-exporter将每个节点的实时费用进行了转换，在统计命名空间维度的费用时，可转换为： `Σ(Pod资源申请/节点容量)*节点单位价格` 这种方式可以精确的估算出命名空间的成本费用，不过由于用户折扣、代金券抵扣、包年包月等各种延迟计费策略，可能会导致命名空间的成本费用和集群的实际账单费用不一致。但是，可以通过命名空间的占比乘以集群总费用的方式获得命名空间级别的分账。
⑤	通过云产品费用趋势与比例分析云产品成本	在集群中会包含多种不同的云产品，不同的使用方式、计费模型会导致云产品的费用上存在比较多的差异。可以通过成本趋势和组合看到不同的云产品消费，从而进行成本决策。

常见问题

问题一：为什么开启成本可视化后没有数据显示？

问题二：为什么命名空间的费用相加与实际的费用账单不相等？

25.7. 使用Git源仓库创建应用

通过ACK的应用中心创建应用可以直观地查看所有Kubernetes资源的部署状态与变化。根据数据源的来源不同，创建应用的方式也不同。本文介绍如何使用Git源仓库在应用中心创建应用。

前提条件

已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

示例仓库地址

- Git Hub仓库地址
- 阿里云Code仓库地址

Git源仓库中应用编排方式

- Kubernetes原生的Manifests编排

```
.
├── deployment.yaml
├── ingress.yaml
└── service.yaml
```

- Helm编排

```
.
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
├── values-pro.yaml
└── values.yaml
```

- Kustomize编排

```
.
├── base
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── pre
    │   ├── deployment.yaml
    │   └── kustomization.yaml
    └── pro
        ├── deployment.yaml
        └── kustomization.yaml
```

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击**创建应用**。
4. 在**应用来源**配置向导配置参数。
   i. 在**安装应用发布组件**区域选择集群，然后单击**开始安装**。

      > **说明** 仅首次安装应用需要执行此步骤。

   ii. 在**应用来源**区域单击**Git仓库**。
   iii. 单击**下一步**。
5. 在**应用配置**配置向导中设置参数，然后单击**下一步**。

参数	说明
应用名称	设置应用名称。
应用描述	设置应用描述。
仓库	选择仓库，您可以选择已创建的仓库或者使用新仓库。本文以使用新仓库为例。
仓库地址	设置仓库的地址。

参数	说明
名称	设置仓库的名称。
连接方式	选择连接方式，支持**HTTPS**和**SSH**。以下为不同连接方式需要设置的参数： ○ 当设置连接方式为**HTTPS**，需要设置以下参数。 ■ **用户名（可选）**：如果是私有Git仓库，需要设置用户名。 ■ **密码（可选）**：如果是私有Git仓库，需要设置密码。 ■ **TLS客户端证书（可选）**：如果Git服务器使用的是自签发证书，需要设置TLS客户端证书。 ■ **TLS客户端证书密钥（可选）**：如果Git服务器使用的是自签发证书，需要设置TLS客户端证书密钥。 ■ **跳过服务端证书校验**：如果Git服务器使用的是自签发证书，且想要省略TLS客户端证书和TLS客户端证书密钥的配置，则可以选中**跳过服务端证书校验**。 ■ **LFS支持**：是否需要配置Git启用LFS支持。 ○ 当设置连接方式为**SSH**，需要设置以下参数。 ■ **SSH私钥**：设置SSH私钥。 ■ **跳过服务端证书校验**：如果Git服务器使用的是自签发证书，且想要省略SSH私钥的配置，则可以选中**跳过服务端证书校验**。 ■ **LFS支持**：是否需要配置Git启用LFS支持。
分支	分支名称。
路径	Git源仓库中的应用编排所在的子目录。

6. 在**发布配置**配置向导页面中设置参数。
   i. 在**发布配置**区域选择发布方式，可选**全量发布**和**灰度发布**。

      > **说明** 第一次发布应用时只能选择**全量发布**，后续对应用进行更新迭代时，可以选择**全量发布**或**灰度发布**。

   ii. 在**可选集群**区域单击集群左侧的**添加**后，在**目标集群**区域可以看到选择的集群，然后在**目标集群**区域选择应用要发布的**命名空间**。

      > **说明** 如果没有开通公网访问的集群，需要为API Server绑定EIP，以开启公网访问能力。

   iii. 单击**开始发布**。
7. 在**发布确认**对话框单击**确定**。

   在**发布日志**配置页面中可以看到发布应用的进度。

相关文档

- 应用中心概述
- 管理应用
- 使用编排模板创建应用

25.8. 在注册集群中使用阿里云CSI存储

阿里云容器服务ACK的容器存储功能深度融合阿里云存储服务，完全兼容Kubernetes原生的存储服务，可通过部署CSI存储组件接入阿里云存储服务。ACK支持Pod自动绑定阿里云云盘、NAS、OSS、CPFS、本地卷等存储服务。本文介绍如何在注册集群中使用阿里云CSI存储。

前提条件

- 已创建注册集群，并将自建Kubernetes集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
- 已为您的自建Kubernetes集群扩容了阿里云ECS节点。关于如何为自建Kubernetes集群扩容阿里云ECS节点，请参见创建节点池并扩容。

  > **注意** 阿里云CSI存储组件只支持运行在阿里云ECS节点上，因此需要为自建Kubernetes集群中的阿里云ECS节点添加节点标签 `alibabacloud.com/external=true` 。

- 通过Kubectl工具连接注册集群。具体操作，请参见通过kubectl工具连接集群。

注意事项

- 如果您的自建Kubernetes集群部署在阿里云ECS上，您需要为ECS实例添加节点标签。关于如何为ECS实例添加节点标签，请参见自建Kubernetes集群接入ACK注册集群后的ECS节点初始化配置。
- 如果您使用注册集群节点池功能为本地数据中心自建Kubernetes集群扩容的阿里云ECS节点，则默认已添加节点标签 `alibabacloud.com/external=true` 。

步骤一：在自建Kubernetes集群中配置CSI组件的RAM权限

在注册集群中安装CSI存储组件前，您需要在自建Kubernetes集群中设置AccessKey用来访问云服务的权限。设置AccessKey前，您需要创建RAM用户并为其添加访问相关云资源的权限。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 创建自定义权限策略。

   关于创建自定义权限策略的操作步骤，请参见创建自定义策略。
   通过以下自定义策略示例，添加磁盘、快照、快照策略、资源标签、实例、文件系统及仓库的管理权限。关于API参考的更多信息，请参见API概览。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ecs:AttachDisk",
        "ecs:DetachDisk",
        "ecs:DescribeDisks",
        "ecs:CreateDisk",
        "ecs:ResizeDisk",
        "ecs:CreateSnapshot",
        "ecs:DeleteSnapshot",
        "ecs:CreateAutoSnapshotPolicy",
        "ecs:ApplyAutoSnapshotPolicy",
        "ecs:CancelAutoSnapshotPolicy",
        "ecs:DeleteAutoSnapshotPolicy",
        "ecs:DescribeAutoSnapshotPolicyEX",
        "ecs:ModifyAutoSnapshotPolicyEx",
        "ecs:AddTags",
        "ecs:DescribeTags",
        "ecs:DescribeSnapshots",
        "ecs:ListTagResources",
        "ecs:TagResources",
        "ecs:UntagResources"
```

```
        "ecs:ModifyDiskSpec",
        "ecs:CreateSnapshot",
        "ecs:DeleteDisk",
        "ecs:DescribeInstanceAttribute",
        "ecs:DescribeInstances"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "nas:DescribeFileSystems",
        "nas:DescribeMountTargets",
        "nas:AddTags",
        "nas:DescribeTags",
        "nas:RemoveTags",
        "nas:CreateFileSystem",
        "nas:DeleteFileSystem",
        "nas:ModifyFileSystem",
        "nas:CreateMountTarget",
        "nas:DeleteMountTarget",
        "nas:ModifyMountTarget",
        "nas:TagResources",
        "nas:SetDirQuota",
        "nas:EnableRecycleBin",
        "nas:GetRecycleBinAttribute"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "oss:PutBucket",
        "oss:GetObjectTagging",
        "oss:ListBuckets",
        "oss:PutBucketTags",
        "oss:GetBucketTags",
        "oss:PutBucketEncryption",
        "oss:GetBucketInfo"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。
4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。
5. 使用AccessKey在注册集群中创建名为*alibaba-addon-secret*的Secret资源。

   安装CSI组件时将自动引用此AccessKey访问对应的云服务资源。

```
kubectl -n kube-system create secret generic alibaba-addon-secret --from-literal='access-key-id=<your access key id>' --from-literal='access-key-secret=<your access key secret>'
```

> **说明** <your access key id> 及 <your access key secret> 为上一步获取的AccessKey信息。

步骤二：安装CSI组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理详情页左侧导航栏，选择**运维管理 > 组件管理**。
5. 单击**存储**页签，在csi-plugin及csi-provisioner卡片单击**安装**。
6. 在**提示**对话框中确认版本信息后单击**确定**。
   安装成功后，对应组件区域会提示安装成功，且可查看组件当前版本。

步骤三：使用存储卷

可在下表查看注册集群对于各存储卷的使用情况：

存储卷类型	参考链接
NAS	• 使用NAS静态存储卷 • 使用NAS动态存储卷
OSS	• 使用OSS静态存储卷 • 加密OSS存储卷
本地存储卷	• LVM数据卷 • HostPath数据卷 • LocalVolume数据卷 • QuotaPath数据卷 • 内存数据卷
CPFS	• CPFS静态卷 • CPFS动态卷
CNFS	• 使用CNFS托管NAS文件系统 • 使用CNFS动态创建NAS共享存储卷
云盘	注册集群不支持使用云盘。

25.9. 安全

25.9.1. 在注册集群中使用集群审计功能

在Kubernetes集群中，API Server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。本文主要介绍如何在注册集群中使用集群审计功能。

前提条件

已创建注册集群，并将自建Kubernetes集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

步骤一：在Master节点上配置审计配置策略文件

依次登录所有Master节点，在审计配置策略文件的路径*/etc/kubernetes/audit-policy.yaml*，请根据以下内容修改审计配置策略：

```
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
```

步骤二：在Master节点上配置Kube API Server文件

依次登录所有Master节点机器，在Kube API Server文件的路径*/etc/kubernetes/manifests/kube-apiserver.yaml*，请完成以下相关配置：

- 根据以下示例添加command参数 `--audit-log-*` ：

```
...
spec:
 containers:
 - command:
  - kube-apiserver
  - --audit-log-maxbackup=10
  - --audit-log-maxsize=100
  - --audit-log-path=/var/log/kubernetes/kubernetes.audit
  - --audit-log-maxage=30
  - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
  ...
```

- 根据以下示例添加env参数 `aliyun_logs_audit-*` ：

```
...
spec:
 containers:
 - command:
  - kube-apiserver
  - --audit-log-maxbackup=10
  - --audit-log-maxsize=100
  - --audit-log-path=/var/log/kubernetes/kubernetes.audit
  - --audit-log-maxage=30
  - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
  ...
  ...
  env:
  - name: aliyun_logs_audit-${cluster_id}
    value: /var/log/kubernetes/kubernetes.audit
  - name: aliyun_logs_audit-${cluster_id}_tags
    value: audit=apiserver
  - name: aliyun_logs_audit-${cluster_id}_product
    value: k8s-audit
  - name: aliyun_logs_audit-${cluster_id}_jsonfile
    value: "true"
  image: registry-vpc.cn-shenzhen.aliyuncs.com/acs/kube-apiserver:v1.20.4-aliyun.1
```

> **注意** 需要将{cluster_id}替换为您集群的Cluster ID。关于集群ID的获取，请参见查看基本信息。

- 根据以下示例挂载*/etc/kubernetes/audit-policy.yaml*到API Server Pod。

```
...
spec:
 containers:
 - command:
  - kube-apiserver
  - --audit-log-maxbackup=10
  - --audit-log-maxsize=100
  - --audit-log-path=/var/log/kubernetes/kubernetes.audit
  - --audit-log-maxage=30
  - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
  ...
  ...
  env:
  - name: aliyun_logs_audit-${cluster_id}
    value: /var/log/kubernetes/kubernetes.audit
  - name: aliyun_logs_audit-${cluster_id}_tags
    value: audit=apiserver
  - name: aliyun_logs_audit-${cluster_id}_product
    value: k8s-audit
  - name: aliyun_logs_audit-${cluster_id}_jsonfile
    value: "true"
  image: registry-vpc.cn-shenzhen.aliyuncs.com/acs/kube-apiserver:v1.20.4-aliyun.1
  ...
  ...
  volumeMounts:
  - mountPath: /var/log/kubernetes
    name: k8s-audit
  - mountPath: /etc/kubernetes/audit-policy.yaml
    name: audit-policy
    readOnly: true
  ...
  ...
 volumes:
 - hostPath:
    path: /var/log/kubernetes
    type: DirectoryOrCreate
  name: k8s-audit
 - hostPath:
    path: /etc/kubernetes/audit-policy.yaml
    type: FileOrCreate
  name: audit-policy
 ...
```

步骤三：安装logtail-ds日志组件

关于安装logtail-ds日志组件的具体操作，请参见步骤二：安装logtail-ds组件。

后续步骤

关于如何使用和查看集群审计功能，请参见使用集群审计功能。

25.9.2. 在注册集群中使用安全巡检功能

CIS Kubernetes基线是CIS推出的一系列用于构建一个安全可靠的Kubernetes集群的安全配置建议。本文介绍如何在命令行下通过security-inspector组件提供的功能实现集群安全CIS基线检查。

前提条件

- 已创建注册集群，并将自建Kubernetes集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心

集群。

- 已在注册集群中安装security-inspector组件。具体操作，请参见管理组件。

CIS基线概述

CIS是互联网安全中心（Center for Internet Security）的缩写，它针对一些常见的系统制定了一套安全配置指南，称之为基线。CIS基线是由一组互联网安全专家共同讨论和制定的，这些安全指南被政府、企业和学术机构广泛认可和执行。

CIS Kubernetes基线是针对开源Kubernetes发行版本所编写的，旨在尽可能广泛应用于各个发行版本，同时不同的基线版本与特定的Kubernetes版本相关联。更多信息，请参见CIS Kubernetes 基线。

CIS也推出了更符合各大云厂商实际场景的CIS Kubernetes基线，例如针对ACK场景的CIS基线，即CIS Alibaba Cloud Container Service For Kubernetes（ACK）Benchmark。

使用security-inspector组件实现集群安全CIS基线检查

您可以通过以下步骤按照CIS Kubernetes基线标准扫描集群，并获取CSV格式的扫描结果报告。

1. 使用以下命令创建一个扫描任务。

   执行CIS基线检查，程序会根据集群版本选择合适的基线版本进行检查。

```
kubectl apply -f - <<EOF
apiVersion: securityinspector.alibabacloud.com/v1alpha1
kind: BenchmarkTask
metadata:
 name: cis-kubernetes-benchmark
spec:
 benchmarkVersion: 'cis-kubernetes-auto'
---
apiVersion: securityinspector.alibabacloud.com/v1alpha1
kind: BenchmarkJob
metadata:
 name: cis-kubernetes-benchmark
spec:
 taskName: cis-kubernetes-benchmark
EOF
```

   以上命令中benchmarkVersion字段的可用值如下，您可以根据需要选取合适的值（推荐使用cis-kubernetes-auto）。

benchmarkVersion字段的值	含义	适用集群
cis-kubernetes-auto	由组件根据集群版本自动选择最合适的CIS Kubernetes基线标准进行扫描。	Kubernetes 1.15及以上版本
cis-kubernetes-ack-1.0	按照CIS Alibaba Cloud Container Service For Kubernetes（ACK）Benchmark v1.0.0基线标准进行扫描。	ACK专有版集群和托管版集群（Kubernetes 1.18及以上版本）
cis-kubernetes-1.20	按照CIS Kubernetes V1.20 Benchmark v1.0.0基线标准进行扫描。	Kubernetes 1.20及以上版本
cis-kubernetes-1.6	按照CIS Kubernetes Benchmark v1.6.0基线标准进行扫描。	Kubernetes 1.16~1.19版本
cis-kubernetes-1.5	按照CIS Kubernetes Benchmark v1.5.1基线标准进行扫描。	Kubernetes 1.15版本

2. 等待5分钟左右，通过以下命令确认本次扫描任务是否完成。

```
kubectl get benchmarkjobs.securityinspector.alibabacloud.com cis-kubernetes-benchmark -o 'jsonpath={.status.phase}'; echo
```

命令执行后返回 `Succeeded` 即表示扫描完成。

3. 确认扫描完成后，可以通过以下命令获取本次扫描结果的CSV格式报告。

```
for name in $(kubectl get benchmarkcsvresults.securityinspector.alibabacloud.com -l securityinspector.task.name=cis-kubernetes-benchmark -o name)
do
 filename="cis-$(echo $name | awk -F '/' '{print $2}')"; \
 kubectl get $name -o jsonpath='{.result.data}' > "$filename".csv; \
 echo "saved $filename.csv"
done
```

获取到报告后，对于报告中的结果您可以根据实际业务场景来判断是否需要采取措施，报告中各列的含义可以参考下文。

报告解读

上文扫描获取报告中的各列的说明如下表。

列名	含义	是否需要采取措施
Date	扫描时间	否
Result Schema	扫描时遵照的基线标准。取值如下： • cis-kubernetes-ack-1.0：CIS Alibaba Cloud Container Service For Kubernetes (ACK) Benchmark v1.0.0基线标准。 • cis-kubernetes-1.20：CIS Kubernetes V1.20 Benchmark v1.0.0基线标准。 • cis-kubernetes-1.6：CIS Kubernetes Benchmark v1.6.0基线标准。 • cis-kubernetes-1.5：CIS Kubernetes Benchmark v1.5.1基线标准。 关于各个基线标准的具体内容，请参见CIS Kubernetes基线标准文档。	否
Node Name	当前报告是哪个集群节点的扫描报告。	否
Total Fail	计分检查项未通过数量。	关于说明信息，请参见下方Result列的说明。
Total Warn	需要关注的不计分检查项数量。	关于说明信息，请参见下方Result列的说明。
Total Pass	检查项通过数量。	否
Section Id	CIS基线标准中定义的章节编号。	否
Section Description	CIS基线标准中定义的章节简述。	否
Test Id	CIS基线标准中定义的检查项ID。	否

列名	含义	是否需要采取措施
Test Description	CIS基线标准中定义的检查项简述。	否
Scored	检查项是否计分。取值如下： • Scored：计分。 • Not scored：不计分。	否
Test Remediation	检查项失败时推荐的调整/修复方法。 更多信息，请参见CIS Kubernetes基线标准文档。	关于说明信息，请参见下方Result列的说明。
Result	检查项结果。取值如下： • fail：计分检查项未通过。 • warn：检查项不计分需要关注。 • pass：检查通过。	各个值可采取的措施如下： • fail：强烈建议根据Test Remediation列的推荐方法进行调整或修复。您也可以根据您的实际业务场景来判断是否调整或修复。 • warn：保持关注，根据实际业务场景进行调整。 • pass：不需要采取措施。

报告解读

上文扫描获取报告中的各列的说明如下表。

列名	含义	是否需要采取措施
Date	扫描时间	否
Result Schema	扫描时遵照的基线标准。取值如下： • cis-kubernetes-ack-1.0：CIS Alibaba Cloud Container Service For Kubernetes (ACK) Benchmark v1.0.0基线标准。 • cis-kubernetes-1.20：CIS Kubernetes V1.20 Benchmark v1.0.0基线标准。 • cis-kubernetes-1.6：CIS Kubernetes Benchmark v1.6.0基线标准。 • cis-kubernetes-1.5：CIS Kubernetes Benchmark v1.5.1基线标准。 关于各个基线标准的具体内容，请参见CIS Kubernetes基线标准文档。	否
Node Name	当前报告是哪个集群节点的扫描报告。	否
Total Fail	计分检查项未通过数量。	关于说明信息，请参见下方Result列的说明。
Total Warn	需要关注的不计分检查项数量。	关于说明信息，请参见下方Result列的说明。
Total Pass	检查项通过数量。	否
Section Id	CIS基线标准中定义的章节编号。	否

列名	含义	是否需要采取措施
Section Description	CIS基线标准中定义的章节简述。	否
Test Id	CIS基线标准中定义的检查项ID。	否
Test Description	CIS基线标准中定义的检查项简述。	否
Scored	检查项是否计分。取值如下： • Scored：计分。 • Not scored：不计分。	否
Test Remediation	检查项失败时推荐的调整/修复方法。 更多信息，请参见CIS Kubernetes基线标准文档。	关于说明信息，请参见下方Result列的说明。
Result	检查项结果。取值如下： • fail：计分检查项未通过。 • warn：检查项不计分需要关注。 • pass：检查通过。	各个值可采取的措施如下： • fail：强烈建议根据Test Remediation列的推荐方法进行调整或修复。您也可以根据您的实际业务场景来判断是否调整或修复。 • warn：保持关注，根据实际业务场景进行调整。 • pass：不需要采取措施。

相关文档

- CIS Kubernetes基线
- security-inspector

25.9.3. 使用配置巡检功能检查注册集群Workload安全隐患

本文将介绍如何使用配置巡检功能来扫描集群中Workload配置的安全隐患和介绍巡检报告相关信息，帮助您实时了解当前状态下运行应用的配置是否有安全隐患。

前提条件

- 您已通过容器服务Kubernetes版接入一个注册的Kubernetes集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
- 您已安装Logtail组件。具体操作，请参见步骤二：安装logtail-ds组件。

子账号（即RAM用户）RAM授权

如果您当前使用的是子账号，请先完成ACK容器服务配置巡检页面的RAM授权操作，确保当前子账号拥有操作当前集群配置巡检页面的权限，否则会出现权限不足无法操作配置巡检页面功能的问题。具体操作，请参见自定义RAM授权策略。

```
{
 "Statement": [
  {
   "Action": [
    "cs:DescribePolarisConfig",
    "cs:DescribePolarisJob",
    "cs:DescribePolarisCronJob",
    "cs:UpdatePolarisJob",
    "cs:UpdatePolarisCronJob"
   ],
   "Effect": "Allow",
   "Resource": [
    "acs:cs:*:*:cluster/<yourclusterID>"
   ]
  }
 ],
 "Version": "1"
}
```

如果您当前使用的是子账号，且需要使用巡检报告功能，请先完成日志服务指定 logproject （当前集群logtail-ds组件所使用的 logproject ）的RAM授权，确保当前子账号拥有日志服务指定 logproject 的数据读取权限，否则会出现权限不足无法查看巡检报告的问题。具体操作，请参见RAM自定义授权场景。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:Get*",
        "log:List*"
      ],
      "Resource": "acs:log:*:*:project/<指定的Project名称>/*",
      "Effect": "Allow"
    }
  ]
}
```

扫描集群Workload

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏选择**安全管理 > 配置巡检**。
5. 如果您未安装巡检组件的话，在**安装组件**下方单击**开始安装**。如果您已经安装请忽略此步骤。
6. 在**配置巡检**页面右上方，单击**立即执行巡检**。
7. 等待巡检扫描完成后，单击刷新图标查看相应的巡检报告。
8. （可选）在**配置巡检**页面右上方，单击**巡检配置**，开启或禁用定期自动执行配置巡检的功能以及配置巡检时执行的检查项。

巡检详情

巡检详情页面通过表格化的方式展示不同工作负载的详细巡检结果，主要包括以下功能：

- 直观展示工作负载的巡检**通过项**和**风险项**数量。

- 在检查结果页面展示巡检结果的各个检查项的通过状态、检查项详细说明以及加固建议。
- 支持为工作负载设置检查项白名单。
- 支持**命名空间**、**工作负载类别**、**是否有风险**等条件对结果进行过滤查询。

巡检报告

巡检报告中主要展示最近执行的巡检扫描结果，主要包括以下几类信息：

- 扫描结果概览，包括总的检查条目数、检查各个资源项数目及百分比、整体的健康度。
- 各大类扫描结果统计，包括报告健康检查、镜像、网络、资源、安全等大类的结果。
- 各个Workload配置的详细扫描结果，包括资源类别、资源名称、命名空间、检查类型、检查项、检查结果等内容。

配置巡检功能会扫描并展示以下检查项的扫描结果。

检查项名称	检查的内容及安全风险	修复建议

hostNetworkSet	通过检查Workload的Pod Spec中是否配置了hostNetwork：true，检查是否配置了共享使用主机的网络namespace。如果配置了，存在Pod中容器攻击主机网络、嗅探主机网络数据的风险。	修改Pod Spec，删除hostNetwork字段。
		示例：
		`@@ -14,7 +14,6 @@ spec:`
		`labels:`
		`app: nginx`
		`spec:`
		`- hostNetwork: true`
		`containers:`
		`- name: nginx`
		`image: nginx:1.14.2`

hostIPCSet	通过检查Workload的Pod Spec中是否配置了hostIPC：true，检查是否配置了共享使用主机的IPC namespace。如果配置了，存在Pod中容器攻击主机上进程、嗅探主机上进程数据的风险。	修改Pod Spec，删除hostIPC字段。
		示例：
		`@@ -14,7 +14,6 @@ spec:`
		`labels:`
		`app: nginx`
		`spec:`
		`- hostIPC: true`
		`containers:`
		`- name: nginx`
		`image: nginx:1.14.2`

hostPIDSet	通过检查Workload的Pod Spec中是否配置了hostPID：true，检查是否配置了共享使用主机的PID namespace。如果配置了，存在Pod中容器攻击主机上进程、采集主机上进程数据的风险。	修改Pod Spec，删除hostPID字段。
		示例：
		`@@ -14,7 +14,6 @@ spec:`
		`labels:`
		`app: nginx`
		`spec:`
		`- hostPID: true`
		`containers:`
		`- name: nginx`
		`image: nginx:1.14.2`

检查项名称	检查的内容及安全风险	修复建议

hostPortSet	通过检查Workload的Pod Spec中是否配置了hostPort，检查是否配置了把容器中监听的端口映射到主机指定端口上。如果配置了，存在挤占主机可用端口以及被非预期的请求方请求容器端口的风险。	修改Pod Spec，删除hostPort字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`-  hostPort: 80`

runAsRootAllowed	通过检查Workload的Pod Spec中是否未配置runAsNonRoot：true，检查是否未配置使用非root用户运行容器。如果未配置，存在被容器中的恶意进程入侵用户应用、入侵主机甚至入侵整个集群的风险。	修改Pod Spec，增加runAsNonRoot：true。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`+  securityContext:`
		`+  runAsNonRoot: true`

runAsPrivileged	通过检查Workload的Pod Spec中是否配置了privileged：true，检查是否配置了允许以特权模式运行容器。如果配置了，存在被容器中的恶意进程入侵用户应用、入侵主机甚至入侵集群的风险。	修改Pod Spec，删除privileged字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`-  privileged: true`

privilegeEscalationAllowed	通过检查Workload的Pod Spec中是否未配置allowPrivilegeEscalation：false，检查是否未配置禁止容器中的子进程拥有比父进程更高的权限。如果未配置，存在被容器中的恶意进程实现越权操作的风险。	修改Pod Spec，增加allowPrivilegeEscalation：false字段。
		示例：
		`ports:`
		`- containerPort: 80`
		`securityContext:`
		`+  allowPrivilegeEscalation: false`
		`  readOnlyRootFilesystem: true`
		`  runAsNonRoot: true`
		`volumeMounts:`

检查项名称	检查的内容及安全风险	修复建议

capabilitiesAdded	通过检查Workload的Pod Spec中capabilities字段，检查是否配置了允许容器中的进程拥有SYS_ADMIN、NET_ADMIN、ALL等特权Linux Capabilities。如果配置了，存在被容器中的恶意进程通过这些特权入侵用户应用、入侵或破坏组件/集群的风险。	修改Pod Spec，根据实际需求只添加必需的Linux Capabilities，删除不需要的Linux Capabilities。
		不依赖额外Linux Capabilities，删除所有不需要的Linux Capabilities。示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`  capabilities:`
		`- add:`
		`-  - SYS_ADMIN`
		`-  - NET_ADMIN`
		`-  - KILL`
		`+ drop:`
		`+  - ALL`
		只添加必需的Linux Capabilities，删除所有不需要的Linux Capabilities。示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`  capabilities:`
		`  add:`
		`-  - SYS_ADMIN`
		`-  - NET_ADMIN`
		`-  - KILL`
		`+  - CHOWN`
		`+ drop:`
		`+  - ALL`

检查项名称	检查的内容及安全风险	修复建议

notReadOnlyRootFileSystem	通过检查Workload的Pod Spec中是否未配置readOnlyRootFilesystem：true，检查是否未配置容器中的文件系统是不可修改的。如果未配置的话，存在被容器中的恶意进程恶意修改系统文件的风险。	修改Pod Spec，增加readOnlyRootFilesystem: true，如果有需要修改某个目录下文件的需求，可以通过volumeMounts实现。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`+  readOnlyRootFilesystem: true`
		如果有需要修改某个目录下文件的需求，通过volumeMounts实现修改文件的需求。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  ports:`
		`  - containerPort: 80`
		`  securityContext:`
		`+  readOnlyRootFilesystem: true`
		`  runAsNonRoot: true`
		`+  volumeMounts:`
		`+  - mountPath: /path/to/write`
		`+  name: writeable`
		`+  volumes:`
		`+  - emptyDir: {}`
		`+  name: writeable`

cpuRequestsMissing	通过检查Workload的Pod Spec中是否未配置resources.requests.cpu字段，检查是否未配置运行容器所需的最少CPU资源。如果未配置的话，Pod有被调度到资源紧张的节点上的风险，可能会出现容器内进程运行缓慢的风险。	修改Pod Spec，增加resources.requests.cpu字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`+  resources:`
		`+  requests:`
		`+  cpu: 100m`

cpuLimitsMissing	通过检查Workload的Pod Spec中是否未配置resources.limits.cpu字段，检查是否未配置运行容器所需的最大CPU资源。如果未配置的话，存在被容器内的异常进程消耗大量节点资源，甚至把整个节点或集群的资源销耗殆尽的风险。	修改Pod Spec，增加resources.limits.cpu字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  resources:`
		`  requests:`
		`  cpu: 100m`
		`+  limits:`
		`+  cpu: 100m`

检查项名称	检查的内容及安全风险	修复建议

memoryRequestsMissing	通过检查Workload的Pod Spec中是否未配置resources.requests.memory字段，检查是否未配置运行容器所需的最少内存资源。如果未配置，Pod有被调度到资源紧张的节点上的风险，可能会出现容器内进程OOM的风险。	修改Pod Spec，增加resources.requests.memory字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  resources:`
		`  requests:`
		`  cpu: 100m`
		`+ memory: 128Mi`
		`  limits:`

memoryLimitsMissing	通过检查Workload的Pod Spec中是否未配置resources.limits.memory字段，检查是否未配置运行容器所需的最大内存资源。如果未配置的话，存在被容器内的异常进程消耗大量节点资源，甚至把整个节点或集群的资源销耗殆尽的风险。	修改Pod Spec，增加resources.limits.memory字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`  resources:`
		`  requests:`
		`  cpu: 100m`
		`  memory: 128Mi`
		`  limits:`
		`  cpu: 100m`
		`+ memory: 128Mi`
		`  ports:`

readinessProbeMissing	通过检查Workload的Pod Spec中是否未配置readinessProbe字段，检查是否未配置检测容器内应用能否正常处理请求的探针。如果未配置的话，存在在容器内应用异常无法处理请求时仍旧有请求发过来导致业务异常的风险。	修改Pod Spec，增加readinessProbe字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`+ readinessProbe:`
		`+ httpGet:`
		`+ path: /health`
		`+ port: 8080`
		`+ initialDelaySeconds: 5`
		`+ periodSeconds: 20`
		`  resources:`

livenessProbeMissing	通过检查Workload的Pod Spec中是否未配置livenessProbe，检查是否未配置检测容器内应用是否出现异常需要重启容器的探针。如果未配置，存在在容器内应用异常需要重启容器才能恢复时未及时重启导致业务异常的风险。	修改Pod Spec，增加livenessProbe字段。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`  image: nginx:1.14.2`
		`+ livenessProbe:`
		`+ httpGet:`
		`+ path: /health`
		`+ port: 8080`
		`+ initialDelaySeconds: 5`
		`+ periodSeconds: 20`
		`  readinessProbe:`

tagNotSpecified	通过检查Workload的Pod Spec中的image字段的值是否未包含镜像Tag或者使用了latest作为镜像Tag，检查是否未配置运行容器时使用指定Tag的容器镜像。如果未配置，存在在运行容器时运行了非预期的容器镜像版本导致业务异常的风险。	修改Pod Spec，修改image字段，使用指定的镜像Tag并且不要使用latest作为镜像Tag。
		示例：
		`spec:`
		`  containers:`
		`  - name: nginx`
		`- image: nginx:latest`
		`+ image: nginx:1.14.2`
		`  livenessProbe:`

检查项名称	检查的内容及安全风险	修复建议

anonymousUserRBACBinding	通过检查集群内的RBAC（Role-based access control）绑定找出配置了匿名用户访问权限的配置项。如果配置了允许匿名用户访问集群资源的配置项，则存在被恶意匿名用户窃取集群敏感信息、攻击和入侵集群的风险。	修改扫描出来的RBAC绑定，根据实际情况删除允许匿名用户访问集群资源的权限配置项。
		示例：
		`- apiGroup: rbac.authorization.k8s.io`
		`kind: Group`
		`name: 'foo-group'`
		`- apiGroup: rbac.authorization.k8s.io`
		`kind: User`
		`name: 'system:anonymous'`
		`- apiGroup: rbac.authorization.k8s.io`
		`kind: Group`
		`name: 'system:unauthenticated'`

相关文档

- Configure a Security Context for a Pod or Container
- Configure Liveness, Readiness and Startup Probes

25.10. 注册集群FAQ

本文介绍在使用注册集群时遇到的常见问题的解决方法。

注册集群是否收费？

注册集群是免费产品，但涉及到部分关联云资源会产生费用。更多信息，请参见注册集群的关联云资源或云服务。

本地数据中心的Kubernetes集群可以扩容云上弹性资源吗？

可以。具体操作，请参见创建混合集群。

接入注册集群的目标集群对于网络连通性有什么要求？

ACK注册集群提供公网和内网接入端点。关于如何通过绑定EIP开通公网访问能力，请参见控制集群API Server的公网访问能力。

在以下场景必须使用内网接入：

- 使用注册集群节点池功能为目标集群扩容阿里云ECS或弹性裸金属服务器。具体操作，请参见创建混合集群。
- 使用注册集群托管ack-virtual-node组件为目标集群部署扩容弹性ECI实例。具体操作，请参见扩容弹性容器实例ECI。

请确保目标集群网络可以正常访问注册集群接入端点的5533端口。关于云上云下网络专线接通方案，请参见方案介绍。

其他集群管理、安全治理、日志监控告警等功能对内网连通性无要求，但需确保目标集群可访问公网。

26.多集群管理

26.1. 应用中心

26.1.1. 应用中心概述

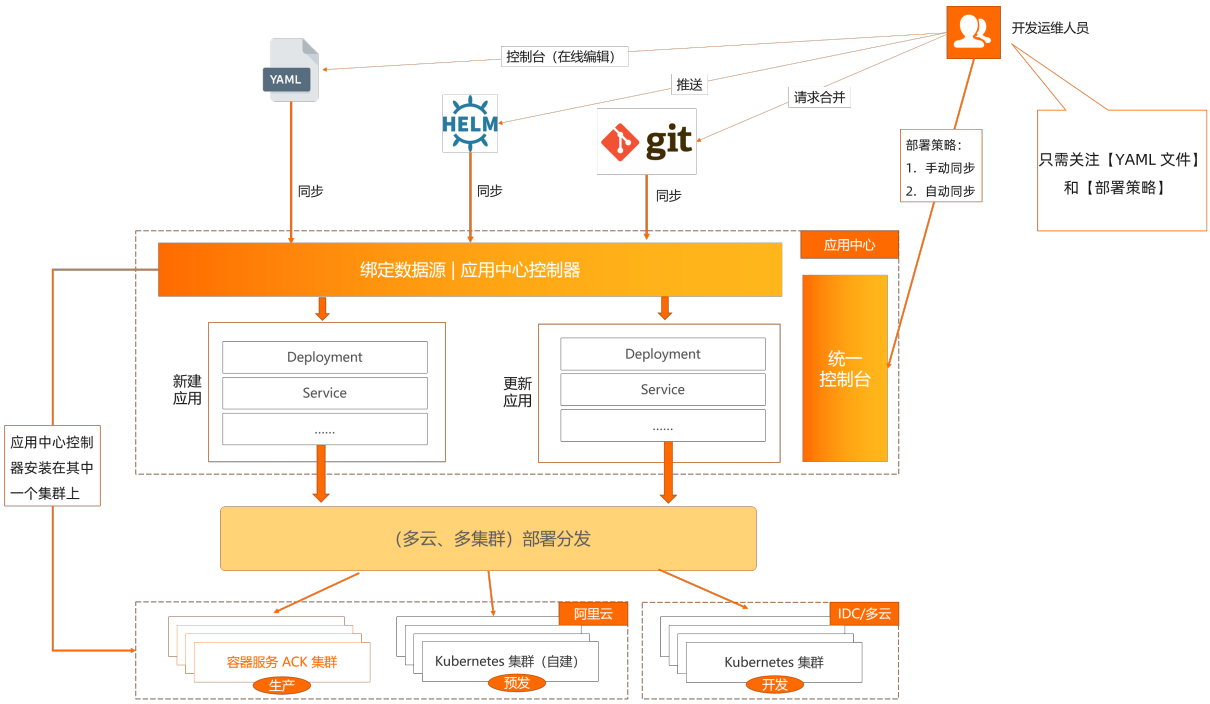
您可以通过ACK控制台的**应用中心**功能部署应用。本文介绍**应用中心**的作用、核心概念、核心组件等。

背景信息

使用编排模板中的工具部署应用完毕后，应用资源比较分散，您无法从一个统一的视角查看应用所有的Kubernetes资源。在Kubernetes平台中，一个应用可能包含多个Kubernetes子资源。例如一个Wordpress应用，会包含Deployment、Service、Ingress、ServiceAccount、Roles等资源。在应用部署完毕后，没有一个统一的视角展现整体应用的拓扑结构，同时对于持续部署等场景无法做到统一的版本管理与回滚。

应用中心能够给应用一个统一的入口视图，帮助您全局地了解应用的部署形态。应用中心能够帮助您提供一个应用层级的视角，您能够在应用级别观察所有Kubernetes子资源的部署状态与变化，同时以Git、Helm Chart作为载体，将应用以版本化的方式部署到Kubernetes集群中，且可以在不同版本之间实现回滚与发布。

使用场景



ACK应用中心的使用场景如下：

- 多集群低成本容灾，日常迭代版本一键同发。
- 国内站、国际站多地域部署业务，版本同发。
- 同套应用，线上线下混合云发布。
- 应用维度查看和管理工作负载。
- 同一套编排模板，多环境发布，可以大幅降低出错率。

核心概念

概念	描述
应用	应用编排模板定义的一组Kubernetes资源集合。
目标状态	应用程序的期望状态由Git或者Chart等存储库中的文件表示。
当前状态	该应用程序的实时状态，例如部署的所有Pod状态等。
部署状态	实时状态是否与目标状态匹配。部署的应用程序是否与Git或者Helm Chart所定义的相同。
部署	使应用程序移至其目标状态的过程。例如，通过**kubectl apply**将应用部署到Kubernetes集群。
刷新	将Git中的最新代码与实时状态进行比较。
健康状态	应用程序的运行状况是否正常运行。

核心组件

应用中心的所有组件都是部署在客户集群**appcenter**命名空间内，主要有四个核心部分，如下表所示。

组件名称	描述
application-controller	负责应用的版本记录、回溯，指定版本回滚等功能。
redis	负责缓存应用数据资源，记录一些用户应用缓存数据信息。
repo-server	负责拉取远程仓库应用部署模板信息，例如Git或者Helm Chart。
server	暴露一个GRPC Server，负责与外部通信。

相关文档

- 管理应用
- 使用编排模板创建应用
- 使用Git源仓库创建应用

26.1.2. 应用管理

26.1.2.1. 使用编排模板创建应用

通过应用中心创建应用可以直观地查看所有Kubernetes资源的部署状态与变化。根据数据源的来源不同，创建应用的方式也不同。本文介绍如何使用编排模板在应用中心创建应用。

前提条件

已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

操作步骤

1. 登录容器服务管理控制台。
2. 为Template资源授权。
   i. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   ii. 在**集群信息**页面上方单击**集群资源**页签。

iii. 在**集群资源**页签单击**Worker RAM角色**右侧的**KubernetesWorkerRole-******。



iv. 在RAM访问控制台的**KubernetesWorkerRole-******页面，单击**权限管理**页签，然后单击**k8sWorkerRolePolicy-******权限策略名称。

v. 在**权限策略**页面，单击**策略内容**页签，然后单击**修改策略内容**。

vi. 在**修改策略内容**面板添加Template相关权限后，单击**确定**。

```
{
  "Action": [
    "cs:CreateTemplate",
    "cs:DescribeTemplates",
    "cs:DescribeTemplateAttribute",
    "cs:UpdateTemplate",
    "cs:DeleteTemplate"
  ],
  "Resource": [
    "*"
  ],
  "Effect": "Allow"
}
```

3. 在容器服务管理控制台左上角集群名称左侧单击 〈 图标。

4. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。

5. 在**应用中心**页面单击**创建应用**。

6. 在**应用来源**配置向导页面配置参数。

i. 在**安装应用发布组件**区域选择集群，然后单击**开始安装**。

> **说明** 仅首次安装应用需要执行此步骤。

ii. 在**应用来源**区域单击**YAML模板**。

iii. 单击**下一步**。

7. 在**应用配置**配置向导页面中设置参数。

i. 设置**应用名称**和**应用描述**。

ii. 设置模板，您可以选择已有模板或者新建模板， 本文以新建模板为例。

a. 单击**新建模板**页签，然后单击右上角的**通过YAML添加**。

b. 将以下内容复制到文本框中，然后单击**确定**。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo
 labels:
  app: demo
spec:
 minReadySeconds: 5
 revisionHistoryLimit: 5
 progressDeadlineSeconds: 60
 strategy:
  rollingUpdate:
   maxUnavailable: 1
  type: RollingUpdate
 selector:
  matchLabels:
   app: demo
 template:
  metadata:
   annotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "9797"
   labels:
    app: demo
  spec:
   containers:
   - name: demo
    image: registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:red
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
     containerPort: 8080
     protocol: TCP
    readinessProbe:
     tcpSocket:
      port: 8080
     initialDelaySeconds: 5
     timeoutSeconds: 5
    resources:
     limits:
      cpu: 2000m
      memory: 512Mi
     requests:
```

```
      requests:
        cpu: 100m
        memory: 64Mi
---
apiVersion: v1
kind: Service
metadata:
 name: demo-svc
spec:
 selector:
  app: demo
 ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: demo
 labels:
  app: demo
spec:
 rules:
  - host: app.demo.example.com
    http:
     paths:
     - path: /
       pathType: Prefix
       backend:
        service:
         name: demo-svc
         port:
          number: 80
```

iii. 单击**下一步**。

8. 在**发布配置**配置向导页面中设置参数。

   i. 在**发布配置**区域选择发布方式，可选**全量发布**和**灰度发布**。

   > **说明** 第一次发布应用时只能选择**全量发布**，后续对应用进行更新迭代时，可以选择**全量发布**或**灰度发布**。

   ii. 在**可选集群**区域单击集群左侧的**添加**，在**目标集群**区域可以看到选择的集群，然后在**目标集群**区域选择应用要发布的**命名空间**。

   > **说明** 如果没有开通公网访问的集群，需要为API Server绑定EIP，以开启公网访问能力。具体操作，请参见控制集群API Server的公网访问能力。

   iii. 单击**开始发布**。

   iv. 在**发布确认**对话框单击**确定**。

   在**发布日志**配置向导中可以看到发布应用的进度。

9. 在**发布日志**配置向导页面中单击**完成**。

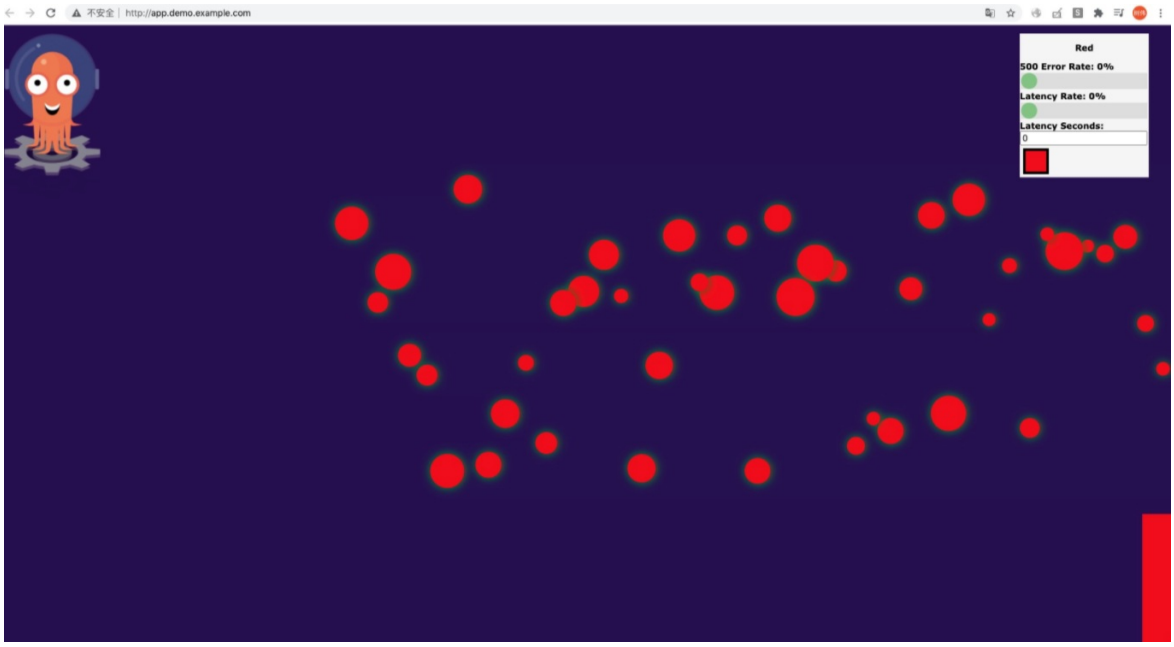
结果验证

1. 获取应用的端点。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
   v. 在**路由**页面查看demo的端点。
2. 在*hosts*文件输入以下内容。

   ```
   <端点> app.demo.example.com
   ```

3. 在浏览器中访问*app.demo.example.com*。

   浏览器显示以下页面，说明应用部署成功。



相关文档

- 应用中心概述
- 管理应用
- 使用Git源仓库创建应用

26.1.2.2. 使用Git源仓库创建应用

通过ACK的应用中心创建应用可以直观地查看所有Kubernetes资源的部署状态与变化。根据数据源的来源不同，创建应用的方式也不同。本文介绍如何使用Git源仓库在应用中心创建应用。

前提条件

已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群。

示例仓库地址

- GitHub仓库地址
- 阿里云Code仓库地址

Git源仓库中应用编排方式

- Kubernetes原生的Manifests编排

```
.
├── deployment.yaml
├── ingress.yaml
└── service.yaml
```

- Helm编排

```
.
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
├── values-pro.yaml
└── values.yaml
```

- Kustomize编排

```
.
├── base
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── pre
    │   ├── deployment.yaml
    │   └── kustomization.yaml
    └── pro
        ├── deployment.yaml
        └── kustomization.yaml
```

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击**创建应用**。
4. 在**应用来源**配置向导配置参数。
   i. 在**安装应用发布组件**区域选择集群，然后单击**开始安装**。

      > **说明** 仅首次安装应用需要执行此步骤。

   ii. 在**应用来源**区域单击**Git仓库**。
   iii. 单击**下一步**。
5. 在**应用配置**配置向导中设置参数，然后单击**下一步**。

参数	说明
应用名称	设置应用名称。

参数	说明
应用描述	设置应用描述。
仓库	选择仓库，您可以选择已创建的仓库或者使用新仓库。本文以使用新仓库为例。
仓库地址	设置仓库的地址。
名称	设置仓库的名称。
连接方式	选择连接方式，支持**HTTPS**和**SSH**。以下为不同连接方式需要设置的参数： ○ 当设置连接方式为**HTTPS**，需要设置以下参数。 ■ **用户名（可选）**：如果是私有Git仓库，需要设置用户名。 ■ **密码（可选）**：如果是私有Git仓库，需要设置密码。 ■ **TLS客户端证书（可选）**：如果Git服务器使用的是自签发证书，需要设置TLS客户端证书。 ■ **TLS客户端证书密钥（可选）**：如果Git服务器使用的是自签发证书，需要设置TLS客户端证书密钥。 ■ **跳过服务端证书校验**：如果Git服务器使用的是自签发证书，且想要省略TLS客户端证书和TLS客户端证书密钥的配置，则可以选中**跳过服务端证书校验**。 ■ **LFS支持**：是否需要配置Git启用LFS支持。 ○ 当设置连接方式为**SSH**，需要设置以下参数。 ■ **SSH私钥**：设置SSH私钥。 ■ **跳过服务端证书校验**：如果Git服务器使用的是自签发证书，且想要省略SSH私钥的配置，则可以选中**跳过服务端证书校验**。 ■ **LFS支持**：是否需要配置Git启用LFS支持。
分支	分支名称。
路径	Git源仓库中的应用编排所在的子目录。

6. 在**发布配置**配置向导页面中设置参数。
   i. 在**发布配置**区域选择发布方式，可选**全量发布**和**灰度发布**。

      > **说明** 第一次发布应用时只能选择**全量发布**，后续对应用进行更新迭代时，可以选择**全量发布**或**灰度发布**。

   ii. 在**可选集群**区域单击集群左侧的**添加**后，在**目标集群**区域可以看到选择的集群，然后在**目标集群**区域选择应用要发布的**命名空间**。

      > **说明** 如果没有开通公网访问的集群，需要为API Server绑定EIP，以开启公网访问能力。

   iii. 单击**开始发布**。
7. 在**发布确认**对话框单击**确定**。

   在**发布日志**配置页面中可以看到发布应用的进度。

相关文档

- 应用中心概述

- 管理应用
- 使用编排模板创建应用

26.1.2.3. 管理应用

通过ACK的应用中心创建应用可以直观地查看所有Kubernetes资源的部署状态与变化。创建应用后，您还可以查看、更新和删除应用。本文介绍如何查看、更新和删除应用。

查看应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击目标应用名称左侧的图标，可以查看部署应用的**集群**，部署应用的**命名空间**、**应用健康状态**、**部署状态**和**已部署版本**。

更新应用

> **说明** 更新应用前，请先更新数据源：
> - 如果您是将Git作为数据源，请确保已将新构建的应用推送至Git。
> - 如果您是将编排模板作为数据源，请确保已更新编排模板。具体操作，请参见更新编排模板。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击目标应用名称右侧**操作**列的**创建变更**。
4. 在**部署状态**区域可以看到**部署状态**显示**未部署最新版本**，单击**下一步**。

   > **说明** 在**部署状态**区域单击**已部署版本**下的**与最新版本比较**，可以查看本次更新版本与旧版本之间差异。

5. 在**发布配置**配置向导中设置参数。
   i. 在**发布配置**区域选择发布方式，可选**全量发布**和**灰度发布**。
   ii. 在**可选集群**区域单击集群左侧的**添加**，在**目标集群**区域可以看到选择的集群，然后在**目标集群**区域选择应用要发布的**命名空间**。
   iii. 单击**开始发布**。
6. 在**发布确认**对话框单击**确定**。

   在**发布日志**配置向导中可以看到发布应用的进度。

回滚应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击目标应用的名称。
4. 在应用详情页面右上角单击**历史版本**。

5. 在**历史版本**面板单击目标版本右侧的**回滚到该版本**，然后在弹出的对话框单击**确定**。

删除应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击目标应用右侧**操作**列的**删除**。
4. 在**删除应用demo**对话框中单击**确定**。

相关文档

- 使用编排模板创建应用
- 使用Git源仓库创建应用

26.1.3. 触发器

26.1.3.1. 触发器概述

应用中心触发器是为您应用生成的一个Webhook，能够帮助您完成远程触发应用部署或重新部署操作。

前提条件

应用中心触发器支持已有应用的更新和重新部署。在使用应用中心触发器前，您需要先在应用中心创建应用。

应用场景

应用中心触发器可以用于以下两种场景：

- 远程触发应用中心管理下的Kubernetes应用重新部署。
- 通过触发器自动化串联已有的持续集成系统（例如容器镜像服务交付链、Jenkins、GitLab CI等）和应用中心系统，形成完整的CICD Pipeline。

注意事项

请您妥善保管触发器信息，避免泄露；如果不慎泄露，请及时注销或重建触发器。

相关文档

- 使用触发器自动更新应用
- 使用Jenkins持续集成自动更新应用
- 使用容器镜像触发器自动更新应用

26.1.3.2. 使用触发器自动更新应用

应用中心触发器可以帮助您完成远程触发应用部署或重新部署操作。本文介绍如何使用触发器自动更新应用。

前提条件

已创建应用，具体操作，请参见使用编排模板创建应用。

背景信息

本文以提前创建的Red版本的Demo应用为例，将应用的镜像由Red版本修改为Blue版本，然后通过触发器更新应用。最后通过浏览器访问应用，验证使用触发器自动更新应用是否成功。如果应用页面由红色变为蓝色，说明应用镜像由Red版本更新为Blue版本，使用触发器自动更新应用成功。

更新编排模板

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。

3. 在**模板列表**页面单击demo应用的编排模板下的**详情**。
4. 将模板中的镜像地址改为 `registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:blue` ，然后单击**保存**。

更新应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击目标应用的**操作**列的**详情**。
4. 在应用详情页面单击**触发器**。
5. 在**触发器**面板单击**创建**，然后单击**复制**，复制触发器URL。
6. 执行以下命令，触发应用自动部署。

   ```
   curl <触发器URL>
   ```

   预期输出：

   ```
   {"code":"200","message":"","requestId":"c811998e-3dda-405c-9564-68726342****"}
   ```

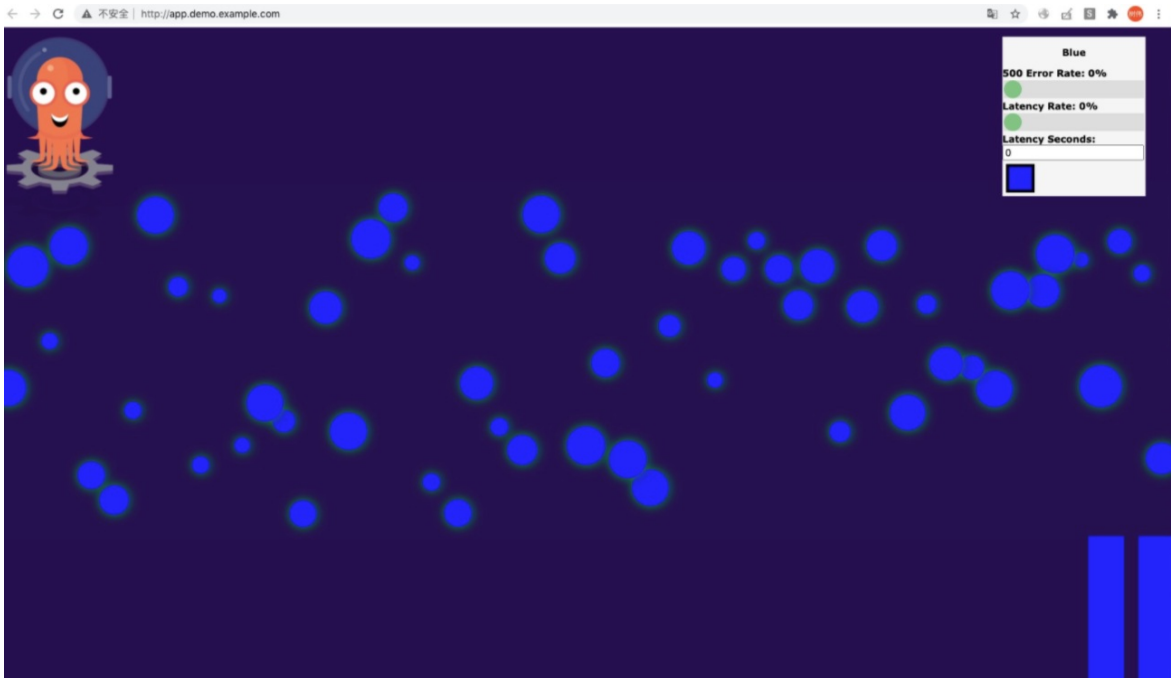
结果验证

1. 获取应用的端点。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
   v. 在**路由**页面查看demo的端点。
2. 在*hosts*文件输入以下内容。

   ```
   <端点> app.demo.example.com
   ```

3. 在浏览器中访问*app.demo.example.com*。

浏览器显示以下页面，说明应用已由Red版本更新为Blue版本。



相关文档

- 触发器概述
- 使用Jenkins持续集成自动更新应用
- 使用容器镜像触发器自动更新应用

26.1.3.3. 使用Jenkins持续集成自动更新应用

应用中心触发器支持更新ACK编排模板源（仅限编排模板）中的容器镜像版本，并触发应用重新部署。您可以使用Jenkins持续集成构建并更新编排模板中的容器镜像版本，然后使用触发器自动更新应用。本文介绍如何使用Jenkins CI自动更新应用。

前提条件

- 已创建应用，并生成触发器。具体操作，请参见使用编排模板创建应用和使用触发器自动更新应用。
- 已部署Jenkins系统。具体操作，请参见快速搭建Jenkins环境并完成流水线作业。

背景信息

本文以提前创建的Red版本的demo应用为例，使用Jenkins将应用的镜像由Red版本修改为Green版本，然后通过触发器更新应用。最后通过浏览器访问应用，验证使用触发器自动更新应用是否成功。如果应用页面由红色变为绿色，说明应用镜像由Red版本更新为Green版本，使用Jenkins持续集成自动更新应用成功。

操作步骤

1. 创建Jenkins构建任务并构建和推送 `registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green` 镜像。具体操作，请参见快速搭建Jenkins环境并完成流水线作业。
2. 执行以下脚本，触发应用中心使用 `registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green` 镜像，并重新部署应用。

```
## 4. trigger appcenter
DEPLOYMENT_NAME=demo
CONTAINER_NAME=demo
IMAGE=registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green
cat <<EOF > patch.json
{
  "resource":{
    "deployment":{
      "metadata":{
        "name":"$DEPLOYMENT_NAME"
      },
      "spec": {
        "template": {
          "spec": {
            "containers": [
              {
                "name":"$CONTAINER_NAME",
                "image":"$IMAGE"
              }]
          }
        }
      }
    }
  }
}
EOF
curl -X POST -H 'content-type: application/json' \
  https://cs.console.aliyun.com/hook/trigger?token=xxxxxxxx \
  -d "$(cat patch.json)"
{"code":"200","message":"","requestId":"151c92fd-d97b-4eff-a6da-4669114fa4f0"}
```

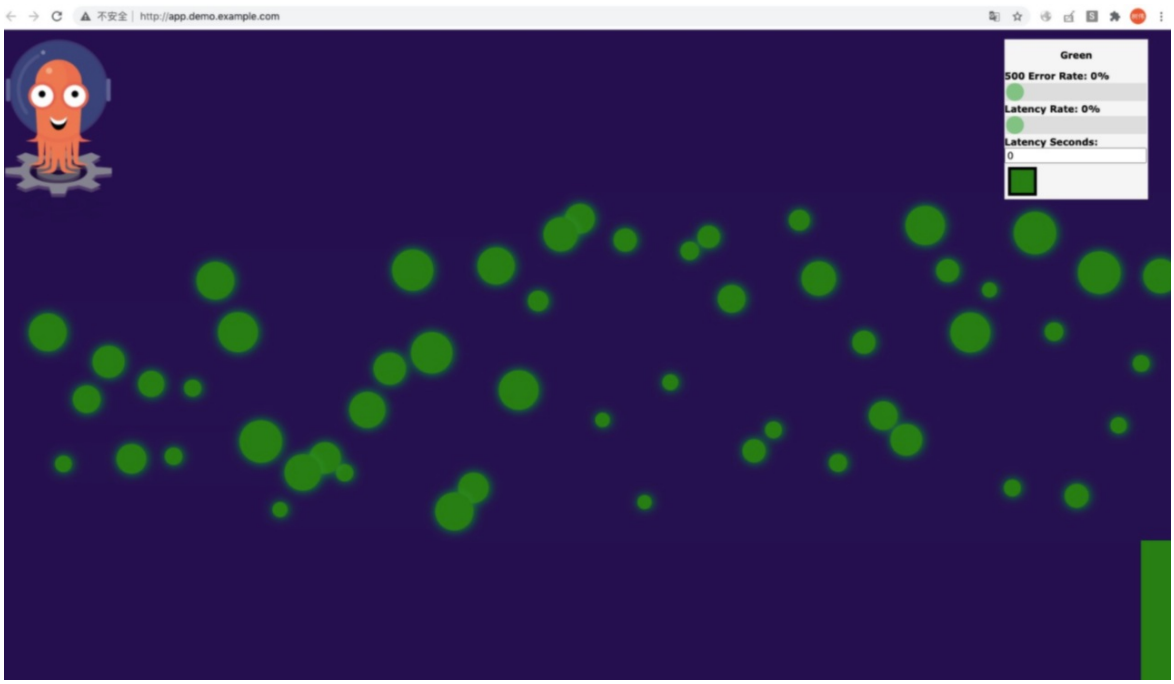
结果验证

1. 获取应用的端点。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv.
   v. 在**路由**页面查看demo的端点。
2. 在*hosts*文件输入以下内容。

   ```
   <端点> app.demo.example.com
   ```

3. 在浏览器中访问*app.demo.example.com*。

   浏览器显示以下页面，说明应用已由Red版本更新为Green版本。



相关文档

- 触发器概述
- 使用触发器自动更新应用
- 使用容器镜像触发器自动更新应用

26.1.3.4. 使用容器镜像触发器自动更新应用

容器镜像服务ACR为每个仓库提供了触发器的功能，帮助您在镜像构建成功后进行消息的推送。当镜像仓库的镜像版本有更新时，将根据设置的触发条件自动触发使用最新镜像部署应用。本文将演示如何在镜像构建成功后，自动触发容器服务上应用中心拉取新的镜像，并重新部署应用。

前提条件

已将应用镜像推送至容器镜像仓库。具体操作，请参见使用企业版实例推送和拉取镜像。

步骤一：创建应用

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击**创建应用**。
4. 在**应用来源**配置向导页面配置参数。
   i. 在**安装应用发布组件**区域选择集群，然后单击**开始安装**。

      > **说明** 仅首次安装应用需要执行此步骤。

   ii. 在**应用来源**区域单击**YAML模板**。
   iii. 单击**下一步**。
5. 在**应用配置**配置向导页面中设置参数。
   i. 设置**应用名称**和**应用描述**。
   ii. 单击**新建模板**页签，然后单击右上角的**通过YAML添加**。
   iii. 将以下内容复制到文本框中，然后单击**确定**。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 creationTimestamp: null
 labels:
  app: demo
 name: demo
spec:
 minReadySeconds: 5
 progressDeadlineSeconds: 60
 revisionHistoryLimit: 5
 selector:
  matchLabels:
   app: demo
 strategy:
  rollingUpdate:
   maxUnavailable: 1
  type: RollingUpdate
 template:
  metadata:
   annotations:
    prometheus.io/port: "9797"
    prometheus.io/scrape: "true"
   creationTimestamp: null
   labels:
    app: demo
  spec:
   containers:
   - image: liusheng-registry.cn-beijing.cr.aliyuncs.com/liusheng/rollouts-demo:blue
    imagePullPolicy: IfNotPresent
    name: demo
    ports:
    - containerPort: 8080
     name: http
     protocol: TCP
    readinessProbe:
     initialDelaySeconds: 5
     tcpSocket:
      port: 8080
     timeoutSeconds: 5
    resources:
     limits:
      cpu: "2"
      memory: 512Mi
     requests:
      cpu: 100m
      memory: 64Mi
status: {}
---
apiVersion: v1
kind: Service
metadata:
 name: demo-svc
spec:
 selector:
  app: demo
 ports:
```

```
  - protocol: TCP
    port: 80
    targetPort: 8080
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: demo
 labels:
  app: demo
spec:
 rules:
  - host: app.demo.example.com
    http:
     paths:
      - backend:
        serviceName: demo-svc
        servicePort: 80
---
```

关于部分配置参数的说明如下所示：

参数	说明
minReadySeconds	设置等待时间。集群在设置的等待时间后才进行升级。
progressDeadlineSeconds	定义升级过程中的最大时间。如果升级过程被暂停，该时间也会同步暂停，时间不会一直增长。
revisionHistoryLimit	定义升级最大记录数。
rollingUpdate.maxUnavailable	滚动更新期间最大多少个Pod不可用。
image	使用容器镜像仓库中的镜像地址，本文以Blue版本的demo应用镜像为例。

iv. 单击**下一步**。

6. 在**发布配置**配置向导页面中设置参数。

i. 在**发布配置**区域选择发布方式，可选**全量发布**和**灰度发布**。

> **说明** 第一次发布应用时只能选择**全量发布**，后续对应用进行更新迭代时，可以选择**全量发布**或**灰度发布**。

ii. 在**可选集群**区域单击集群左侧的**添加**后，在**目标集群**区域可以看到选择的集群，然后在**目标集群**区域选择应用要发布的**命名空间**。

> **说明** 如果没有开通公网访问的集群，需要为API Server绑定EIP，以开启公网访问能力。

iii. 单击**开始发布**。

7. 在**发布确认**对话框单击**确定**。

在**发布日志**配置页面中可以看到发布应用的进度。

8. 在**发布日志**配置向导页面中单击**完成**。

步骤二：在应用中心生成触发器

1. 登录容器服务管理控制台。

2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击步骤一创建的应用名称。
4. 在应用详情页面右上角单击**触发器**。
5. 在**触发器**面板单击**创建**，然后单击**复制**，复制触发器URL。

> **说明** 请妥善保管触发器URL，以免泄露。

步骤三：在ACR创建触发器

1. 登录容器镜像服务控制台。
2. 在顶部菜单栏，选择所需地域。
3. 在左侧导航栏，选择**实例列表**。
4. 在**实例列表**页面单击目标企业版实例。
5. 在企业版实例管理页面选择**仓库管理 > 镜像仓库**。
6. 在**镜像仓库**页面单击步骤一创建应用时所使用的镜像仓库的名称。
7. 在镜像仓库详情页面单击**触发器**，然后单击**创建**。
8. 在**创建触发器**对话框设置触发器名称，输入步骤二复制的触发器URL，设置**触发方式**为**全部触发**，单击**确定**。

   容器镜像触发器支持**全部触发**、**表达式触发**和**Tag触发**。以下为3种触发方式介绍：

   - 全部触发：只要镜像版本有更新，就会触发触发器。
   - 表达式触发：基于正则表达式来进行镜像版本的过滤，只有当符合正则表达式的镜像版本才会触发触发器。
   - Tag触发：基于镜像版本列表来进行镜像版本的过滤，只有镜像版本在设置的镜像版本列表中才会触发触发器。

> **说明** 如果触发器执行异常，您可以在**触发器**页面单击目标触发器操作列的**访问记录**，在**访问记录**对话框Response Body中查看异常原因。

步骤四：触发应用部署

当镜像仓库的镜像版本有更新时，会自动触发使用最新镜像部署应用。

1. 执行以下命令，登录步骤一创建应用时所使用的镜像仓库。

> **说明** 如果您企业版实例所在地域为杭州，则命令行中<企业版实例所在地域>为hangzhou。

```
docker login --username=<镜像仓库登录名> <企业版实例名称>-registry.cn-<企业版实例所在的地域>.cr.aliyuncs.com
```

2. 执行以下命令，给镜像设置标签。

```
docker tag registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green liusheng-registry.cn-beijing.cr.aliyuncs.com/liusheng/rollouts-demo:green
```

3. 执行以下命令，推送Green版本镜像至步骤一创建应用时所使用的镜像仓库。

```
docker push liusheng-registry.cn-beijing.cr.aliyuncs.com/liusheng/rollouts-demo:green
```

结果验证

1. 查看编排模板。

镜像更新后，自动触发容器服务上应用中心拉取新的镜像并更新编排模板。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏选择**市场 > 编排模板**。

iii. 在**模板列表**页面单击步骤一创建的模板名称下的**详情**。在**详情**页面可以看到模板镜像已更新为最新的镜像。

2. 获取应用的端点。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv.

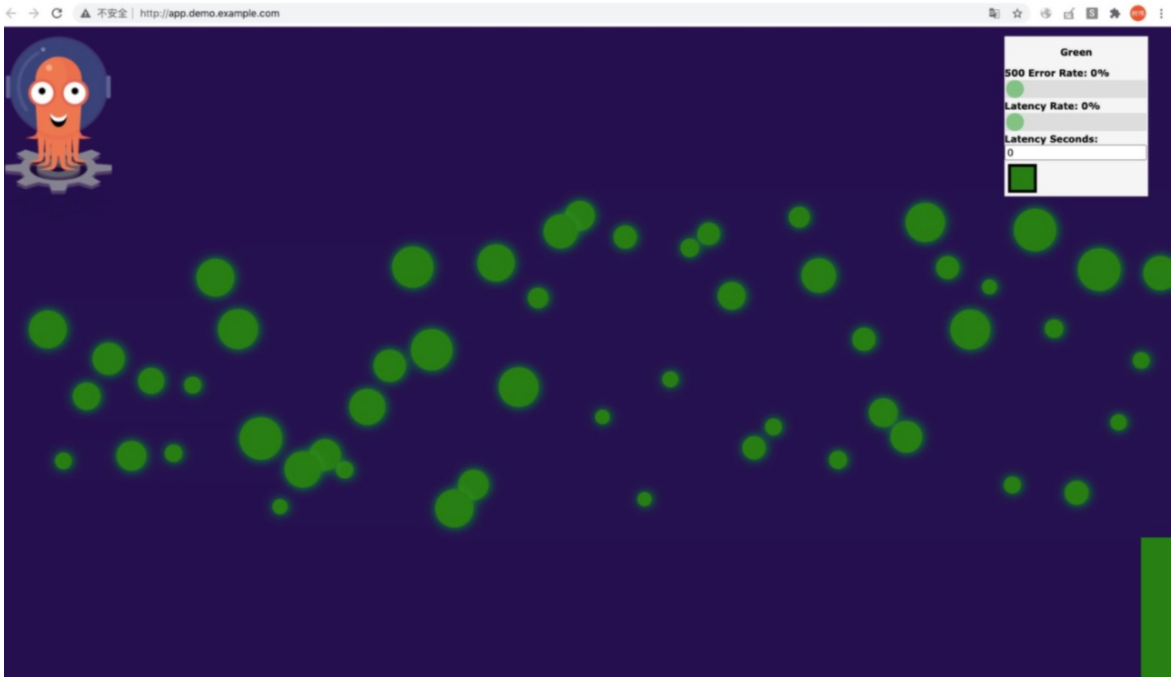
v. 在**路由**页面查看demo的端点。

3. 在*hosts*文件输入以下内容。

```
<端点> app.demo.example.com
```

4. 在浏览器中访问*app.demo.example.com*。

浏览器显示以下页面，说明应用已由Blue版本更新为Green版本。



相关文档

- 触发器概述
- 使用触发器自动更新应用
- 使用Jenkins持续集成自动更新应用

26.1.4. 多集群发布

26.1.4.1. 使用编排模板部署应用到多集群

当您的账号下有多个集群分布在不同的地域，您可以使用应用中心的多集群发布功能将一个应用同时发布到多个集群里。本文介绍如何使用编排模板将一个应用一键部署到2个不同地域的Kubernetes集群。

前提条件

已创建2个不同地域集群，且集群具有访问公网的能力。具体操作，请参见创建Kubernetes托管版集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击**创建应用**。
4. 在**应用来源**配置向导页面配置参数。
   i. 在**安装应用发布组件**区域选择集群，然后单击**开始安装**。

   > **说明** 仅首次安装应用需要执行此步骤。

   ii. 在**应用来源**区域单击**YAML模板**。
   iii. 单击**下一步**。
5. 在**应用配置**配置向导页面中设置参数。
   i. 设置**应用名称**和**应用描述**。
   ii. 设置模板，您可以选择已有模板或者新建模板， 本文以新建模板为例。
      a. 单击**新建模板**页签，然后单击右上角的**通过YAML添加**。
      b. 将以下内容复制到文本框中，然后单击**确定**。
      本文以Red版本的demo应用为例。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo
 labels:
  app: demo
spec:
 minReadySeconds: 5
 revisionHistoryLimit: 5
 progressDeadlineSeconds: 60
 strategy:
  rollingUpdate:
   maxUnavailable: 1
  type: RollingUpdate
 selector:
  matchLabels:
   app: demo
 template:
  metadata:
   annotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "9797"
   labels:
    app: demo
  spec:
   containers:
   - name: demo
    image: registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:red
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
```

```
      containerPort: 8080
      protocol: TCP
     readinessProbe:
      tcpSocket:
       port: 8080
      initialDelaySeconds: 5
      timeoutSeconds: 5
     resources:
      limits:
       cpu: 2000m
       memory: 512Mi
      requests:
       cpu: 100m
       memory: 64Mi
---
apiVersion: v1
kind: Service
metadata:
 name: demo-svc
spec:
 selector:
  app: demo
 ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: demo
 labels:
  app: demo
spec:
 rules:
  - host: app.demo.example.com
    http:
     paths:
      - backend:
         serviceName: demo-svc
         servicePort: 80
```

iii. 单击**下一步**。

6. 在**发布配置**配置向导页面中设置参数。

   i. 在**发布配置**区域选择发布方式，可选**全量发布**和**灰度发布**。

      第一次发布应用时只能选择全量发布，后续对应用进行更新迭代时，可以选择全量发布或灰度发布。

   ii. 在**可选集群**区域单击集群左侧的**添加**，选择2个不同地域的集群。在**目标集群**区域可以看到选择的集群，然后选择应用要发布的**命名空间**。

   > **说明** 在**可选集群**区域显示当前账号下所有地域的集群。

   iii. 单击**开始发布**。

7. 在**发布确认**对话框单击**确定**。

   在**发布日志**配置页面中可以看到发布应用的进度。

8. 在**发布日志**配置向导页面中单击**完成**。

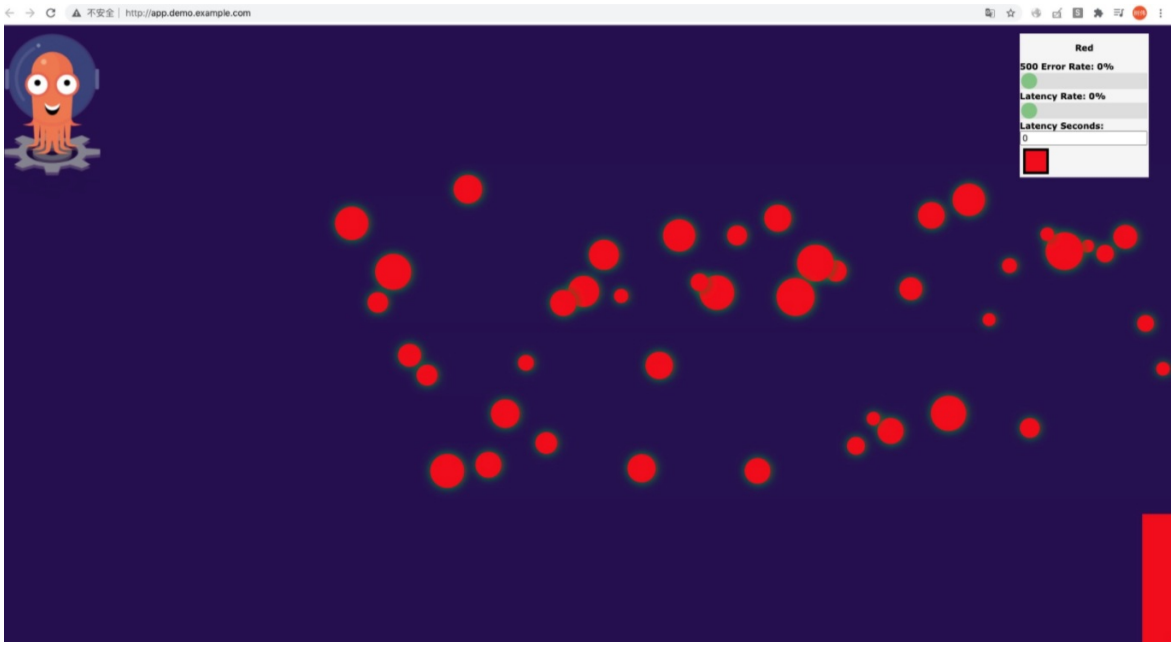
结果验证

1. 获取应用的端点。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv.
   v. 在**路由**页面查看demo的端点。
2. 在*hosts*文件输入以下内容。

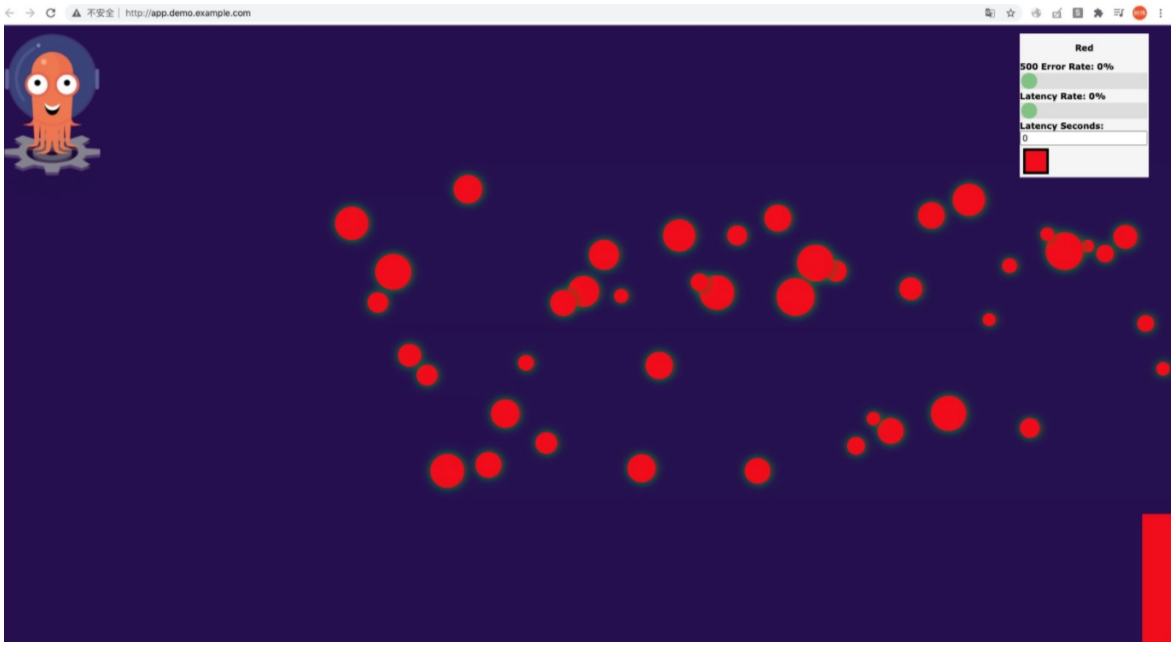
   ```
   <端点> app.demo.example.com
   ```

3. 在浏览器中访问*app.demo.example.com*。

   浏览器显示以下页面，说明应用部署成功。



4. 在*hosts*文件删除步骤1绑定的地址，重复步骤2至步骤3，访问另一个集群部署的应用，显示以下页面，说明应用部署成功。



相关文档

- 多地域集群应用灰度发布

26.1.4.2. 多地域集群应用灰度发布

容器服务ACK支持将一个应用发布到不同地域集群，并按照地域进行灰度发布。本文介绍编排模板更新后，A集群保持原版本应用不变，B集群根据编排模板对应用进行更新。

前提条件

已创建2个不同地域的集群，且集群具有访问公网的能力。具体操作，请参见创建Kubernetes托管版集群。

背景信息

本文以北京和杭州地域的集群为例，设置北京地域的集群为A集群，杭州地域的集群为B集群。在2个地域集群都部署Red版本的Demo应用，A集群保持Red版本的Demo应用不变，B集群根据编排模板更新为Green版本的Demo应用。

步骤一：部署应用到集群

本文以Red版本的Demo应用为例，将同一个应用部署到2个不同地域集群。具体操作，请参见使用编排模板部署应用到多集群。

步骤二：更新应用的编排模板。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 编排模板**。
3. 在**模板列表**页面单击Demo应用的目标模板下的**详情**。
4. 在模板详情页面修改模板信息，将镜像地址改为 `registry.cn-hangzhou.aliyuncs.com/acs/rollouts-demo:green` ，然后单击**创建**。

步骤三：设置应用灰度发布

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用中心**页面单击Demo应用**操作**列的**创建变更**。
4. 在**创建变更**配置向导页面中单击**下一步**。

5. 在**发布配置**配置向导页面中**目标集群**区域选中B集群，单击**开始发布**。
6. 在**发布确认**对话框中单击**确定**。

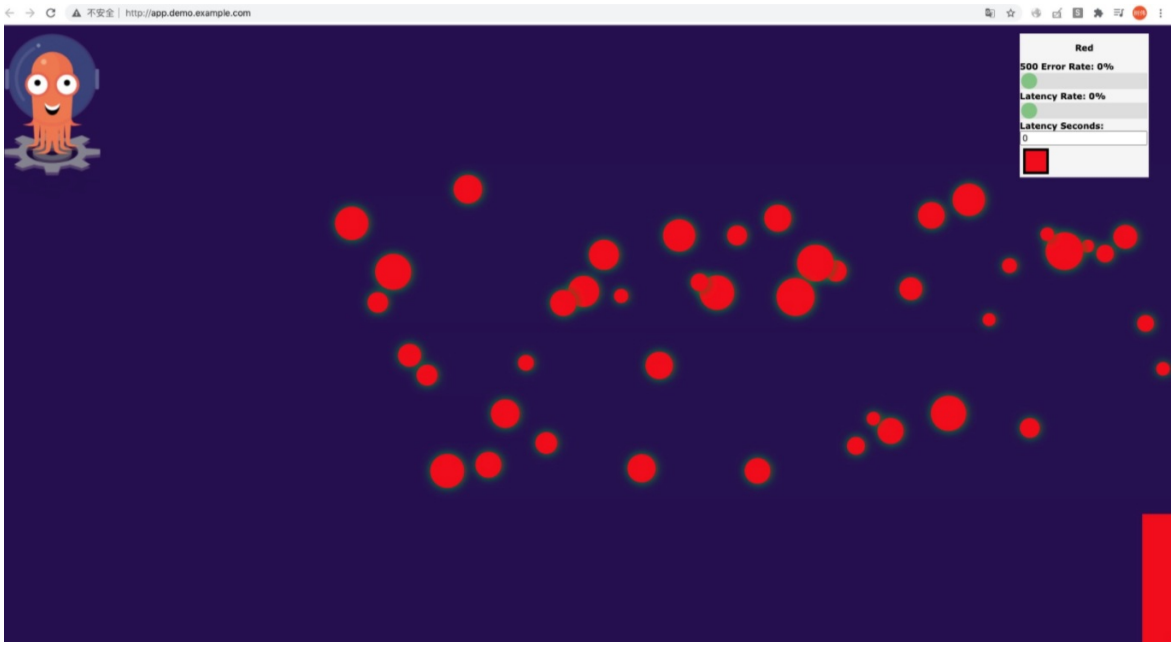
结果验证

1. 获取应用的端点。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击A集群名称或者A集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 路由**。
   v. 在**路由**页面查看Demo的端点。
2. 在*hosts*文件输入以下内容。

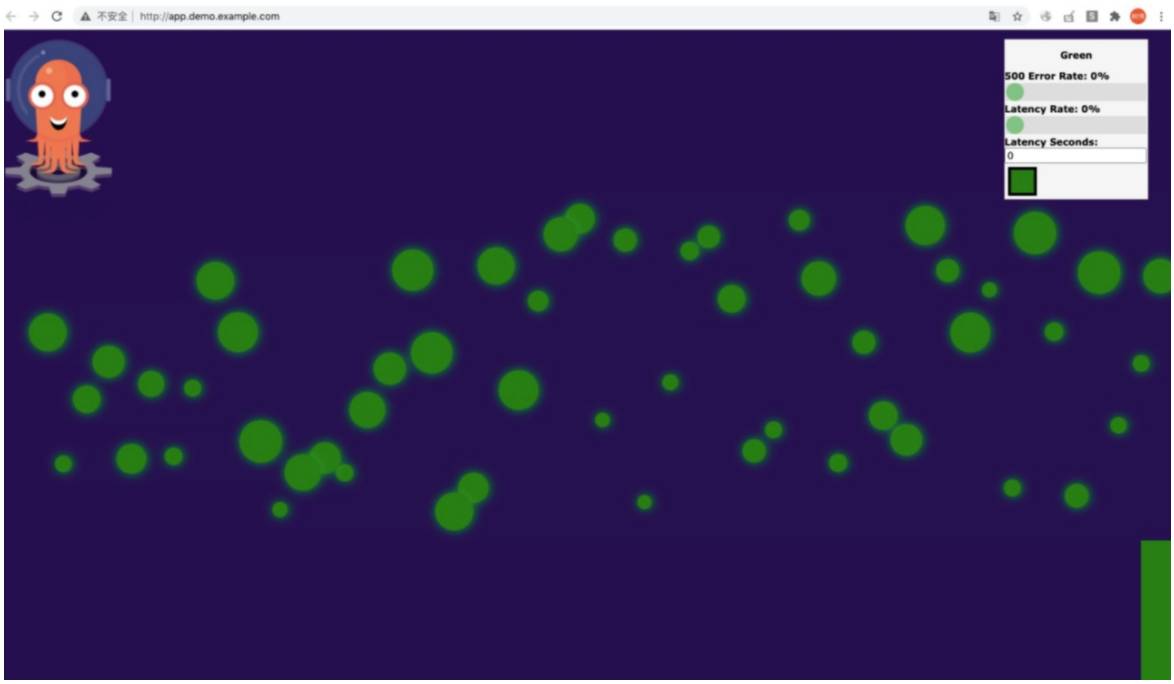
   ```
   <端点> app.demo.example.com
   ```

3. 在浏览器中访问*app.demo.example.com*。

   浏览器显示以下页面。

4. 在*hosts*文件删除步骤2绑定的地址，重复步骤1至步骤3，访问B集群部署的应用，显示以下页面。



可以看到，A集群部署的是Red版本的应用，B集群部署的是Green版本的应用，说明多地域集群应用灰度发布成功。

相关文档

- 使用编排模板部署应用到多集群

26.2. 备份中心

26.2.1. 备份中心概述

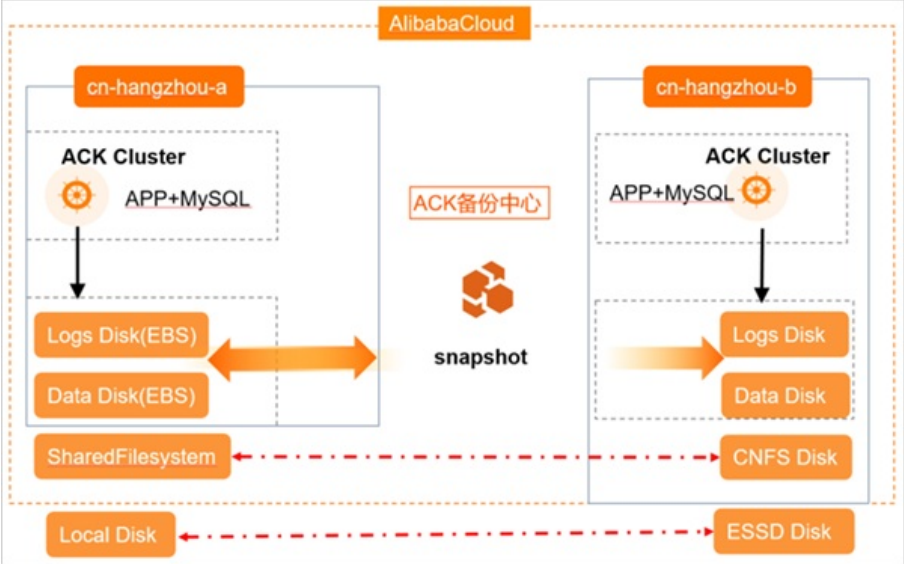
阿里云容器服务ACK备份中心为无状态或有状态应用的备份、恢复与迁移提供了一站式的解决方案，特别是对混合云，多集群的有状态应用提供了数据容灾和应用迁移能力。例如，集群应用的元数据备份、应用和数据的统一快照恢复以及应用和数据的迁移。通过容器编排和应用管理的CRD能力，实现有状态应用的崩溃一致性备份、有状态应用一致性备份及跨可用区、跨地域的备份和恢复。本文主要介绍备份中心的使用场景、核心概念及核心组件。

使用场景

ACK备份中心的使用场景如下：

- ACK集群内持久化应用容灾和同地域集群间迁移
- 混合云容灾和应用迁移
- 公共云跨地域集群之间的容灾

ACK集群内持久化应用容灾和同地域集群间迁移



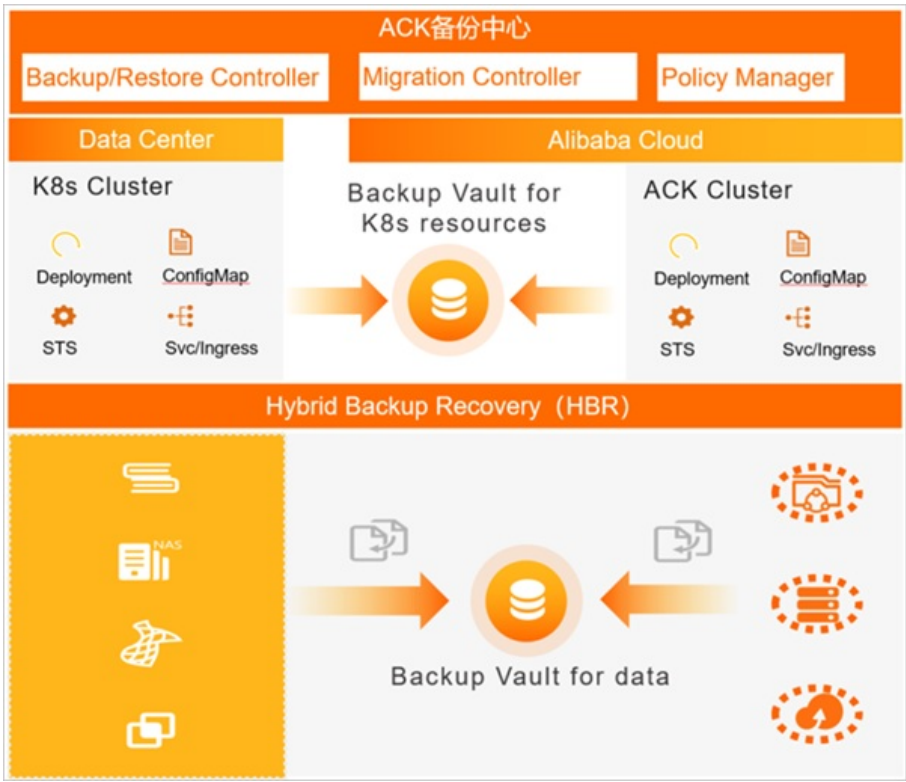
支持在云上ACK集群内及同地域ACK集群之间进行应用与数据的备份恢复。您可以在源集群完成备份操作，在目标集群制作快照，并在目标集群实现RTO秒级的应用和数据恢复。针对云上数据，备份中心提供了以下功能：

- 按时间点的全量极速快照备份
- 定时增量备份能力
- TB级云盘的极速快照
- 应用一致性的备份和恢复

支持的云上存储类型为：

- 云上块存储云盘、本地盘等。
- 云上NAS、CPFS等。
- 云上对象存储OSS。

混合云容灾和应用迁移



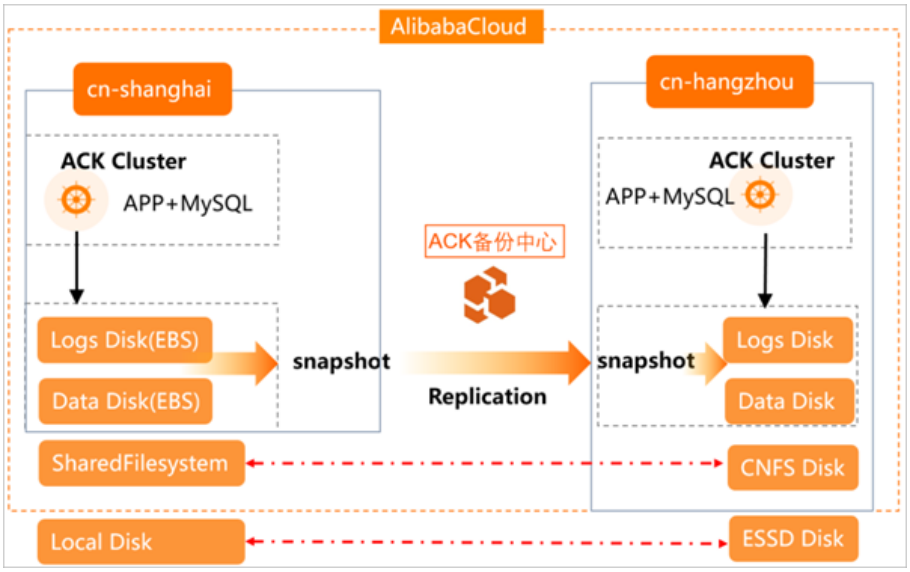
支持云下数据中心自建K8s集群，通过注册集群方式注册为混合云集群后，可以允许备份中心对应用和数据备份和定时备份至云端。您可以在目标集群（例如，混合云集群或云上的ACK集群）恢复整个应用和数据。针对数据，备份中心的混合云灾备份能力提供了以下功能：

- 按时间点的全量快照备份
- 定时增量备份能力
- 最小化数据增量、去重复及压缩

支持的云下存储类型为：

- 数据中心本地块存储SAN、Ceph OBD及本地盘LVM等。
- 数据中心文件系统HostPath、NFS、NetApp、GlusterFS及CephFS等。
- 数据中心本地对象存储Ceph S3等。

公共云跨地域集群之间的容灾



支持在云上同地域不同可用区ACK集群之间进行应用与数据的备份恢复。您可以在源集群完成备份操作，在目标集群制作快照，并在目标集群实现RTO秒级的应用和数据恢复。针对云上数据，备份中心提供了以下功能：

- 按时间点的全量极速快照备份
- 定时增量备份能力
- TB级云盘的极速快照
- 应用一致性的备份和恢复

支持的云上存储类型为：

- 云上块存储云盘、本地盘等。
- 云上NAS、CPFS等。
- 云上对象存储OSS。

核心概念

概念	描述
崩溃一致性备份	不保证应用缓存和操作系统缓存完全备份在存储介质中。
应用一致性备份	保证应用缓存及操作系统缓存备份在存储介质。
跨可用区	同地域跨可用区备份与恢复。
跨地域	跨地域备份与恢复。
快照	文件系统快照，云盘块存储快照。
极速快照	TB级云盘的秒级快照制作。
恢复	应用和数据的统一还原。
应用备份	应用、服务及配置的元数据备份。

核心组件

备份中心的所有组件都是部署在客户集群csdr命名空间内，主要有三个核心部分，如下表所示：

组件名称	描述
csdr-controller	负责整体应用模板备份、PV数据备份及恢复的流程、策略、状态控制等功能。
csdr-velero	负责应用模板的筛选、备份、恢复等功能。
hbr-client	负责应用所依赖的文件系统级别PV数据的备份及恢复。

相关文档

- 关于如何开启集群备份，请参见开启集群备份。
- 开启集群备份后，关于如何创建备份和恢复应用的任务，请参见备份和恢复应用。
- 关于如何跨集群迁移应用，请参见跨集群迁移应用。

26.2.2. 开启集群备份

阿里云容器服务备份中心为无状态或有状态应用的备份、恢复与迁移提供了一站式的解决方案，特别是对混合云，多集群的有状态应用提供了数据容灾和应用迁移能力。本文介绍如何开启集群备份能力及专有版集群与注册集群的相关授权配置。

前提条件

- 若是在混合云部署应用，您需要开通HBR服务。具体操作，请参见混合云备份HBR。
- 备份中心功能处于公测期。使用该功能前，您需要提交工单申请白名单。
- 已创建Kubernetes集群。具体操作，请参见创建Kubernetes托管版集群、创建Kubernetes专有版集群或创建注册集群并接入本地数据中心集群。

  > **注意** 请确保您的Kubernetes集群版本大于1.10，否则影响该功能的使用。

- 已通过kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

随着越来越多的应用运行在Kubernetes中，对应用进行定时的备份就显得格外重要。备份中心可以有效的防护由于意外情况导致服务长时间中断而无法恢复的情况。区别于传统的备份单机、备份磁盘的方案，基于Kubernetes的应用备份，主要关注于运行在Kubernetes中的应用及其数据、资源对象、配置及整个命名空间等。

步骤一：安装备份服务组件

> **说明** 首次使用备份中心功能，您需安装备份服务组件；如果您已安装，则请忽略此步骤。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏选择**运维管理 > 应用备份（公测）**。
5. 在**应用备份**页面单击**开始安装**。

   > **说明** 备份服务组件安装完之后，会生成csdr的命名空间，在使用备份过程中，请勿删除该命名空间。

   安装完成后，显示以下界面，说明备份服务组件安装成功。



（可选）步骤二：为专有版集群配置OSS权限

当前备份中心只使用阿里云OSS存储备份的应用。在使用阿里云对象存储OSS前，您需要配置OSS权限。

您需要为已有专有版集群配置OSS权限，新建集群无需进行以下操作。

1. 为OSS创建自定义权限策略。具体操作，请参见创建自定义策略。

   > **说明** 更多OSS细粒度授权配置信息，请参见通过RAM对OSS进行权限管理。

   如果您想要所有OSS的权限，请按以下示例设置权限策略：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "oss:PutObject",
        "oss:GetObject",
        "oss:DeleteObject",
        "oss:GetBucket",
        "oss:ListObjects",
        "oss:ListBuckets"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

   如果您只想要指定OSS的读写权限，请按以下示例设置权限策略：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "oss:PutObject",
        "oss:GetObject",
        "oss:DeleteObject",
        "oss:GetBucket",
        "oss:ListObjects",
        "oss:ListBuckets"
      ],
      "Resource": [
        "acs:oss:*:*:mybackups",
        "acs:oss:*:*:mybackups/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

您需要将上述代码中的 `mybackups` 替换为您自己的OSS名称。

2. 为集群RAM角色添加权限。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群信息页面单击**集群资源**页签，查看**Worker RAM角色**。
   v. 登录RAM控制台。
   vi. 在控制台左侧导航栏单击**授权**。
   vii. 在**授权**页面单击**新增授权**，在**添加权限**对话框配置参数，然后单击**确定**。

参数	说明
授权范围	可选**整个云账号**和**指定资源组**。
被授权主体	输入Worker RAM角色。
选择权限	单击**自定义策略**，在搜索框中搜索步骤中创建的权限策略，然后单击目标权限策略名称。

（可选）步骤三：配置注册集群权限

需要为注册集群创建RAM用户，为RAM用户添加访问云资源权限，然后创建AccessKey。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 为OSS创建自定义权限策略。具体操作，请参见步骤1。
3. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。
4. 为RAM用户创建AccessKey。具体操作，请参见获取AccessKey。
5. 在注册集群中创建Secret。

   为了保证您的AccessKey信息只在您的注册集群内安全使用，您需要先在注册集群中使用AccessKey信息部署一个名为**alibaba-addon-secret**的Secret资源，以降低泄露风险。

i. 执行以下命令，创建命名空间csdr。

```
kubectl create ns csdr
```

ii. 执行以下命令，创建名为alibaba-addon-secret的Secret资源。

```
kubectl -n csdr create secret generic alibaba-addon-secret --from-literal='access-key-id=<your AccessKey ID>' --from-literal='access-key-secret=<your AccessKey Secret>'
```

您需要将上述代码中*your AccessKey ID*和*your AccessKey Secret*替换为您在上一步获取的AccessKey信息。

（可选）步骤四：为专有版集群或者注册集群的用户配置HBR权限

如果您需要为专有版集群或者注册集群需要开通HBR权限。

1. 为HBR创建自定义权限策略。具体操作，请参见创建自定义策略。

   请按以下示例设置权限策略：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "hbr:CreateVault",
        "hbr:CreateBackupJob",
        "hbr:DescribeVaults",
        "hbr:DescribeBackupJobs2",
        "hbr:DescribeRestoreJobs",
        "hbr:SearchHistoricalSnapshots",
        "hbr:CreateRestoreJob",
        "hbr:AddContainerCluster",
        "hbr:DescribeContainerCluster",
        "hbr:DescribeRestoreJobs2"
      ],
      "Resource": "*"
    }
  ]
}
```

2. 为RAM用户添加权限。具体操作，请参见为RAM用户授权。

相关文档

- 备份和恢复应用
- 跨集群迁移应用

26.2.3. 备份和恢复应用

阿里云容器服务备份中心为集群内的有状态应用提供灾难备份和恢复能力，对于Kubernetes集群内的有状态应用的崩溃一致性、应用一致性及跨地域的灾难恢复提供了一站式的解决方案。本文介绍如何通过控制台应用备份功能实现应用的备份、制作快照及应用的恢复。

前提条件

已安装备份服务组件。具体操作，请参见开启集群备份。

使用限制

- 备份应用时，正在被删除的资源不会被备份。

- 如果您需要使用HBR文件备份功能，需要提交工单申请。
- 如果想使用云盘快照备份卷，需要安装CSI插件，且其版本不低于1.1.0。关于如何安装CSI插件，请参见安装与升级CSI组件。
- 如果需要创建NAS快照（即制作快照时对应PVC的**转换为存储类**列选择为**alibabacloud-cnfs-nas**），需要您先创建storageClass。具体操作，请参见使用CNFS托管NAS文件系统。

计费说明

若备份时，使用了云盘、NAS、HBR或快照等存储资源，计费说明如下所示：

- 关于云盘的计费说明，请参见块存储计费。
- 关于NAS的计费说明，请参见NAS计费说明。
- 关于HBR的计费说明，请参见计费方式与计费项。
- 关于快照的计费说明，请参见快照计费。

步骤一：创建备份仓库

当前应用备份功能使用阿里云OSS对备份数据进行存储。若创建备份任务时无可用的备份仓库，则需执行该步骤。一个集群只需要指定一次备份仓库。

> **说明** 在您集群的所在地域只需要创建一次备份仓库，即您可在同一个地域的不同集群使用同一个备份仓库。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏选择**运维管理 > 应用备份（公测）**。
5. 在**应用备份**页面**备份仓库**页签下单击**创建**。
6. 在**创建**面板配置参数，然后单击**确定**。

参数	说明
名称	备份仓库的名称，支持小写字母和数字。
Bucket区域	OSS Bucket所在地域。
Bucket名称	OSS Bucket的名称。若为ACK托管版集群，则OSS Bucket的名称需要以**cnfs-**开头。可以通过CNFS OSS声明一个新的Bucket
Bucket子目录	OSS Bucket的子目录（非必填）。

步骤二：创建备份任务

1. 在**应用备份**页面单击**备份计划**页签。
2. 在**备份计划**页签单击**创建**。
3. 在**创建**面板设置备份应用的相关参数，然后单击**确定**。



参数	描述
名称	设置备份任务的名称，必填项。
备份仓库	选择需要关联的备份仓库，必填项。
备份命名空间	支持选中一个或者多个备份命名空间。表示备份选中的一个或多个命名空间中的应用，必填项。
指定标签备份	填写需要备份的应用标签，以指定备份带有该标签的应用。
排除命名空间	支持选中一个或者多个排查命名空间。表示备份时排除选中的命名空间中的应用。
指定资源	支持填写多个资源对象名称，多个资源对象名称以英文逗号（,）间隔，例如 `deploy,configmap` 等。指定资源后可以只备份特定资源。
排除资源	支持填写多个资源对象名称，多个资源对象名称以英文逗号（,）间隔，例如 `pod,secret` 等。表示排除备份选中的资源对象应用。
备份存储卷	设置是否开启备份存储卷以及存储应用中的数据。 当选中**备份存储卷**时，默认为您进行数据备份。如果存储卷是云盘，则通过快照进行备份；如果存储卷不是云盘，则会通过HBR方式进行备份。

参数	描述
备份有效期	设置备份任务的数据存储有效期，过期之后数据将无法恢复。
备份类型	选择**立即备份**或**定时备份**。 当设置为**定时备份**时，需要配置备份周期。支持Crontab表达式设置定时备份。关于如何使用Crontab表达式，请参见how-use-cron-linux。

> **说明**
>
> - 备份任务名称仅支持小写字母和数字，不能有空格。
> - 备份命名空间及排除命名空间均可以选择多个。
> - 如果指定标签备份，每个备份任务只能输入一个标签。
> - 备份存储卷时，云盘快照集成了阿里云云盘快照技术，而HBR是阿里云混合云备份服务的简称，是一种高效、安全、低成本的全托管式云备份存储服务。
> - 定时备份的表达式，支持Linux Crontab以及按照间隔备份。
> - 如果开启了定时备份功能，则在**应用备份**页面的**备份计划**页签可以看到定时备份的详细配置信息。

在**备份和快照**页签下创建的备份任务状态显示**Completed**，表示该任务备份成功。

步骤三：为文件系统类型的数据卷制作快照

若创建备份任务时已选中**备份存储卷**且存储类型使用了非云盘类型，即使用混合云备份服务HBR（Hybrid Backup Recovery）进行文件备份。根据恢复需求，您可在恢复集群中选择制作云盘快照或文件系统CNFS快照。

> **注意** 若需要跨集群恢复应用，需要在恢复集群中执行以下步骤。

1. 在**应用备份**页面单击**备份和快照**页签。
2. 在对应备份任务右侧**操作**列单击**制作快照**。
3. 在**生成快照**对话框，选中对应可以转换的PVC，对应PVC的**转换为存储类**列选择对应存储卷类型，然后单击**制作快照**（对应显示为Snapshot的系统已经创建快照完成，无须再次转换）。



目前支持NAS（alibabacloud-cnfs-nas）及云盘（alicloud-disk）两种存储卷类型。

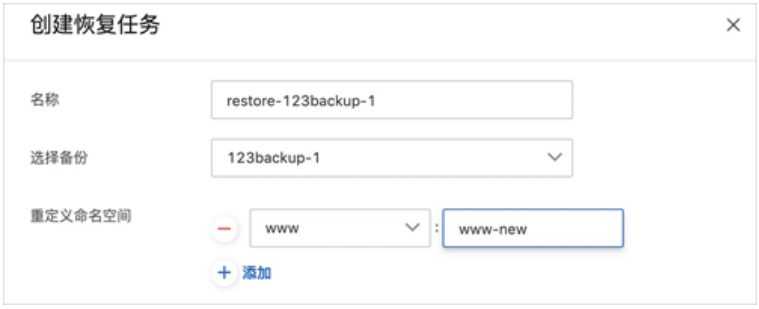
生成快照后，对应PVC的状态会显示为**ConvertionCompleted**。如下图所示：



步骤四：恢复应用和数据卷

> **注意** 对于需要备份存储卷的备份任务，在恢复数据前，请确认已完成备份任务的快照制作。

1. 在**应用备份**页面，单击**恢复**页签。
2. 在**恢复**页签，单击**创建恢复任务**。
3. 在**创建恢复任务**面板配置恢复任务，然后单击**确定**。



配置项	说明
名称	恢复任务名称，仅支持小写字母和数字。
选择备份	选择备份文件。
（可选）重定义命名空间	若需要修改备份数据的命名空间，请单击**添加**，选择备份数据所属的命名空间，在冒号后设置恢复后数据所属的命名空间。

> **说明** 恢复时对于已经存在的资源不会覆盖，只会恢复当前集群中不存在的资源。如果想恢复到之前的版本，则需要删除现有资源再进行恢复。

4. 在提示对话框中单击左下角的**确定**。
5. 验证相关的有状态应用或无状态应用，存储卷和服务可以正常启动和正常访问。
   i. 在恢复集群管理页左侧导航栏选择**工作负载 > 无状态**。
   ii. 在目标应用右侧**操作**列单击**详情**。
   在**容器组**页签下，确认目标应用的状态为Runnning。
   iii. 在集群管理页左侧导航栏中，选择**存储 > 存储声明**。
   在**存储声明**页面可查看已恢复的存储声明。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
   v. 在**服务**页面单击目标的外部端点，确认是否可以访问。

26.2.4. 跨集群迁移应用

阿里云容器服务备份中心为集群内的有状态应用提供灾难备份和恢复能力，对于Kubernetes集群内的有状态应用的崩溃一致性，应用一致性，跨地域的灾难恢复提供了一站式的解决方案。本文介绍如何通过应用备份功能实现跨集群迁移应用。

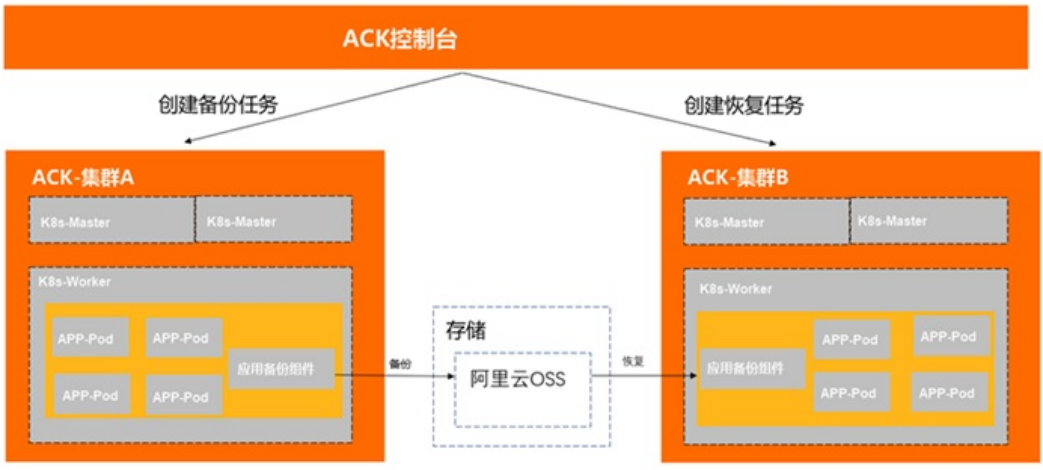
前提条件

已创建两个Kubernetes版本一致的集群。具体操作，请参见创建Kubernetes托管版集群。

示例介绍

本示例中一个名为Cluster_A集群，另一个名为Cluster_B集群。接下来介绍在Cluster_A集群备份应用，在Cluster_B集群恢复应用。

> 注意 创建集群时，请确保两个集群的Kubernetes版本一致，否则会导致跨集群迁移应用失败。



使用应用备份功能迁移应用的流程

1. 授予Cluster_A和Cluster_B集群OSS权限。具体操作，请参见配置OSS权限。
2. 在Cluster_A和Cluster_B集群中安装应用备份组件。具体操作，请参见安装应用备份组件。
3. 在Cluster_A和Cluster_B集群创建备份仓库。创建备份仓库时，给Cluster_A和Cluster_B集群设置同一个OSS，并且选择公网访问。具体操作，请参见创建备份仓库。
4. 在Cluster_A集群创建备份任务。具体操作，请参见创建备份任务。

   Cluster_A集群创建备份任务完成后，您可以在Cluster_B集群**应用备份**页面**备份和快照**页签下看到Cluster_A集群的备份任务。
5. 在Cluster_B集群制作快照。具体操作，请参见制作快照。

   若创建备份任务时已选中**备份存储卷**且存储类型使用了非云盘类型，即使用混合云备份服务HBR（Hybrid Backup Recovery）进行文件备份，则需要制作快照。
6. 在Cluster_B集群**应用备份**页面的**恢复**页签下选择Cluster_A集群的备份任务，执行恢复操作。具体操作，请参见创建恢复任务。

示例一：使用备份中心功能跨集群迁移WordPress应用（云上ACK集群之间）

通过在源K8s集群安装WordPress应用，并且通过发布博客的方式，将博客中文本内容存储在云盘，图片内容存储在NAS，实现应用中既有NAS数据和云盘数据。通过备份中心，将源集群的WordPress应用进行备份，然后在目标集群中将恢复应用和数据，验证数据及应用的完整性。

1. 在**cn-huhehaote**地域创建ACK托管版集群Cluster_A和Cluster_B。

2. 在Cluster_A集群中安装应用WordPress。具体操作，请参见通过应用市场一键部署WordPress。

   在**参数**页签，选择**default**命名空间，将 `storageClass: "alicloud-disk-ssd"` 修改为 `storageClass: "alibabacloud-cnfs-nas"` ，如下所示：

   ```
   persistence:
     enabled: true
     ## wordpress data Persistent Volume Storage Class
     ## If defined, storageClassName: <storageClass>
     ## If set to "-", storageClassName: "", which disables dynamic provisioning
     ## If undefined (the default) or set to null, no storageClassName spec is
     ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
     ##   GKE, AWS & OpenStack)
     storageClass: "alibabacloud-cnfs-nas"
     ## If you want to reuse an existing claim, you can pass the name of the PVC using
     ## the existingClaim variable
     # existingClaim: your-claim
     accessMode: ReadWriteOnce
     size: 20Gi
   ```

3. 在Cluster_A和Cluster_B集群中安装备份服务组件，并创建备份仓库。

   关于安装备份服务组件的操作，请参见安装应用备份组件。关于创建备份仓库的操作，请参见创建备份仓库。

4. 在Cluster_A集群中创建备份任务。具体操作，请参见创建备份任务。

   开启对**default**命名空间的备份。



待备份任务**123backup-1**的状态为**Completed**后，您可以在Cluster_B集群**应用备份**页面的**备份和快照**页签下看到Cluster_A集群的备份任务。

5. 在Cluster_B集群中**备份和快照**页签下，在备份任务**123backup-1**的右侧**操作**列单击**制作快照**。

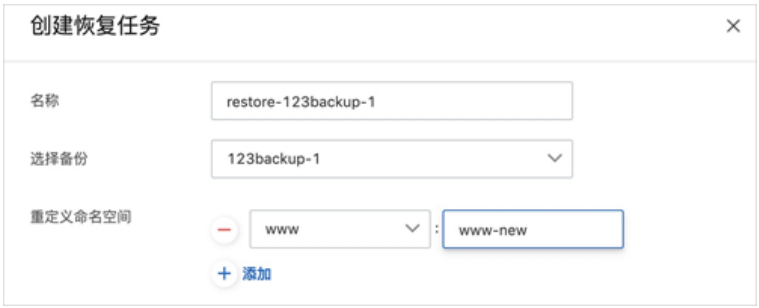
   本示例，在**生成快照**对话框，**转换为存储类**列选择**alicloud-disk**（即云盘）。



待快照生成，即对应PVC的状态变为**ConvertionCompleted**。



6. 在Cluster_B集群中创建恢复任务**restore-123backup-1**。



7. 验证任务是否恢复成功。
   i. 在Cluster_B集群管理页左侧导航栏选择**工作负载 > 无状态**，在WordPress应用的右侧**操作**列单击**详情**，确认WordPress应用的状态为**Runnning**。
   ii. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
   iii. 在**服务**页面单击WordPress的外部端点，可以看到WordPress主页。



示例二：使用备份中心功能跨集群迁移MySQL应用（云下数据中心K8s到云上ACK集群）

示例简述

1. 在线下K8s创建集群（例如，Rancher集群），将线下集群接入注册集群后通过YAML部署使用本地盘的MySQL应用。
2. 云上通过ACK创建托管版集群，部署备份服务组件。
3. 云下纳管集群（即注册集群）通过备份中心备份线下集群的MySQL应用及数据。
4. 在云上ACK集群中，通过备份中心使云下应用和数据在云上恢复。

1. 将线下集群接入注册集群。具体操作，请参见创建注册集群并接入本地数据中心集群。
2. 使用kubectl工具连接注册集群。具体操作，请参见通过kubectl工具连接集群。
3. 创建PV、PVC及MySQL应用。
   i. 使用以下示例创建*mysql.yaml*。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: local-pv
spec:
 capacity:
  storage: 100Gi
 accessModes:
 - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
 local:
  path: /home/
 nodeAffinity:
  required:
   nodeSelectorTerms:
   - matchExpressions:
     - key: kubernetes.io/hostname
       operator: In
       values:
       - for-rancher-k8s-cluster002
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: local-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 100Gi
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: mysql
spec:
 serviceName: "mysql"
 selector:
  matchLabels:
   app: mysql
 template:
  metadata:
   labels:
    app: mysql
  spec:
   containers:
   - image: mysql:5.6
     name: mysql
     env:
     - name: MYSQL_ROOT_PASSWORD
       value: "mysql"
```

```
        value: "mysql"
      volumeMounts:
      - name: local
        mountPath: /var/lib/mysql
    volumes:
      - name: local
        persistentVolumeClaim:
          claimName: local-pvc
```

ii. 执行以下命令部署PV、PVC及MySQL应用。

```
kubectl --kubeconfig rancher_huhehaote_kubeconfig apply -f mysql/mysql.yaml
```

iii. 执行以下命令查看创建的MySQL应用。

```
kubectl --kubeconfig rancher_huhehaote_kubeconfig get pod
```

预期输出：

```
NAME   READY  STATUS   RESTARTS  AGE
mysql-0  1/1    Running  0      171m
```

4. 执行以下命令登录MySQL，创建数据库、建表，并写入测试数据。

```
kubectl --kubeconfig rancher_huhehaote_kubeconfig exec -it mysql-0 sh
root@mysql-0:/# mysql -u root -pmysql
create database test;
create table student(id int, name varchar(20),age int);
insert into student values (1,2,3);
```

5. 在注册集群（即Cluster_A集群）中创建备份任务。具体操作，请参见创建备份任务。

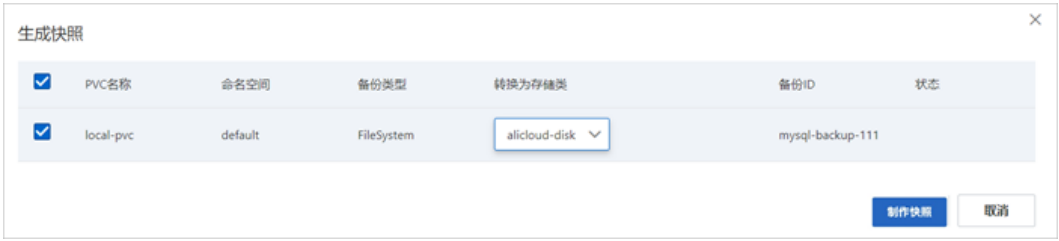
   开启对**default**命名空间的备份。



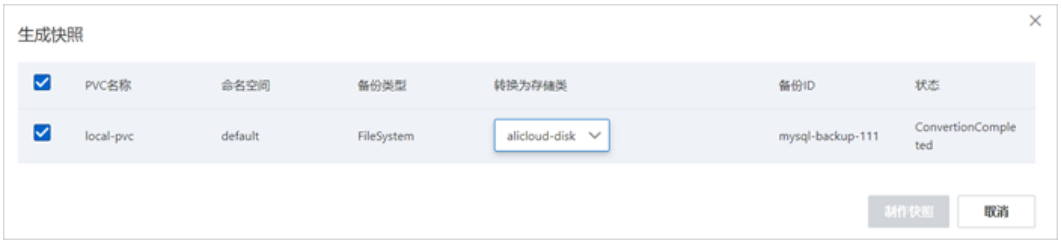
待备份任务**mysql-backup-111**的状态为**Completed**后，您可以在Cluster_B集群**应用备份**页面的**备份和快照**页签下看到Cluster_A集群的备份任务。

6. 在托管版集群（即Cluster_B集群）中**备份和快照**页签下，在备份任务**mysql-backup-111**的右侧**操作**列单击**制作快照**。

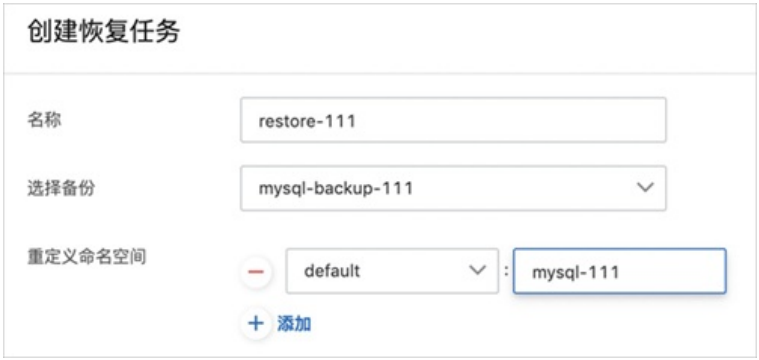
   本示例，在**生成快照**对话框，**转换为存储类**列选择**alicloud-disk**（即云盘）。



   待快照生成，即对应PVC的状态变为**ConvertionCompleted**。



7. 在Cluster_B集群中创建恢复任务restore-111。



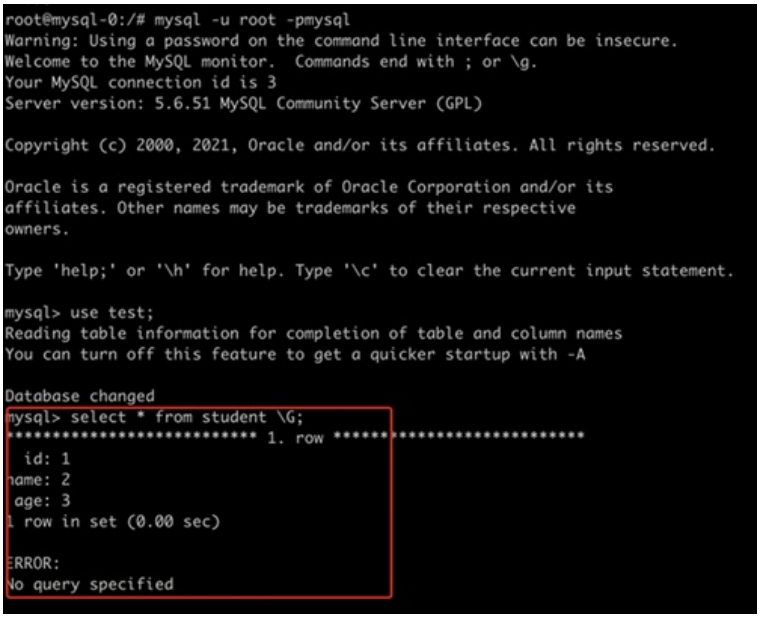
8. 验证任务是否恢复成功。
   i. 执行以下命令查看MySQL应用。

   ```
   kubectl --kubeconfig rancher_online_kube -n mysql-111 get pod
   ```

   预期输出：

   ```
   NAME    READY STATUS  RESTARTS AGE
   mysql-0 1/1   Running 0        37m
   ```

   ii. 登录恢复的数据库查看数据。



   从预期输出可得，应用及写入数据均和线下集群的数据相同。

26.3. 服务网格

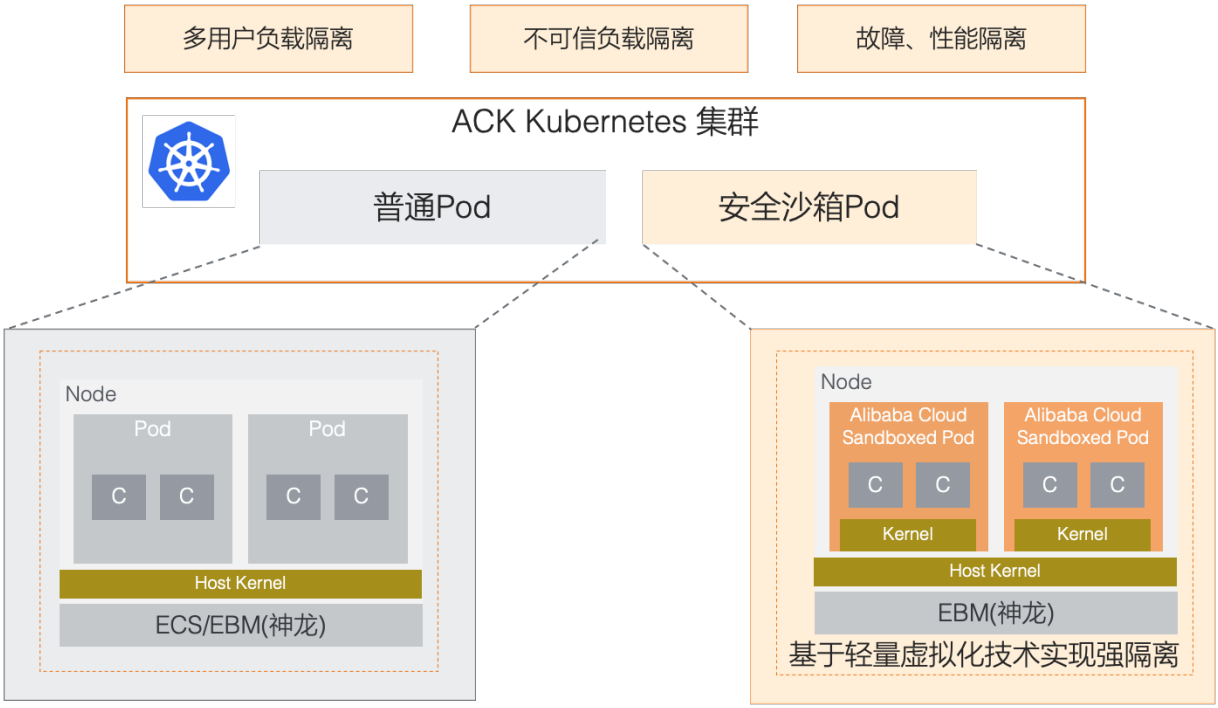
阿里云服务网格（Alibaba Cloud Service Mesh，简称ASM）提供了一个全托管式的服务网格平台，兼容于社区Istio开源服务网格，用于简化服务的治理，包括服务调用之间的流量路由与拆分管理、服务间通信的认证安全以及网格可观测性能力，从而极大地减轻开发与运维的工作负担。
关于服务网格的更多信息，请参见什么是服务网格ASM？。

27.安全沙箱

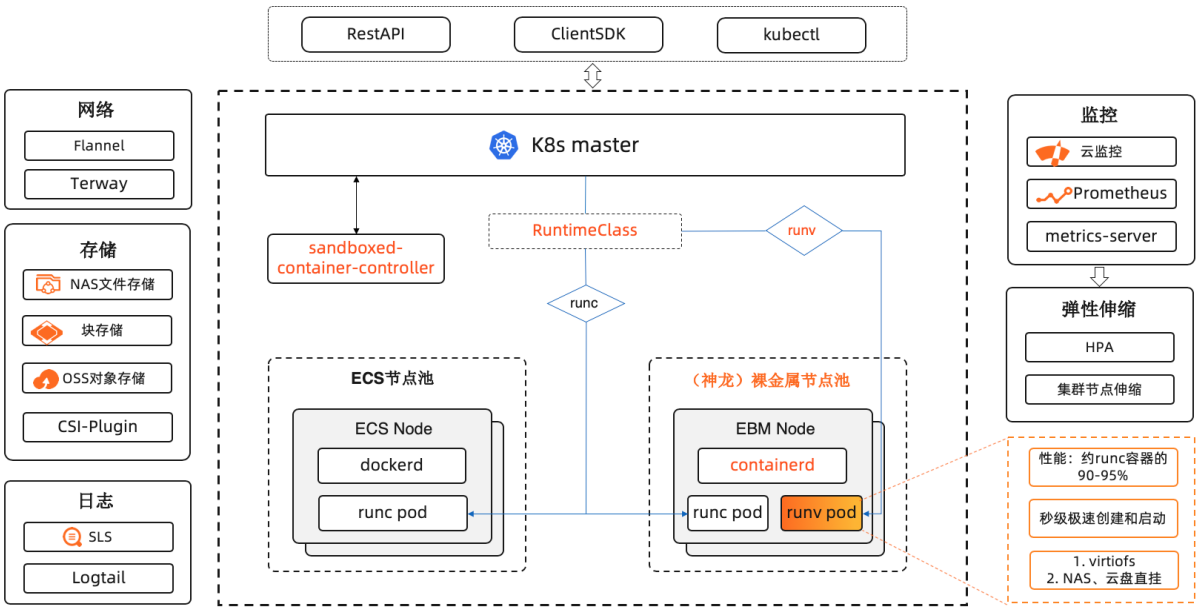
27.1. 安全沙箱概述

相比于原有Docker运行时，安全沙箱为您提供的一种新的容器运行时选项，可以让您的应用运行在一个轻量虚拟机沙箱环境中，拥有独立的内核，具备更好的安全隔离能力。

安全沙箱特别适合于不可信应用隔离、故障隔离、性能隔离、多用户间负载隔离等场景。在提升安全性的同时，对性能影响非常小，并且具备与Docker容器一样的用户体验，例如日志、监控、弹性等。



架构图



特点

安全沙箱v2是阿里云全新自研的基于轻量虚拟机技术的安全容器运行时，相比于v1版本，在保持超强隔离性的同时，overhead降低了90%，沙箱启动速度提升了3倍，单机密度提升了10倍。主要特点如下：

- 基于轻量虚拟机，沙箱之间实现超强隔离。
- 具有传统runC容器的应用兼容性。
- 应用综合性能高，达到runC容器应用性能的90%。
- NAS、云盘、OSS Volume同时支持virtiofs和直通两种模式。
- 在监控、日志、存储等方面有着runC一样的使用体感。
- 支持RuntimeClass（runC和runV），请参见RuntimeClass。
- 技能门槛要求低、简单易用。
- 相比社区Kata Containers，稳定性更强。有关Kata Container详情，请参见Kata Containers。

技术交流&答疑

您可以通过钉钉扫描以下二维码或搜索群号**30521601**加入**ACK安全沙箱&ACK-TEE密计算用户交流**群。

相关文档

- 如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？
- runC容器和安全沙箱（runV）容器的区别
- 为什么选择安全沙箱？
- 创建安全沙箱托管版集群

27.2. runC容器和安全沙箱（runV）容器的区别

本文通过对比runC容器和安全沙箱（runV）容器的性能和创建方式，让您了解安全沙箱容器的优势以便更好地使用安全沙箱容器。

runC容器和安全沙箱（runV）容器方案对比

配置	runC容器	安全沙箱（runV）容器
容器引擎	Docker、Containerd	Containerd
容器所在节点型号	ECS、EBM	EBM
容器内核	与宿主机共享内核	独占内核
容器隔离方式	Cgroups和Namespace	轻量虚拟机
Rootfs Graph Driver	OverlayFS	OverlayFS
Rootfs IO限速	Cgroups	否
NAS直挂	否	是
云盘直挂	否	是

配置	runC容器	安全沙箱（runV）容器
容器日志文件采集	logtail可直接从节点侧采集容器内的日志。	logtail sidecar，详情请参见通过Sidecar-CRD方式采集容器文本日志。
Pod Overhead	无	Memory：M=64 Mi+Pod内存申请 *2%。当M>512，M最终取值为512 Mi。当M<64，M最终取值为64。

runC Pod和安全沙箱（runV）Pod的创建方式对比

您可以通过kubectl命令行工具连接容器服务Kubernetes版（ACK）集群，请参见通过kubectl工具连接集群。

- 创建runC Pod
  i. （可选）通过 `runtimeClassName: runc` 指定使用runC容器运行时。

  > **说明** 您也可以不配置，Pod会用默认的容器运行时runC。

  ii. 执行以下命令创建runC Pod。

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
 name: busybox-runc
 labels:
  app: busybox-runc
spec:
 containers:
 - name: busybox
  image: registry.cn-hangzhou.aliyuncs.com/acs/busybox:v1.29.2
  command:
  - tail
  - -f
  - /dev/null
  resources:
   limits:
    cpu: 1000m
    memory: 512Mi
   requests:
    cpu: 1000m
    memory: 512Mi
EOF
```

- 创建安全沙箱容器（runV）
  i. 通过 `runtimeClassName: runv` 指定使用安全沙箱容器运行时。
  ii. （可选）执行以下命令查看集群中已经存在的 `runv` 这个RuntimeClass对象。

```
kubectl get runtimeclass runv -o yaml
```

  > **说明** 安全沙箱容器集群中默认创建了RuntimeClass对象 `runv` 。

  iii. 执行以下命令创建安全沙箱Pod。

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
 name: busybox-runv
 labels:
  app: busybox-runv
spec:
 runtimeClassName: runv
 nodeSelector:
  alibabacloud.com/container-runtime: Sandboxed-Container.runv
 containers:
 - name: busybox
  image: registry.cn-hangzhou.aliyuncs.com/acs/busybox:v1.29.2
  command:
  - tail
  - -f
  - /dev/null
  resources:
   limits:
    cpu: 1000m
    memory: 512Mi
   requests:
    cpu: 1000m
    memory: 512Mi
EOF
```

注意 如果您的Kubernetes版本小于1.16（不包括1.16），需添加以下nodeSelector配置；如果您的Kubernetes版本在1.16（包括1.16）以上则不需要添加。

```
nodeSelector:
  alibabacloud.com/container-runtime: Sandboxed-Container.runv
```

iv. 执行以下命令查看创建的Pod。如果输出结果为 `runv` ，则表示Pod已经运行在安全沙箱内了。

```
kubectl get pod busybox-runv -o jsonpath={.spec.runtimeClassName}
```

v. 执行以下命令登录容器查看CPU和内存规格。

```
kubectl exec -ti pod busybox-runv /bin/sh
/ # cat /proc/meminfo | head -n1
MemTotal:     1130692 kB
/ # cat /proc/cpuinfo | grep processor
processor   : 0
```

通过以上命令返回的结果，可以看出CPU核数并非宿主机CPU核数，内存总大小为Pod内存与Overhead内存之和（系统会消耗一些内存，所以看到的内存会少一些）。

相关文档

- 安全沙箱兼容性说明
- RuntimeClass

27.3. 如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？

镜像和容器已然成为应用的打包和交付标准。Kubernetes成为了整个容器云原生应用的标准OS，越来越多的企业和用户选择在ACK中部署自己的应用。容器服务Kubernetes版（ACK）支持Containerd、Docker、安全沙箱三种运行时。本文通过对比三种运行时的实现和使用限制、部署结构，并通过对比Docker和Containerd两种容器引擎常用命令，帮助您根据需求场景选择合适的容器运行时。

容器运行时实现和使用限制的对比

特性	Containerd运行时	Docker运行时	安全沙箱运行时	说明
集群类型	支持托管版和专有版	支持所有类型	支持托管版和专有版	无
节点型号	支持： • ECS • EBM	支持： • ECS • EBM	支持： EBM	无
节点OS	支持： • CentOS • Alibaba Cloud Linux 2 • ACK v1.20.4版本的Windows	支持： • CentOS • Alibaba Cloud Linux 2 • Windows	支持： Alibaba Cloud Linux 2定制版	• 不支持同一节点同时部署Docker和安全沙箱两种运行时。 • 集群内可以通过创建不同节点池来实现Docker运行时节点和安全沙箱运行时节点混合部署。
容器引擎	支持Containerd	支持Docker	支持Containerd	无
监控	支持	支持	支持	无
容器日志文件采集	支持	支持	手动支持（Sidecar）	有关Sidecar配置，请参见通过Sidecar-CRD方式采集容器文本日志。
容器标准输出采集	支持	支持	支持	无
RuntimeClass	不支持	不支持	支持（runV）	无

特性	Containerd运行时	Docker运行时	安全沙箱运行时	说明
Pod调度	无需配置	无需配置	配置如下： • Kubernetes v1.14.x版本必须为nodeSelector增加以下配置。 `alibabacloud.com/sandboxed-container: Sandboxed-Container.runv` • Kubernetes v1.16.x以及更高版本不需要任何配置。	无
HostNetwork	支持	支持	不支持	无
exec/logs	支持	支持	支持	无
节点数据盘	可选	可选	必选（不小于200 GiB）	无
网络插件	支持： • Flannel • Terway	支持： • Flannel • Terway	支持： • Flannel • Terway：仅支持非独占弹性网卡模式。	无
kube-proxy代理模式	支持： • Iptables • IPVS	支持： • Iptables • IPVS	支持： • Iptables • IPVS	无
存储插件	支持CSI Plugin	支持CSI Plugin	支持CSI Plugin	无
容器RootFS	支持OverlayFS	支持OverlayFS	支持配置磁盘Quota的OverlayFS	无

运行时部署结构对比

运行时	部署结构
Docker	`kubelet -> dockerd -> containerd -> containerd-shim -> runC容器`

运行时	部署结构
Containerd	kubelet -> containerd -> containerd-shim -> runC容器
安全沙箱v2	kubelet -> (CRI)containerd \-> containerd-shim -> runC容器 \-> containerd-shim-rund-v2 -> runV安全沙箱容器

Docker和Containerd两种容器引擎常用命令对比

Docker运行时和安全沙箱运行时的容器引擎分别是Docker和Containerd。这两种容器引擎都有各自的命令工具来管理镜像和容器。两种容器引擎常用命令对比如下。

命令	Docker	Containerd	
	docker	crictl（推荐）	ctr
查看容器列表	docker ps	crictl ps	ctr -n k8s.io c ls
查看容器详情	docker inspect	crictl inspect	ctr -n k8s.io c info
查看容器日志	docker logs	crictl logs	无
容器内执行命令	docker exec	crictl exec	无
挂载容器	docker attach	crictl attach	无
显示容器资源使用情况	docker stats	crictl stats	无
创建容器	docker create	crictl create	ctr -n k8s.io c create
启动容器	docker start	crictl start	ctr -n k8s.io run
停止容器	docker stop	crictl stop	无
删除容器	docker rm	crictl rm	ctr -n k8s.io c del
查看镜像列表	docker images	crictl images	ctr -n k8s.io i ls
查看镜像详情	docker inspect	crictl inspecti	无
拉取镜像	docker pull	crictl pull	ctr -n k8s.io i pull
推送镜像	docker push	无	ctr -n k8s.io i push
删除镜像	docker rmi	crictl rmi	ctr -n k8s.io i rm
查看Pod列表	无	crictl pods	无
查看Pod详情	无	crictl inspectp	无
启动Pod	无	crictl runp	无

命令	Docker	Containerd	
	docker	crictl（推荐）	ctr
停止Pod	无	crictl stopp	无

27.4. 为什么选择安全沙箱？

本文介绍安全沙箱运行时的核心优势、适用场景，并对比容器服务Kubernetes版（ACK）安全沙箱和社区Kata Containers的性能，帮助您了解为什么选择安全沙箱运行时。

背景信息

相比原有Docker运行时，安全沙箱为您提供了一种新的容器运行时选项，具备以下特点：

- 它可以让您的应用运行在一个轻量虚拟机沙箱环境中，拥有独立的内核，具备更好的安全隔离能力。
- 相比社区方案（Kata Containers），安全沙箱在存储、网络和稳定性方面都有优化和提升。
- 安全沙箱特别适合于不可信应用隔离、故障隔离、性能隔离、多用户间负载隔离等场景，在提升安全性的同时，对性能影响非常小，并且具备与Docker容器一样的用户体验，例如日志、监控、弹性等。

安全沙箱核心优势

对比Docker运行时，安全沙箱核心优势如下：

- 基于轻量虚拟机沙箱超强隔离。
- NAS、云盘直挂沙箱，直挂后的存储性能与宿主机挂载模式性能一致。
- 媲美传统runC容器的应用兼容性。
- 极高的应用综合性能（相当于90% runC容器应用性能）。
- 在监控、日志、存储等方面有着与runC一样的使用体感。
- 支持RuntimeClass。
- 技能门槛要求低、简单易用，您无需理解虚拟机技术原理，也不需要在虚拟机上操作。
- 相对社区Kata Containers，有着更强的稳定性。

ACK安全沙箱和社区Kata Containers对比

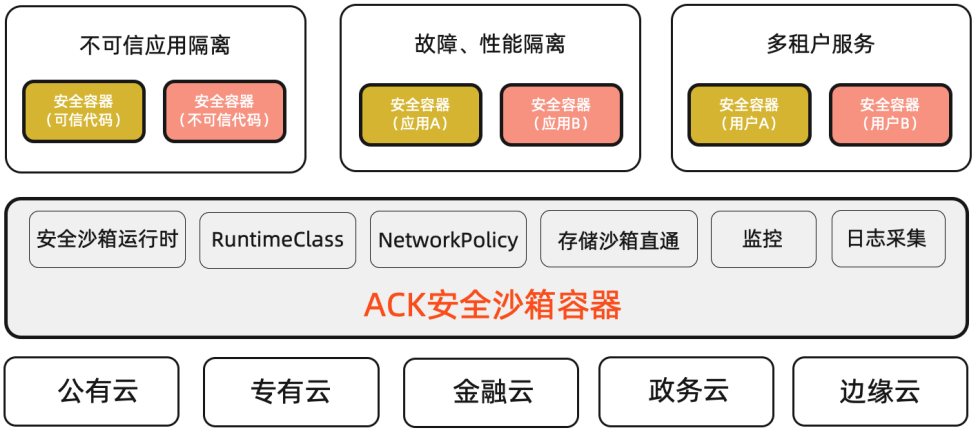
ACK安全沙箱相比社区Kata Containers，在性能上面具有较大优势，如下表所示。

性能	性能分类	ACK安全沙箱v2	社区Kata Containers
沙箱启动速度		约150ms	约500ms
沙箱额外开销		低	高
容器RootFS		virtio-fs，性能：☆☆☆☆	9pfs，性能：☆
容器存储卷	HostPath/EmptyDir	virtio-fs，性能：☆☆☆☆	
	云盘块存储	• virtio-fs（默认），性能：☆☆☆☆ • 直通沙箱，性能：☆☆☆☆☆	

性能	性能分类	ACK安全沙箱v2	社区Kata Containers
	NAS文件存储	• virtio-fs（默认），性能：☆☆☆☆ • 直挂沙箱，性能：☆☆☆☆☆	
	OSS对象存储	virtio-fs，性能：☆☆☆☆	
网络插件		• Terway：比Flannel网络性能提升了20%~30%，支持NetworkPolicy，带宽限速等。更多特性请参见Terway网络插件。 • Flannel：支持VPC路由。	Flannel
监控告警		• 增强了安全沙箱Pod的磁盘和网络监控指标。 • 默认集成了阿里云云监控，可方便地配置集群的监控告警。	缺失安全沙箱Pod的磁盘和网络监控指标。
稳定性		☆☆☆☆☆	☆☆

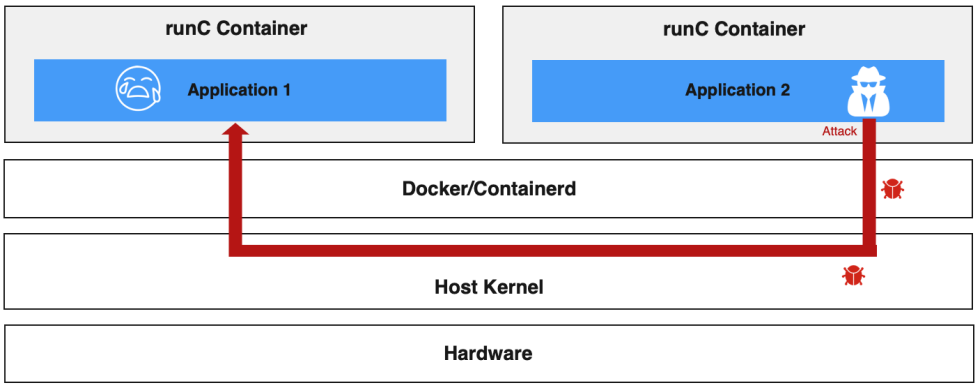
ACK安全沙箱三大适用场景

以下是安全沙箱的三大适用场景。



- 场景一：相对于runC容器，安全沙箱（runV）容器可以强隔离不可信应用。

- runC容器的安全风险

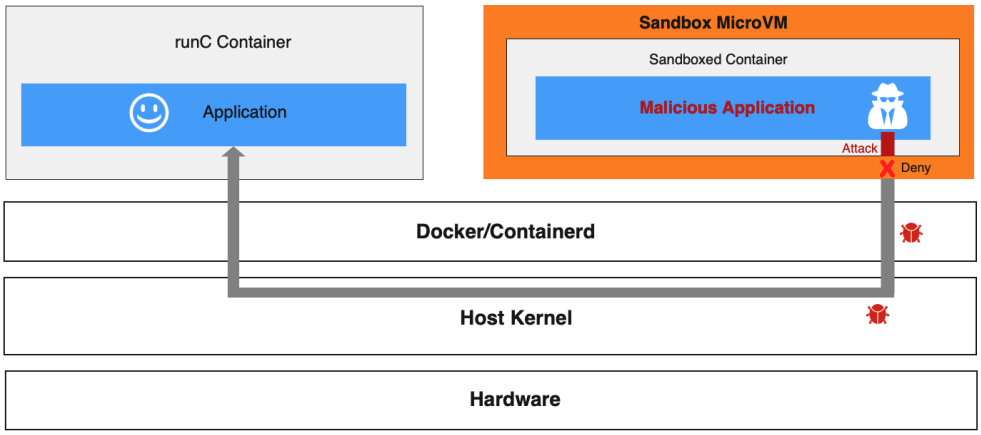


  - 通过Namespace和Cgroups技术隔离的容器攻击面大。
  - 节点上所有容器共享Host Kernel，一旦内核出现漏洞，恶意代码会逃逸到Host上，渗透到后端内网，执行恶意特权代码，破坏系统服务和其他应用，窃取重要数据等。
  - 应用本身的漏洞同样会造成攻击者渗透到内网。

  您可以通过以下措施降低runC容器的安全风险：
  - Seccomp：系统调用过滤。
  - SElinux：限制容器进程、文件和用户的权限。
  - Capability：限制容器进程Capability。
  - dockerdrootless模式：禁止用户以root身份运行Docker守护进程和容器。

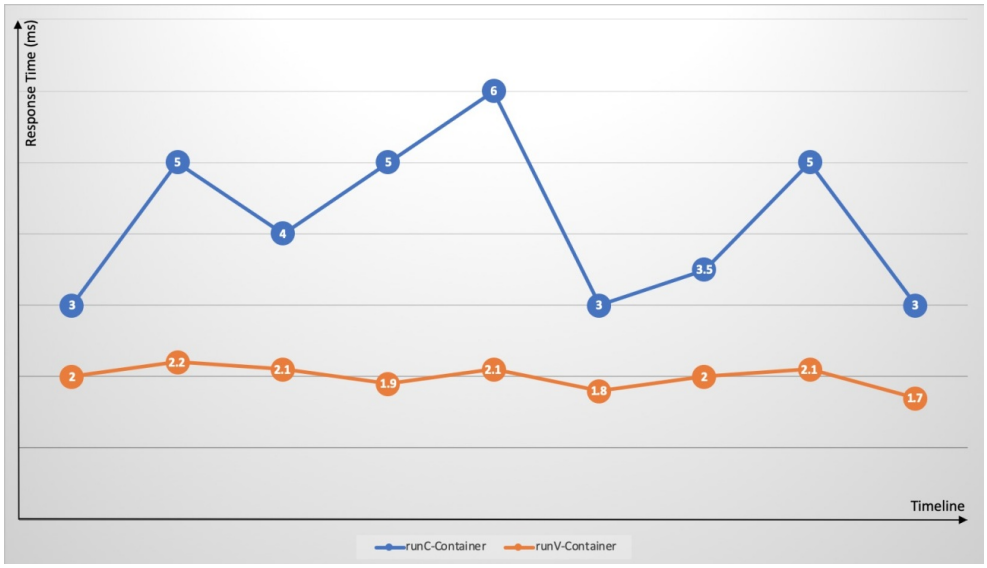
  虽然以上措施可以在一定程度上强化runC容器的安全性，降低恶意容器应用攻击Host Kernel的几率，但是仍然无法解决容器逃逸利用Host Kernel漏洞的安全问题。

- 安全沙箱（runV）容器隔离潜在安全风险



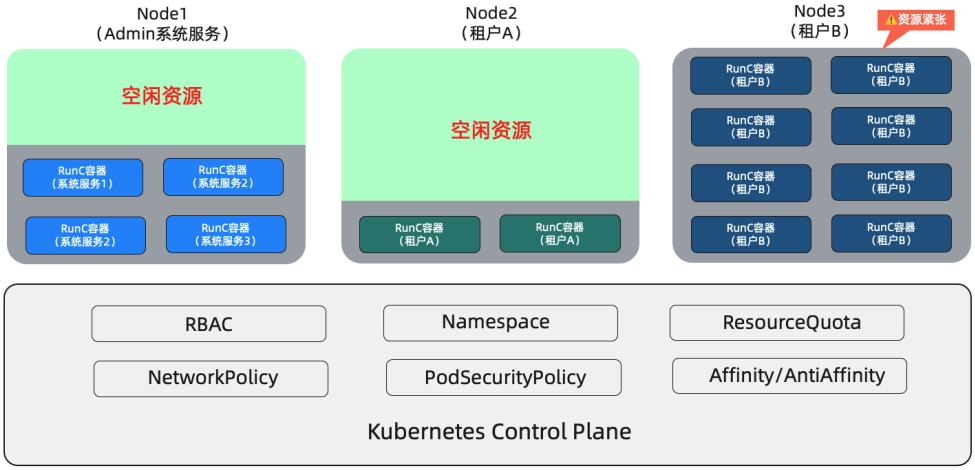
  通过把存在潜在安全风险的应用放置到成熟的轻量级虚拟机沙箱中，应用运行在独立的Guest OS Kernel上，即便Guest OS Kernel出现安全漏洞，那么攻击破坏面仅限于一个沙箱内，不会对Host Kernel以及其他容器有任何影响。安全沙箱（runV）容器配合Terway的NetworkPolicy能力，可以灵活地配置Pod的网络访问策略，真正做到系统隔离、数据隔离以及网络隔离。

- 场景二：解决runC容器在故障放大、资源争抢、性能干扰方面的问题。



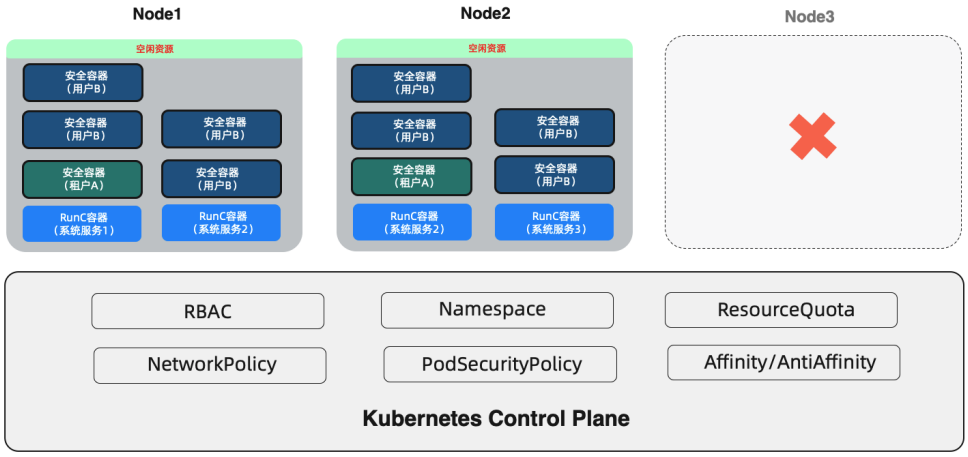
Kubernetes使得我们很容易在一个节点上混合部署不同的应用容器，由于Cgroups并不能很好解决资源争抢问题，导致同一节点上相同资源密集型（如CPU密集型、IO密集型等）的不同应用相互争抢资源，导致应用的响应时间出现了严重的波动，总体响应时间边长。当节点上某一应用异常和故障，如内存泄露、频繁CoreDump等等导致节点整体负载升高，单容器触发Host Kernel Bug导致系统宕机，单应用的故障延展到了整个节点，甚至进一步导致整个集群的不响应。安全沙箱（runV）容器通过独立的Guest OS Kernel和Hypervisor，可以很好地解决runC容器在故障放大、资源争抢、性能干扰方面的问题。

- 场景三：多租户服务
  通常一个企业内有多个业务线或部门部署自己的应用，不同的业务线或部门（多个租户）之间有着较强的隔离诉求，如金融类业务不期望自己的物理环境运行着其他非安全敏感应用，传统runC容器是无法有效避免不可信应用带来的潜在安全风险。这种情况下，通常会选择：
  - 多个单租户集群，如金融业务集群和其他非安全敏感业务分隔到多个独立集群。
  - 单个多租户集群，需把不同业务线应用分隔到不同的Namespace中，每个节点只可被某个业务线独占，配合资源配额、网络策略等实现多租户隔离。相对于多个单租户集群，管控面较少，管理成本较低，但仍然没有解决因某些租户资源利用率低导致节点资源闲置浪费问题。



有了安全沙箱（runV）容器后，可以把集群内不可信应用通过虚拟机沙箱隔离起来，而不用担心不可信应用容器逃逸造成的安全风险，这样所有节点都可以混合各类应用容器，这样做的好处是：

- 减少了资源调度的复杂度。
- 节点不再被单个业务独占，减少资源碎片，提高节点资源利用率，节省集群整体资源成本。
- 安全沙箱（runV）容器使用轻量级虚拟机，性能媲美runC容器。



27.5. 创建安全沙箱集群

27.5.1. 创建安全沙箱托管版集群

您可以通过容器服务管理控制台创建安全沙箱托管版集群。

前提条件

- 您需要开通容器服务和访问控制RAM服务。
- 登录容器服务管理控制台和RAM管理控制台开通相应的服务。

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。
- ACK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。

    > **注意** 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，默认路由条目上限是48个。如果需要更大的路由条目数，需要您先对目标VPC提交工单，申请提高配额。

  - 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。
- 创建安全沙箱Kubernetes集群时，以下界面参数需要按照表中的要求配置，否则创建的集群将无法使用安全沙箱的场景。

参数	配置
可用区	目前只有神龙实例支持安全沙箱容器，请确保所选可用区有神龙实例。
Kubernetes版本	1.14.6-aliyun.1或更高版本。
容器运行时	安全沙箱

参数	配置
Worker实例	新增实例
实例规格	弹性裸金属服务器（神龙）
挂载数据盘	请至少挂载一块不小于200 GiB的数据盘，建议选择1 T以上。
操作系统	安全沙箱集群默认为Alibaba Cloud Linux操作系统，不可选。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**页面的**托管集群**区域，选择**标准托管集群**，并单击**创建**。
5. 在**ACK托管版**页面，完成集群配置。
   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
集群规格	选择集群规格，支持**标准版**和**Pro版**。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。
账号全部资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
Kubernetes版本	选择版本。

配置项	描述
容器运行时	选择**安全沙箱**。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体请参见Terway与Flannel对比。 ■ 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，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。 **说明** ■ 在Terway模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。 ■ 网络插件选择**Terway**时，会使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 **说明** ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。

配置项	描述
Pod虚拟交换机	网络插件选择Terway时，您需要为Pod分配IP的虚拟交换机。每个Pod虚拟交换机分别对应一个Worker实例的虚拟交换机，Pod虚拟交换机和Worker实例的虚拟交换机的可用区需一致。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。
配置SNAT	创建集群时，默认不开通公网。如果您选择的VPC不具备公网访问能力，选中**为专有网络配置SNAT**后，ACK将为您创建NAT网关并自动配置SNAT规则。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 **注意** 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 **说明** 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 **说明** ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。
资源组	创建的集群将归属于选择的资源组。一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念。更多信息，请参见资源组。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。
标签	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。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	自定义配置集群apiserver服务端证书中的SAN。多个IP或域名以英文逗号（,）分隔。具体操作，请参见自定义集群API Server证书SAN。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影

6. 单击**下一步：节点池配置**，完成Worker节点配置。

说明 安全沙箱集群中，Worker节点仅支持神龙工作节点。

i. 完成Worker节点基础选项配置。

配置项	描述
Worker实例	只支持**新增实例**，暂不支持**添加已有实例**。

配置项	描述
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	选择**弹性裸金属服务器（神龙）**，目前机型仅支持神龙裸金属服务器。
已选规格	已选的实例规格。只能选择一个规格。
数量	新增Worker实例（ECS实例）的数量。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。 **注意** 挂载的数据盘用于存储节点上所有容器的根文件系统。因此，请至少挂载一块不小于200 GiB的数据盘，建议1 T以上。
操作系统	安全沙箱集群默认为Alibaba Cloud Linux操作系统，不可选。
登录方式	■ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

7. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认**不安装**。 ○ 若选择**Nginx Igress**，则默认选中**安装Ingress组件**。关于Nginx Igress的更多使用说明，请参见Ingress高级用法。 ○ 若选择**ALB Ingress（公测）**，则默认安装ALB Ingress Controller组件。关于如何在ACK集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。 **说明** 如果您勾选**创建Ingress Dashboard**，则需要同步安装日志服务组件。
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	安全沙箱仅支持CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-CSI。
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。
报警配置	选中**使用默认报警模板配置报警**，开启默认报警规则。详细介绍，请参见容器服务报警管理。

配置项	描述
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**安装node-problem-detector并创建事件中心**，您可以选择是否在日志服务控制台中添加事件中心。更多信息，请参见创建并使用Kubernetes事件中心。
控制平面组件日志	选中**开启**，将从ACK控制层收集托管集群控制平面组件日志到您账号中的SLS日志服务的Log Project中。更多信息，请参见收集托管集群控制平面组件日志。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

8. 单击**下一步：确认配置**。
9. 选中**服务协议**并单击**创建集群**。

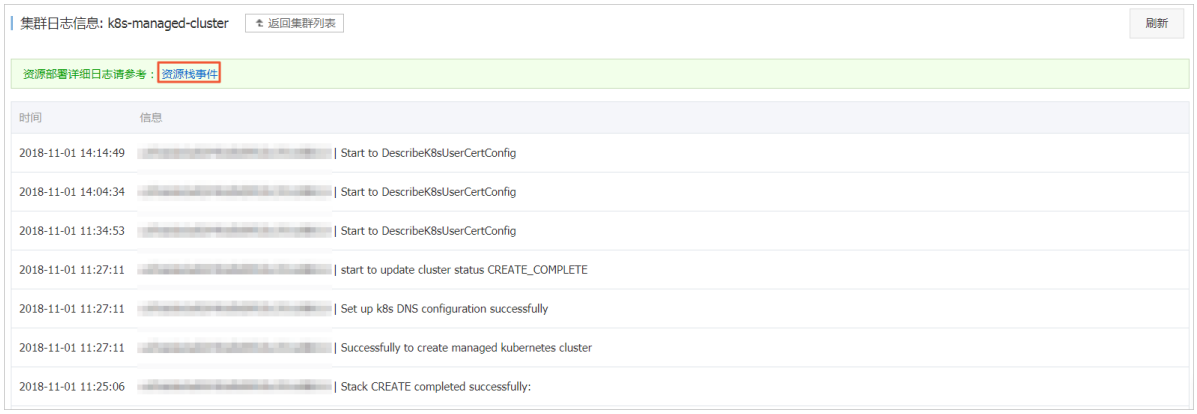
说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

执行结果

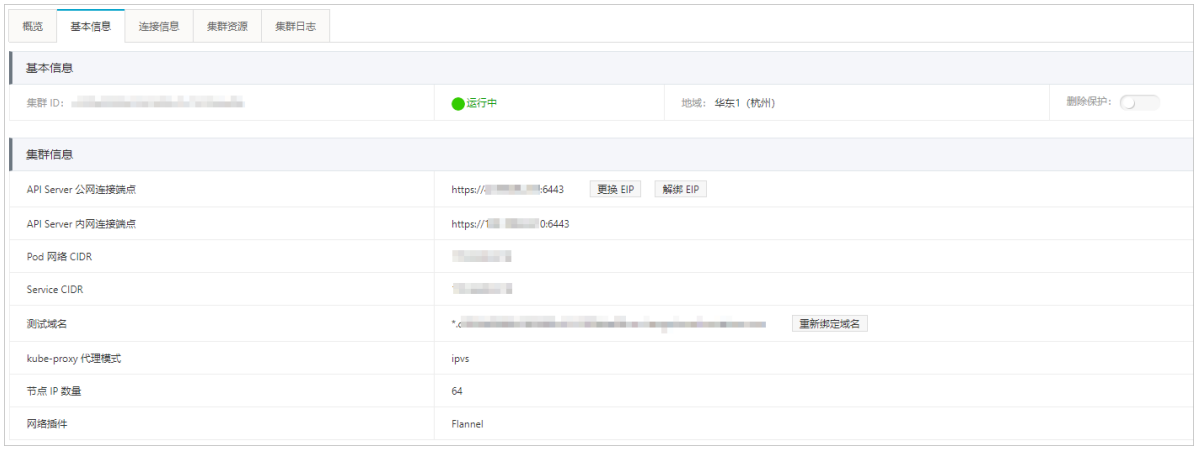
- 集群创建成功后，您可以在容器服务管理控制台的Kubernetes**集群列表**页面查看所创建的集群。



- 您可以单击集群**操作**列下的**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。



- 您可以单击集群**操作**列下的**详情**，然后单击**基本信息**和**连接信息**页签，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。
  绑定EIP和**解绑EIP**功能仅支持托管版Kubernetes集群。
  - 绑定EIP：您可以选择在已有EIP列表中绑定EIP或者新建EIP。
    绑定EIP操作为导致API Server短暂重启，请避免在此期间操作集群。
  - 解绑EIP：解绑EIP后您将无法通过公网访问API Server。
    解绑EIP操作会导致API Server短暂重启，请避免在此期间操作集群。
- **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。
- **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` 。

  > **说明** 单击**重新绑定域名**，您可以重新绑定访问域名。

- 您可以通过kubectl工具连接集群，执行 `kubectl get node` 查看集群的节点信息。

```
For more tutorials, visit https://api.aliyun.com/#/lab
shell@Alicloud:~$ use-k8s-cluster

Type "kubectl" to manage your kubenetes cluster

shell@Alicloud:~$ kubectl get node
NAME                    STATUS   ROLES    AGE    VERSION
cn-beijing.             Ready    <none>   4d4h   v1.14.6-aliyun.1
cn-beijing.             Ready    <none>   4d     v1.14.6-aliyun.1
cn-beijing.             Ready    <none>   4d4h   v1.14.6-aliyun.1
shell@Alicloud:~$ kubectl get runtimeclass
NAME   CREATED AT
runc   2019-09-19T04:01:00Z
runv   2019-09-19T04:01:00Z
shell@Alicloud:~$
```

27.5.2. 创建安全沙箱专有版集群

您可以通过容器服务管理控制台创建安全沙箱专有版集群。

前提条件

开通相应服务，具体操作，请参见首次使用容器服务Kubernetes版。

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。
- ACK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。

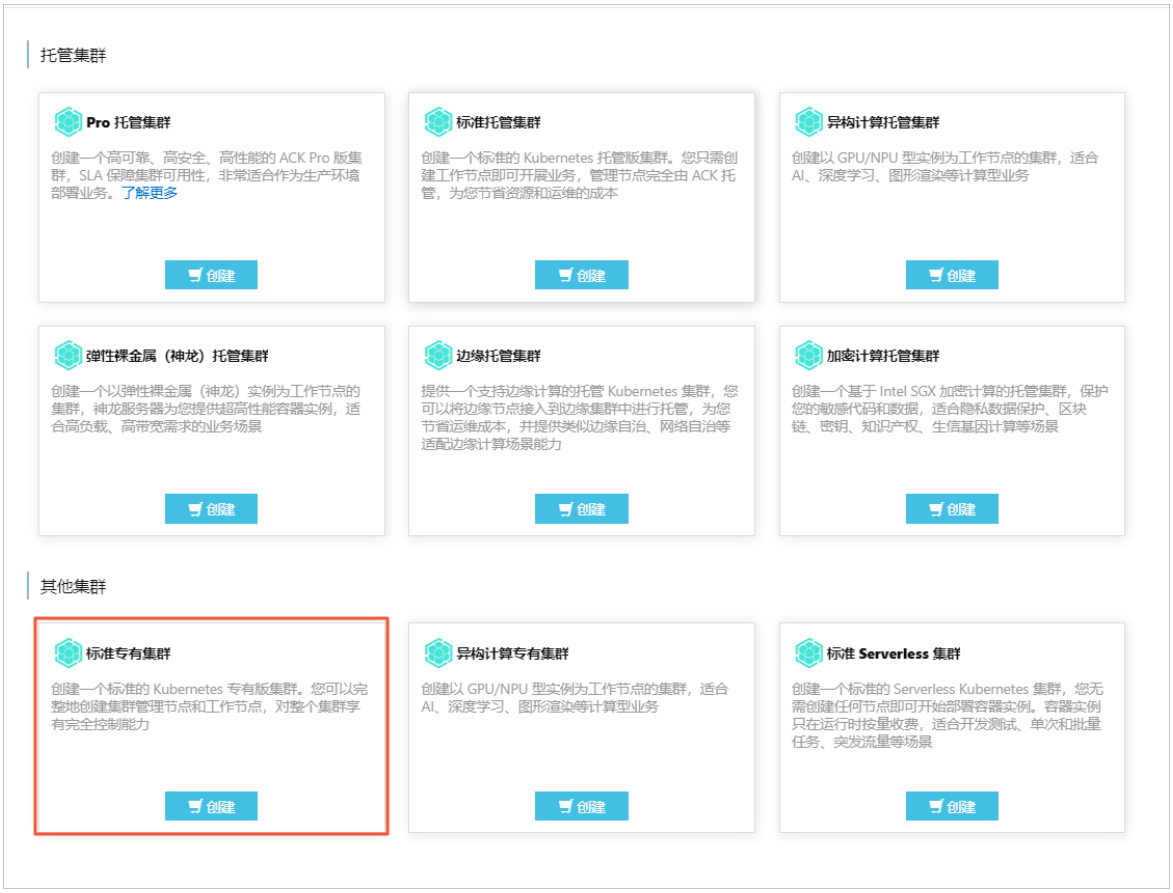
    > **注意** Kubernetes集群中，VPC默认路由条目不超过48条，意味着Kubernetes集群使用VPC时，默认路由条目上限是48个。如果需要更大的路由条目数，需要您先对目标VPC提交工单，申请提高配额。

  - 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。
- 创建安全沙箱Kubernetes集群时，以下界面参数需要按照表中的要求配置，否则创建的集群将无法使用安全沙箱的场景。

参数	配置
可用区	目前只有神龙实例支持安全沙箱容器，请确保所选可用区有神龙实例。
Kubernetes版本	1.14.6-aliyun.1或更高版本。
容器运行时	安全沙箱
Worker实例	新增实例
实例规格	弹性裸金属服务器（神龙）
挂载数据盘	请至少挂载一块不小于200 GiB的数据盘，建议选择1 T以上。
操作系统	安全沙箱集群默认为Alibaba Cloud Linux操作系统，不可选。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**页面的**其他集群**区域，选择**标准专有集群**，并单击**创建**。



5. 在**ACK专有版**页面完成集群配置。

   i. 完成集群基础选项配置。

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。

配置项	描述
资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
Kubernetes版本	选择版本。
容器运行时	选择**安全沙箱**。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。
网络插件	设置启用的网络插件和插件配置，支持Flannel和Terway网络插件，具体请参见Terway与Flannel对比。 ■ 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，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。 **说明** ■ 在Terway模式下，节点上可以运行的Pod数均受节点的弹性网卡和辅助IP的配额限制。 ■ 当专有网络选择共享VPC时，网络插件仅支持Terway。 ■ 网络插件选择Terway时，会使用弹性网卡的辅助IP分配给Pod，一个Pod占用一个弹性网卡辅助IP地址。

配置项	描述
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 说明 ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Pod虚拟交换机	网络插件选择Terway时，您需要为Pod分配IP的虚拟交换机。每个Pod虚拟交换机分别对应一个Worker实例的虚拟交换机，Pod虚拟交换机和Worker实例的虚拟交换机的可用区需一致。 由于Pod的弹性网卡需要和所在的机器在同一个可用区，所以需要为上面所选的集群节点的VSwitch相对应可用区的Pod选择另一个VSwitch，集群运行时Pod的IP地址将会从所选择的VSwitch中分配，为了保障Pod可用数量，建议选择掩码不大于19的VSwitch。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。
配置SNAT	创建集群时，默认不开通公网。如果您选择的VPC不具备公网访问能力，选中**为专有网络配置SNAT**后，ACK将为您创建NAT网关并自动配置SNAT规则。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 注意 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。

配置项	描述
SSH登录	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，具体操作请参见通过SSH连接ACK专有版集群的Master节点。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 **说明** 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 **说明** ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。

ii. 完成集群高级选项配置。

配置项	描述
时区	选择集群所要使用的时区。默认时区为浏览器所配置的时区。
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。
标签	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。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影
集群CA	如果选中自定义集群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。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。
资源组	创建的集群将归属于选择的资源组。一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念。更多信息，请参见资源组。

6. 单击**下一步：Master配置**，完成Master节点配置。

配置项	描述
Master实例数量	设置您所需的Master节点数量。目前支持创建3个或者5个Master节点。
实例规格	选择Master节点的实例规格，详细的实例规格请参见实例规格族。
系统盘	默认为Master节点挂载系统盘，支持**SSD云盘**和**高效云盘**。 **说明** 支持选中**开启云盘备份**备份云盘数据。

7. 单击**下一步：节点池配置**，完成Worker节点配置。
   i. 完成Worker基础选项配置。

配置项	描述
Worker实例	只支持**新增实例**，暂不支持**添加已有实例**。
节点池名称	填写节点池名称。 **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	选择**弹性裸金属服务器（神龙）**。目前机型仅支持神龙裸金属服务器。
已选规格	已选的实例规格。只能选择一个规格。
数量	新增Worker实例（ECS实例）的数量。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 **说明** ■ 支持选中**开启云盘备份**以备份云盘数据。 ■ **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。 **注意** 挂载的数据盘用于存储节点上所有容器的根文件系统。因此，请至少挂载一块不少于200GiB的数据盘，建议1T以上。
操作系统	安全沙箱集群默认为Alibaba Cloud Linux操作系统，不可选。

配置项	描述
登录方式	■ 设置密钥。 ■ **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。 ■ **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ■ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。 **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	请参见ECS实例自定义数据概述。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。 例如：节点IP地址为192.1xx.x.xx，指定前缀为aliyun.com，IP地址段长度为5，后缀为test，则节点名称为aliyun.com00055test。
节点服务端口范围	设置节点服务端口范围。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

8. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认**不安装**。 ○ 若选择**Nginx Igress**，则默认选中**安装Ingress组件**。关于Nginx Igress的更多使用说明，请参见Ingress高级用法。 ○ 若选择**ALB Ingress（公测）**，则默认安装ALB Ingress Controller组件。关于如何在ACK集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。

配置项	描述
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。 NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	选择**CSI**。安全沙箱仅支持CSI，Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-CSI。
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**安装node-problem-detector并创建事件中心**，您可以选择是否在日志服务控制台中添加事件中心。更多信息，请参见创建并使用Kubernetes事件中心。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

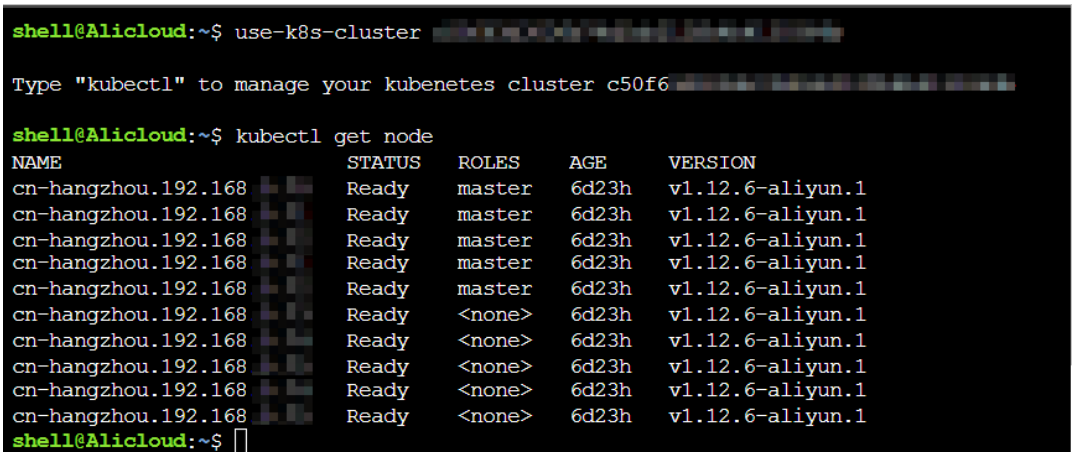
9. 单击**下一步：确认配置**，
10. 选中**服务协议**并单击**创建集群**。

> **说明** 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

执行结果

- 集群创建成功后，您可以在容器服务管理控制台的**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**详情**，单击**基本信息**和**连接信息**页签，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，可以通过此服务在客户端使用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。
  - **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` 。

    > **说明** 单击**重新绑定域名**，您可以重新绑定访问域名。
- 您可以通过kubectl工具连接集群，执行 `kubectl get node` 查看集群的节点信息。



27.6. 扩容安全沙箱集群

通过容器服务管理控制台，您可以根据实际业务需要对安全沙箱集群的Worker节点进行扩容。

前提条件

目前不支持集群中Master节点的扩容。

扩容安全沙箱集群时，以下界面参数需要按照表中的要求配置，否则扩容的集群节点无法运行安全沙箱容器。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 集群扩容**。
4. 在**节点池**页面单击目标节点右侧**操作**列的**扩容**。
5. 在扩容对话框配置参数，然后单击**提交**。

   本示例进行集群扩容，实现将集群的Worker节点数由3个扩容到5个。配置描述如下表。

配置项	描述
扩容数量	需要扩容的节点数量。
地域	默认显示集群所在的地域。
容器运行时	默认显示安全沙箱运行时。
专有网络	设置节点的网络。您可以在已有VPC列表中选择所需的VPC。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时， ○ 需设置**购买时长**。目前支持选择1、2、3、6个月和1~5年。 ○ 需设置是否**自动续费**。
实例规格	选择**神龙裸金属服务器（神龙）**。

配置项	描述
已选规格	选中的规格呈现在这里。
系统盘	支持SSD云盘、高效云盘和ESSD云盘。
挂载数据盘	支持SSD云盘、高效云盘和ESSD云盘。 **说明** 挂载数据盘时，支持云盘加密。挂载的数据盘用于存储节点上所有容器rootfs。因此，请至少挂载一块不小于200 GiB的数据盘，建议1 T以上。
登录方式 密钥对	○ 设置密钥。 您需要在创建集群的时候选择密钥对登录方式，单击**新建密钥对**，跳转到ECS云服务器控制台，创建密钥对，参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。 ○ 设置密码。 ■ **登录密码**：设置节点的登录密码。 ■ **确认密码**：确认设置的节点登录密码。
公网IP	选中**分配公网IPv4地址**后，弹出的节点自动具备公网IPv4地址。您可以通过这个IP直接访问该节点。关于公网IP地址详情，请参见弹性公网IP。
ECS标签	为ECS绑定标签。 **说明** ○ 键是必需的，而*值*是可选的，可以不填写。 ○ 键不能是aliyun、http://、https://开头的字符串，不区分大小写，最多64个字符。 ○ *值*不能是http://或https://，可以为空，不区分大小写，最多128个字符。 ○ 同一个资源，标签键不能重复，相同标签键（Key）的标签会被覆盖。 ○ 如果一个资源已经绑定了20个标签，已有标签和新建标签会失效，您需要解绑部分标签后才能再绑定新的标签。

配置项	描述
节点标签	为节点绑定标签。 **说明** ○ 键是必需的，而*值*是可选的，可以不填写。 ○ 键不能是aliyun、http://、https://开头的字符串，不区分大小写，最多64个字符。 ○ *值*不能是http://或https://，可以为空，不区分大小写，最多128个字符。 ○ 同一个资源，标签键不能重复，相同标签键（Key）的标签会被覆盖。 ○ 如果一个资源已经绑定了20个标签，已有标签和新建标签会失效，您需要解绑部分标签后才能再绑定新的标签。
自定义资源组	指定节点池所扩容节点的资源组信息。
污点（Taints）	为该节点添加污点。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。
云监控插件	选中在ECS节点上安装云监控插件，可以在云监控控制台查看所创建ECS实例的监控信息。
实例自定义数据	请参见ECS实例自定义数据概述。

后续步骤

扩容完成后，单击左侧导航栏中的**节点池**，查看节点池页面，您可以看到节点的数量从3变成了5。

27.7. 创建安全沙箱应用

您可以在安全沙箱Kubernetes集群中使用镜像创建一个可公网访问的nginx应用。

前提条件

创建一个安全沙箱集群。详情请参见创建安全沙箱托管版集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。
5. 在**无状态**页签中，单击**使用镜像创建**。
6. 在**应用基本信息**页签中，设置应用的基本信息，然后单击**下一步**。

   设置**应用名称**、**副本数量**、**类型**、**注解**、**标签**和**时区同步**，副本数量即应用包含的Pod数量。

> **说明**
> 本例中选择**无状态**类型，即Deployment类型。

7. 设置容器相关的配置。

> **说明** 在**容器配置**页签上方，单击**添加容器**为应用的Pod设置多个容器。

容器配置的配置项及描述如下。

- 基本配置

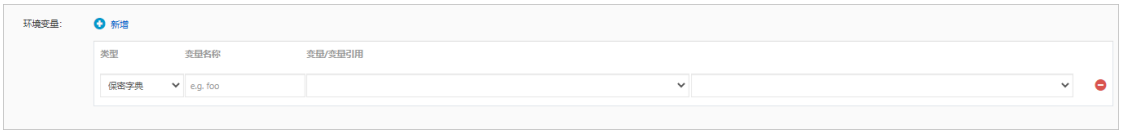


配置项	描述
镜像名称	您可以单击**选择镜像**，在弹出的对话框中选择所需的镜像并单击**确定**，本例中为Nginx。 您还可以填写私有镜像。填写的格式为 `domainname/namespace/imagename:tag` 。

配置项	描述
镜像Tag	■ 您可以单击**选择镜像Tag**选择镜像的版本。若不指定，默认为最新版。 ■ ACK支持以下三种镜像拉取策略（imagePullPolicy）： ■ **优先使用本地镜像（IfNotPresent）**：如果本地有该镜像（之前拉取过该镜像至宿主机中），则使用本地镜像，本地不存在时拉取镜像。 ■ **总是拉取镜像（Always）**：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。 ■ **仅使用本地镜像（Never）**：仅使用本地镜像。 **说明** 当您选中**镜像拉取策略**时，默认不设置镜像拉取策略。 ■ 单击**设置镜像密钥**，您可以实现免密拉取镜像。具体操作，请参见使用免密组件拉取容器镜像。
资源限制	可指定该应用所能使用的资源上限，包括CPU、内存和ephemeral-storage三种资源，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MiB；ephemeral-storage的单位为GiB。
所需资源	即为该应用预留资源额度，包括CPU和内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争夺资源，导致应用不可用。
容器启动项	■ stdin：将控制台输入发送到容器。 ■ tty：将标准输入控制台作为容器的控制台输入。
特权容器	■ 选择特权容器，则 `privileged=true` ，开启特权模式。 ■ 不选择特权容器，则 `privileged=false` ，关闭特权模式。
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，详情请参见Init Containers。

- （可选）端口设置
  设置容器的端口。
  - 名称：设置容器端口名称。
  - 容器端口：设置暴露的容器访问端口或端口名，端口号必须介于1~65535。
  - 协议：支持TCP和UDP。
- （可选）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。

- 类型：设置环境变量的类型，支持**自定义**、**配置项**、**密钥**和**变量/变量引用**、**资源引用**。配置项、密钥支持全部文件的引用，以密钥为例。
  选择**密钥**，选择**变量**，默认全部文件引用。



对应的yaml，则引用了整个secret。

```
envFrom:
  - secretRef:
      name: test
```

- 变量名称：设置环境变量名称。
- 变量/变量引用：设置变量引用的值。

- （可选）健康检查
  支持存活检查（liveness）、就绪检查（Readiness）和启动探测。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；启动探测用于检测容器内应用是否启动。关于健康检查的更多信息，请参见Configure Liveness, Readiness and Startup 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HTTP Headers，HTTP请求中自定义的请求头，HTTP允许重复的Header。支持键值对的配置方式。 ■ 延迟探测时间（秒）：即initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为3秒。 ■ 执行探测频率（秒）：即periodSeconds，指执行探测的时间间隔，默认为10秒，最小为1秒。 ■ 超时时间（秒）：即timeoutSeconds，探测超时时间。默认1秒，最小1秒。 ■ 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 ■ 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3，最小值是1。

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

- 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。具体参见配置生命周期。
  - **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。



- （可选）数据卷
  支持增加本地存储和云存储声明（PVC）。
  - **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。
  - **云存储声明（PVC）**：支持云存储。

  本例中配置了一个云存储类型的数据卷声明disk-ssd，将其挂载到容器的*/tmp*路径下。



- （可选）日志配置
  配置**日志服务**，您可进行采集配置和自定义Tag设置。

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

配置项	描述
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：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。

配置项	描述
自定义Tag	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。

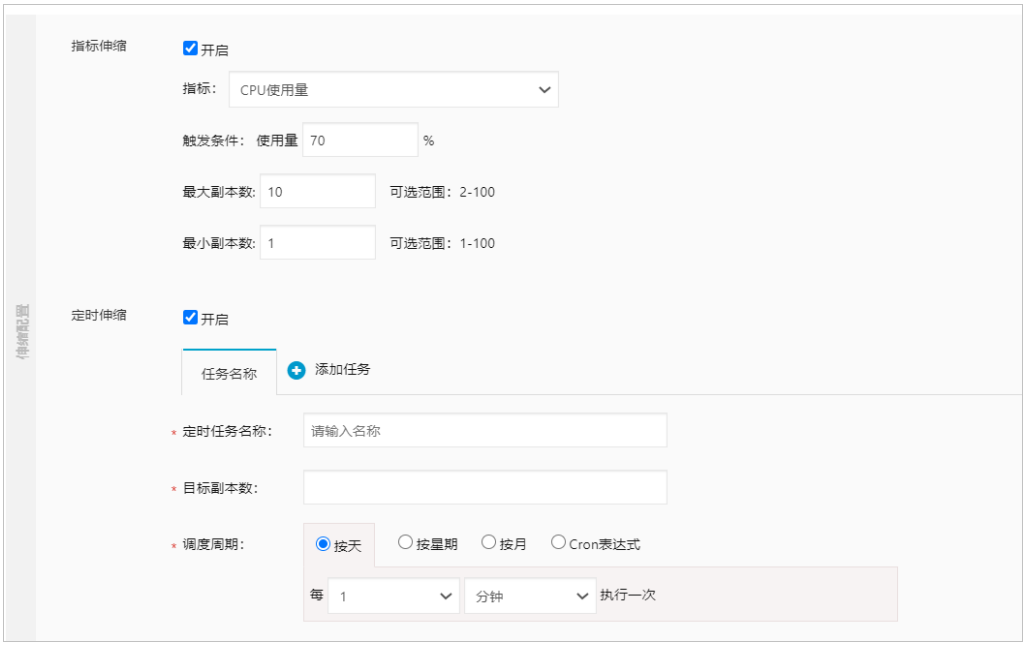
8. 完成容器配置后，单击**下一步**。
9. （可选）进行高级配置。
   - 访问设置

     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SLB（Server Load Balancer），该服务提供公网访问能力。
       - 创建路由（Ingress）：通过路由（Ingress）提供公网访问能力，详情请参见Ingress。

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

配置项	描述
服务（Service）	在**服务（Service）**右侧，单击**创建**设置创建服务配置项。配置项的详情，请参见管理服务。本例中服务类型选择**ClusterIP**。
路由（Ingress）	在**路由（Ingress）**右侧，单击**创建**。在弹出的对话框中，为后端Pod配置路由规则。详细的路由配置信息，请参见路由配置说明。 **说明** 通过镜像创建应用时，您仅能为一个服务创建路由（Ingress）。本例中使用一个虚拟主机名称作为测试域名，您需要在hosts中添加一条记录。在实际工作场景中，请使用备案域名。 `101.37.224.146 foo.bar.com #即ingress的IP`

     在**访问设置**区域中，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。
   - 伸缩配置
     在**伸缩配置**区域，配置是否开启**指标伸缩**和**定时伸缩**，从而满足应用在不同负载下的需求。



- 容器服务支持容器组的指标伸缩，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。

配置项	描述
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	该应用可扩容的容器数量上限。
最小副本数量	该应用可缩容的容器数量下限。

- 容器服务支持容器组的定时伸缩，即定时地对容器服务Kubernetes集群进行扩缩容。关于定时伸缩的详细说明，请参见创建定时伸缩CronHPA任务。

- 调度设置

  您可设置升级方式、节点亲和性、应用亲和性、应用非亲和性和调度容忍，详情请参见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。

配置项	描述
升级方式	升级方式包括滚动升级（Rollingupdate）和替换升级（Recreate），详细请参见详情请参见Deployments。

配置项	描述
节点亲和性	设置**节点亲和性**，通过Worker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Worker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。

应用亲和性	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。

配置项	描述
应用非亲和性	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（例如不同主机）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。
调度容忍	容忍被应用于Pod，允许这个Pod被调度到相对应的污点上。
调度到虚拟节点	设置是否调度到虚拟节点。如果您集群中没有虚拟节点，则无法设置该配置项。

- 标签和注释
  - Pod标签：为该Pod添加一个标签，标识该应用。
  - Pod注解：为该Pod添加一个注解（Annotation）。

10. 单击**创建**。
    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**查看nginx-deployment的详情。



11. 在集群管理页左侧导航栏中，单击**路由**。
    可以看到路由列表下出现一条规则。



执行结果

在浏览器中访问路由测试域名，您可访问nginx欢迎页。

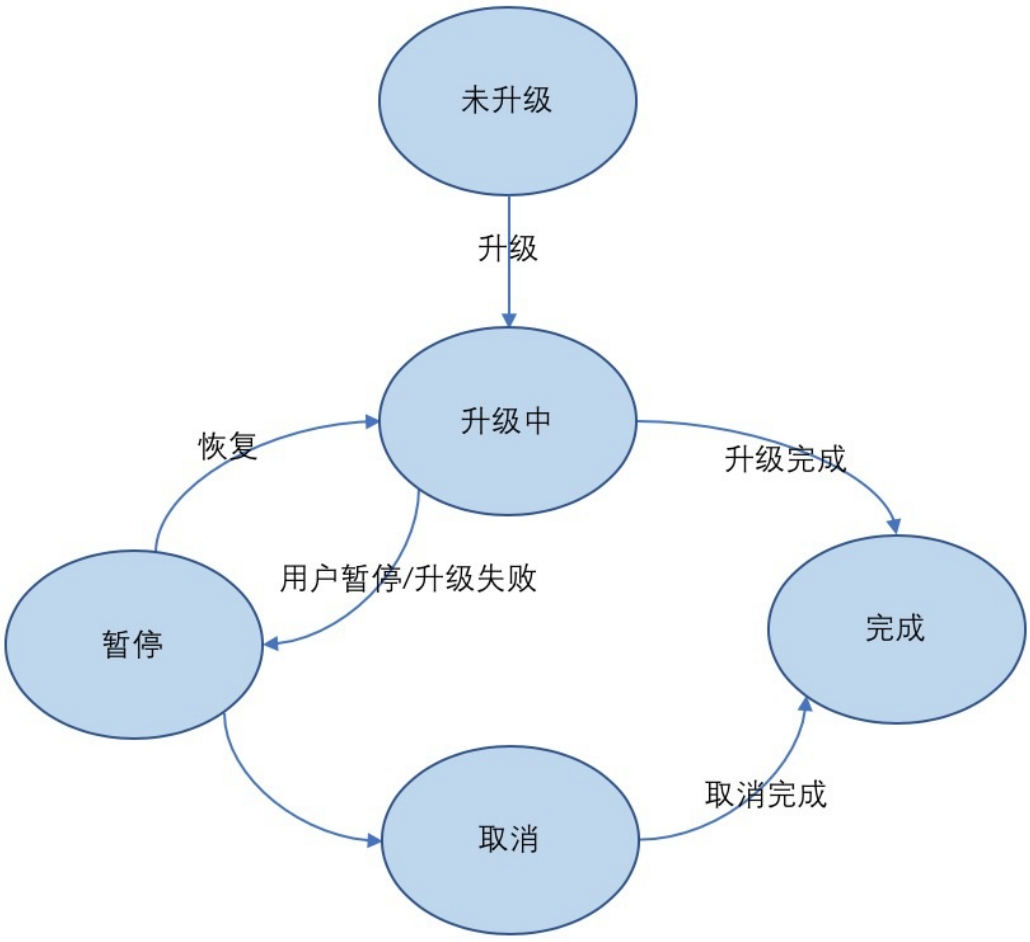


27.8. 升级安全沙箱运行时

您可以通过容器服务Kubernetes版ACK（Container Service for Kubernetes）控制台，可视化地升级您节点上的安全沙箱运行时。本文介绍安全沙箱运行时的升级原理、注意事项以及如何升级安全沙箱运行时。

功能原理

- 升级流程及状态转换
  当您执行升级操作后，系统会在后台创建并执行对应的升级任务，升级任务会自动分批次地向需要升级运行时的节点下发并执行升级Pod。在升级过程中，您可以根据需要，暂停、恢复甚至取消当前任务，如下图所示。



  - 当您单击**升级**按钮后，升级任务进入**升级**状态。
  - 您可以暂停**升级**状态的任务，任务进入**暂停**状态。
  - 您可以恢复**暂停**状态的任务，任务重新进入**升级**状态。

- 您也可以取消**暂停**状态的任务，任务进入**取消**状态。

- 升级策略
  运行时升级采用的是分批升级的策略，具体如下：
  - 第一批升级的节点数为1，后续的批次以2的幂数进行增长。暂停后重新恢复升级的第一批次为1，后续也是以2的幂数进行增长。
  - 每一批节点的最大数量不会超过需要升级运行时节点总数的10%。

  假如集群中，有50个节点的运行时可以升级，当您触发升级后：
  - 第一批次升级1个节点的运行时。
  - 第二批次升级2个节点的运行时。
  - 第三批次升级4个节点的运行时。
  - 第四批次升级5个节点的运行时（虽然2^3=8，但是集群只有50个节点，每批次最大为50×10%=5）。
  - 第五批次升级5个节点的运行时。
  - 以此类推。

- 暂停升级
  通过升级暂停功能，您可以在运行时升级的任意阶段对升级任务进行暂停。说明如下：
  - 暂停升级之后，当前批次已经开始升级的目标节点会继续执行运行时升级，还未开始升级的节点不会开始升级。
  - 运行时升级处于暂停状态时，集群也处于中间状态，建议您不要在此时对集群进行操作，并尽快完成升级过程。
  - 当您触发暂停时，系统会自动判断是否还有需要升级的目标节点，如果没有，则暂停无效，升级任务正常结束。

  您可以在运行时升级成功暂停之后，单击**继续**恢复当前的升级任务。
  如果运行时升级过程中发生错误，运行时升级任务会被系统自动暂停。具体失败原因会展示在页面下方详情中。您可根据报错进行排查或者提交工单。

- 取消升级
  您可以在暂停升级后，单击**取消**，对本次运行时升级任务进行取消操作。说明如下：
  - 取消升级之后，当前批次已经开始升级的目标节点会继续执行运行时升级，还未开始升级的节点不会开始升级。
  - 已经完成运行时升级的节点无法撤销升级逻辑。

注意事项

- 运行时升级需要节点具备公网访问能力，以便下载升级所需的软件包。
- 运行时升级过程中，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。
- 运行时升级过程中，集群中的应用不会中断。推荐您关注运行时版本的发布记录来决定是否需要重新发布应用，请参见安全沙箱运行时发布记录。
- 运行时升级过程中您可以在一批节点升级完成后暂停，此时整个集群处于升级的中间状态，建议您尽快完成升级过程。处于中间状态的升级任务会在15日之后关闭升级，同时清理一切升级相关的事件和日志信息。
- 请勿在运行时升级过程中添加或者删除节点。如果确实需要添加或者删除节点，请先取消升级。
- 运行时升级过程中，如非发生错误，请勿修改**runtime-upgrade**命名空间下面的相关资源。
- 如运行时升级失败，升级过程会暂停。您需要分析失败原因并清理**runtime-upgrade**命名空间下失败的Pod，确认修复成功后继续升级过程。

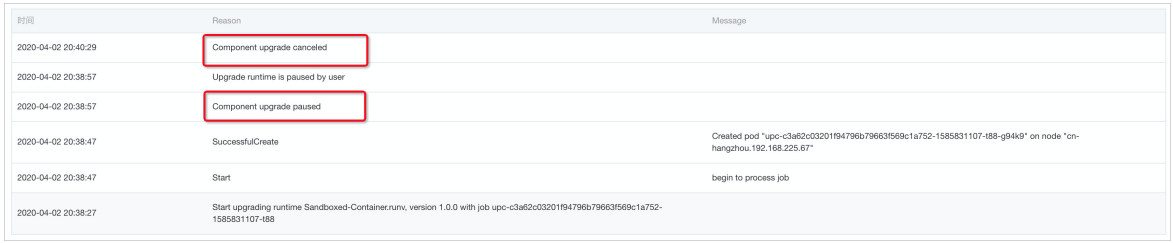
> **注意** 即使发生失败也不需要删除或修改**runtime-upgrade**命名空间下面的除Pod外的其他资源。如需帮助，请联系在线客服。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页面中，选择**运维管理 > 运行时升级**。
5. 在**运行时升级**页面，单击**升级**。
6. 在弹出的对话框中，单击**确定**。
   如果您需要暂停升级任务，单击**暂停**。任务暂停后，如果需要恢复或者取消升级，单击**继续**或者**取消**。



单击**升级**或者**继续**后，您可以在事件列表中查看操作内容。



执行结果

升级成功后，当您刷新页面或者重新进入时，已经升级的运行时不再显示待升级的列表中。

27.9. 安全沙箱配置

27.9.1. 配置安全沙箱节点和Docker节点的混合集群

节点池支持多种容器运行时，但一个节点池只能选一种容器运行时。您可以在一个集群创建不同容器运行时节点池，从而创建拥有不同容器运行时节点的混合集群。本文介绍如何在一个集群中混合配置安全沙箱节点池和Docker节点池。

前提条件

已创建一个ACK集群，请参见创建Kubernetes托管版集群。

> **注意** 创建的ACK集群需要符合以下条件。
>
> - Kubernetes版本大于等于1.14.6-aliyun.1。
> - 网络插件必须为Flannel或Terway非独占弹性网卡模式。
> - 存储插件不支持FlexVolume，只支持CSI-Plugin（版本大于或等于v1.14.8.39-0d749258-aliyun）。
> - logtail-ds版本大于或等于V0.16.34.2-f6647154-aliyun。

注意事项

- 默认情况下，每个集群中最多可包含100个节点。如果您需要添加更多节点，请提交工单申请，请参见提交工单。
- 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已配置NAT网关。同时您需要确保相应节点能正常访问公网，否则会导致添加云服务器失败。

创建Docker节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在**节点池**页面单击**创建节点池**，设置创建Docker运行时节点池的配置项。

   有关配置项的详细描述请参见创建Kubernetes托管版集群。需要注意的配置项如下：

配置项	描述
容器运行时	选择**Docker**，该节点池下的所有节点的容器运行时都是Docker。
数量	设置节点池初始节点数量。如不需要创建节点，可以设置为0。
操作系统	选择节点的操作系统，支持CentOS、AliyunLinux、Windows操作系统。

6. 单击**确认配置**。

创建安全沙箱节点池

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
5. 在节点池页面单击**创建节点池**，设置创建安全沙箱运行时节点池的配置项。

   有关配置项的详细描述请参见创建Kubernetes托管版集群。需要注意的配置项如下：

配置项	描述
容器运行时	选择**安全沙箱**，该节点池下的所有节点的容器运行时都是安全沙箱。
数量	设置节点池初始节点数量。如不需要创建节点，可以设置为0。
付费类型	安全沙箱运行时只能运行在神龙节点上，默认情况下神龙节点只支持包年包月付费类型。
挂载数据盘	请至少挂载一块不小于200GiB的数据盘。
操作系统	选择节点的操作系统，安全沙箱节点只支持AliyunLinux操作系统。

6. 单击**确认配置**。

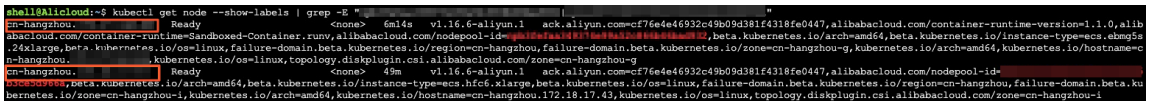
执行结果

- 您可以在**节点池**页面查看节点池状态，节点池状态显示**已激活**，表明节点池创建成功。
- 您可以连接ACK集群，查看节点的详细信息。
  i. 在**节点池**页面，单击新建的节点池名称，进入节点池详情页。在**节点池信息**区域查找并记录**节点池ID**。



  ii. 连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
  iii. 输入以下命令，获取节点名称。

```
kubectl get node --show-labels | grep -E "${节点池ID}|${节点池ID}"
```



  iv. 输入以下命令，查看节点详情。

```
kubectl get node -o wide | grep -E "${节点名}|${节点名}"
```



27.9.2. 配置安全沙箱Pod内核参数

通常容器提供的默认内核参数配置能够满足大部分业务的日常需求，但在某些特殊的业务场景中为了让应用达到最佳的运行状态，往往需要优化部分内核参数配置。安全沙箱支持通过Annotation为Pod配置自定义内核参数，也支持开启特权容器模式后在Pod内通过Sysctl设置内核参数。本文主要介绍如何为安全沙箱Pod配置内核参数。

前提条件

- 创建安全沙箱托管版集群。
- 通过kubectl工具连接集群。

背景信息

在Kubernetes中可通过Pod的SecurityContext和Annotation来配置命名空间的安全和非安全Sysctl参数，除此之外还可以启动特权容器在容器内配置非命名空间的Sysctl参数。

由于runC Pod与主机共享内核，存在以下问题：

- 虽然特权容器几乎拥有与主机进程一样的访问权限，但在修改Sysctl参数或非安全的Sysctl参数时，可能会影响其他Pod或者导致节点不可用等问题。
- 无法为同一节点上的不同Pod分别定制非命名空间的内核配置。

安全沙箱核心特点

安全沙箱Pod拥有独享内核。通过Annotation为Pod配置自定义内核参数或开启特权容器模式，可以解决runC Pod由于共享内核存在的问题：

- 通过Annotation可以为Pod配置所有合法的Sysctl参数，包括命名空间和非命名空间Sysctl参数，这些配置仅对Pod所在的内核有效，不会影响其他Pod和宿主机的Sysctl配置。
- 通过开启特权容器模式，可以使容器几乎享有沙箱内核的所有访问权限。由于沙箱独享内核，容器的访问权限仅限于沙箱内部，不会对其他Pod和宿主机的内核有影响。

通过Pod Annotation定制Sysctl参数

通过 Annotation securecontainer.alibabacloud.com/sysctls 配置Pod的Sysctl参数，取值可以是一个或多个Sysctl参数，多个Sysctl参数用逗号分隔。

比如当需要开启网络包转发功能时，可通过以下Annotation配置Sysctl参数：

```
annotations:
 securecontainer.alibabacloud.com/sysctls: "net.bridge.bridge-nf-call-ip6tables=1,net.bridge.bridge-nf-call-iptables=1,net.ipv4.ip_forward=1"
```

完整的YAML示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  app: busybox
 name: busybox
 namespace: default
spec:
 replicas: 2
 selector:
  matchLabels:
   app: busybox
 template:
  metadata:
   labels:
    app: busybox
   annotations:
    securecontainer.alibabacloud.com/sysctls: "net.bridge.bridge-nf-call-ip6tables=1,net.bridge.bridge-nf-call-iptables=1,net.ipv4.ip_forward=1"
  spec:
   containers:
    - command:
      - sleep
      - infinity
     image: 'busybox:latest'
     imagePullPolicy: IfNotPresent
     name: busybox
     resources:
      limits:
       cpu: '1'
       memory: 1Gi
      requests:
       cpu: 500m
       memory: 512Mi
   dnsPolicy: ClusterFirst
   restartPolicy: Always
   runtimeClassName: runv
```

配置特权容器

安全沙箱Pod和runC Pod配置特权容器的方式相同。他们的区别在于runC Pod配置的是主机的内核，安全沙箱Pod配置的是沙箱的内核。

可通过以下命令配置特权容器：

```
containers:
 - name: busybox
   securityContext:
     privileged: true
```

完整的YAML示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
   app: busybox
 name: busybox
 namespace: default
spec:
 replicas: 2
 selector:
   matchLabels:
     app: busybox
 template:
   metadata:
     labels:
       app: busybox
   spec:
     containers:
       - command:
           - sh
           - -c
           - "sysctl -w fs.inotify.max_user_watches=524286 && tail -f /dev/null"
         image: 'busybox:latest'
         imagePullPolicy: IfNotPresent
         name: busybox
         securityContext:
           privileged: true
         resources:
           limits:
             cpu: '1'
             memory: 1Gi
           requests:
             cpu: 500m
             memory: 512Mi
     dnsPolicy: ClusterFirst
     restartPolicy: Always
     runtimeClassName: runv
```

相关文档

- 创建安全沙箱托管版集群
- runC容器和安全沙箱（runV）容器的区别
- 通过kubectl工具连接集群

27.10. 安全沙箱存储

27.10.1. 安全沙箱的NAS直挂

您可以通过安全沙箱的NAS直挂功能，有效地解决IO性能问题。本文介绍安全沙箱NAS直挂的实现原理，并通过示例说明如何使用安全沙箱的NAS直挂功能。

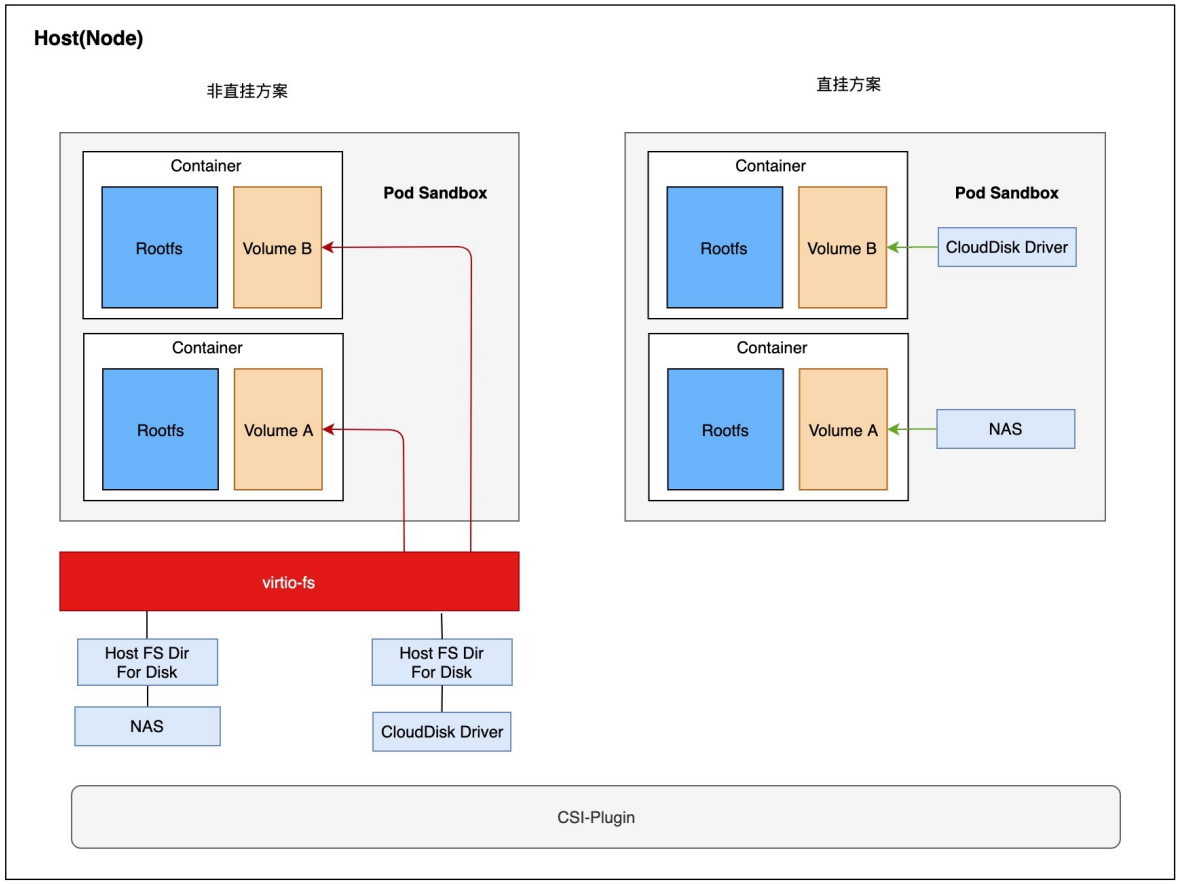
前提条件

- 创建安全沙箱托管版集群
- 通过kubectl工具连接集群

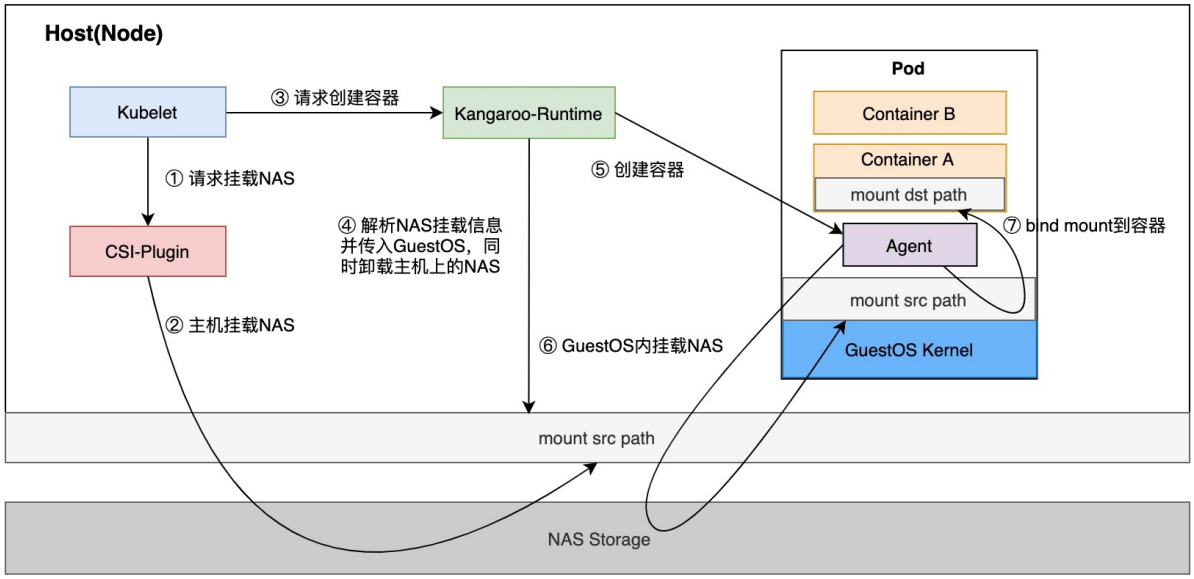
背景信息

virtio-fs是一个共享的文件系统。容器服务Kubernetes版（ACK）安全沙箱使用virtio-fs可以把Volume、Secret、ConfigMap等共享到虚拟机Guest OS内，从而可以用原生的方式通过Volume挂载NAS。但这种方式下，NAS是挂载到主机上的，在容器内要经过virtio-fs读写主机上的NAS，由此会带来一些性能损耗。

安全容器提供了安全沙箱NAS直挂的功能。该功能会先卸载主机上的NAS挂载点，然后在Guest OS内挂载NAS，最后把NAS bind mount到容器内，从而能够在容器内直接读写NAS，达到接近原生的性能。



实现原理



实现安全沙箱的NAS直挂功能的流程说明如下。

步骤序号	说明
①	Kubelet请求CSI-Plugin挂载NAS卷。
②	CSI-Plugin在主机挂载NAS。
③	Kubelet请求Kangaroo-Runtime创建容器。
④	Kangaroo-Runtime解析NAS挂载信息并传入GuestOS，同时卸载主机上的NAS。
⑤	Kangaroo-Runtime请求Agent创建容器。
⑥	Agent把NAS挂载到GuestOS内。
⑦	Agent把GuestOS上的NAS bind mount到容器内。

示例

以下通过创建NAS实例及使用YAML模板创建资源对象举例说明如何在安全沙箱容器中挂载NAS卷。

1. 创建一个NAS实例。具体操作，请参见通过控制台创建通用型NAS文件系统。

   > **注意** NAS实例的VPC需要和集群的VPC一致。

   获取到NAS实例挂载点地址，如下图**挂载命令**中的*file-system-id.region.nas.aliyuncs.com*即为挂载点地址。



> **说明** *file-system-id.region.nas.aliyuncs.com*：表示挂载点地址。您可以在NAS控制台中，单击模板文件系统名称，在**挂载使用**页面获取挂载点地址。

2. 执行以下命令，使用模板创建资源对象。

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: nas-pvc-csi
 namespace: default
spec:
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 5Gi
 selector:
  matchLabels:
   alicloud-pvname: nas-pv-csi
---
apiVersion: v1
kind: PersistentVolume
metadata:
 labels:
  alicloud-pvname: nas-pv-csi
 name: nas-pv-csi
spec:
 accessModes:
  - ReadWriteMany
 capacity:
  storage: 5Gi
 csi:
  driver: nasplugin.csi.alibabacloud.com
  volumeAttributes:
   options: noresvport,nolock
   path: /csi
   server: ${nas-server-address}
   vers: "3"
  volumeHandle: nas-pv-csi
 persistentVolumeReclaimPolicy: Retain
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deploy-nas-csi
spec:
 replicas: 2
 selector:
  matchLabels:
   app: busybox
 template:
  metadata:
   labels:
    app: busybox
   annotations:
    storage.alibabacloud.com/enable_nas_passthrough: "true"
```

```
  spec:
    runtimeClassName: runv
    containers:
      - name: busybox
        image: registry.cn-hangzhou.aliyuncs.com/acs/busybox:v1.29.2
        command:
        - tail
        - -f
        - /dev/null
        volumeMounts:
          - name: nas-pvc
            mountPath: "/data"
    restartPolicy: Always
    volumes:
      - name: nas-pvc
        persistentVolumeClaim:
          claimName: nas-pvc-csi
EOF
```

> **说明**
>
> - 您需要把模板中的 ${nas-server-address} 替换成NAS实例挂载点地址。
>
>   ```
>   server: ${nas-server-address}
>   ```
>
> - 默认Pod不启用NAS直挂功能，您需要在模板中通过Annotation开启NAS直通功能。
>
>   ```
>   annotations:
>       storage.alibabacloud.com/enable_nas_passthrough: "true"
>   ```

3. 执行以下命令，进入Pod内验证挂载点类型。

   ```
   kubectl get pods
   ```

   ```
   kubectl exec -it ${podid} sh
   ```

   ```
   mount | grep /data | grep nfs
   ```

   如果执行命令后有内容，表明NAS直挂功能成功。

27.10.2. 安全沙箱的云盘直挂

您可以通过安全沙箱的云盘直挂功能，有效地解决IO性能问题。本文介绍安全沙箱云盘直挂的实现原理，并通过示例说明如何使用安全沙箱的云盘直挂功能。

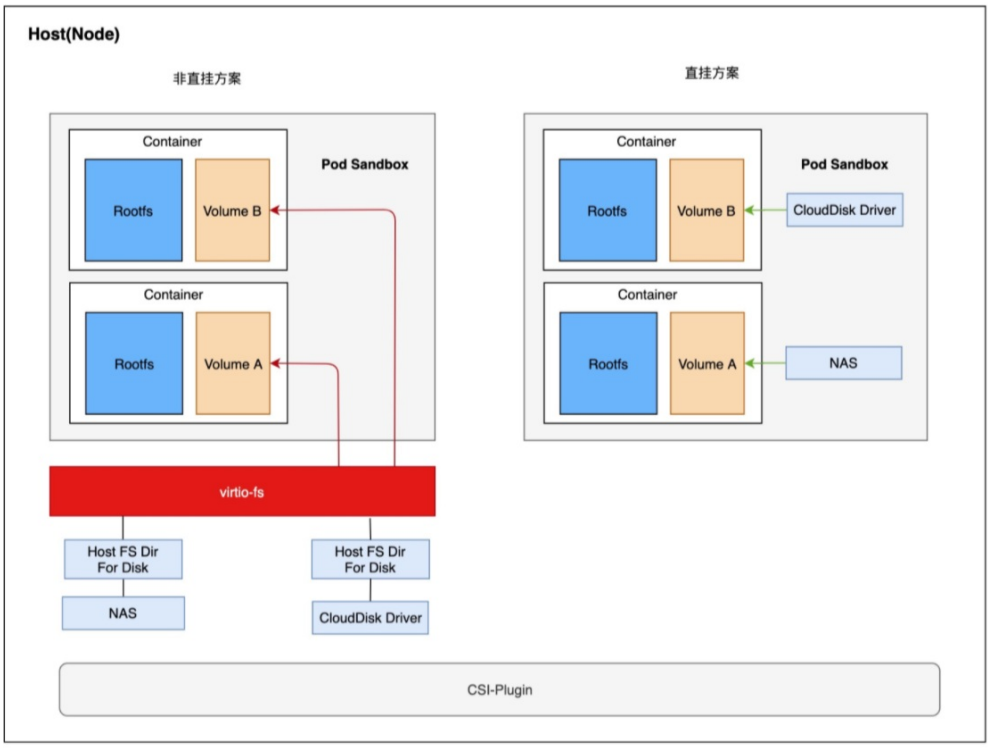
前提条件

- 创建安全沙箱托管版集群
- 通过kubectl工具连接集群

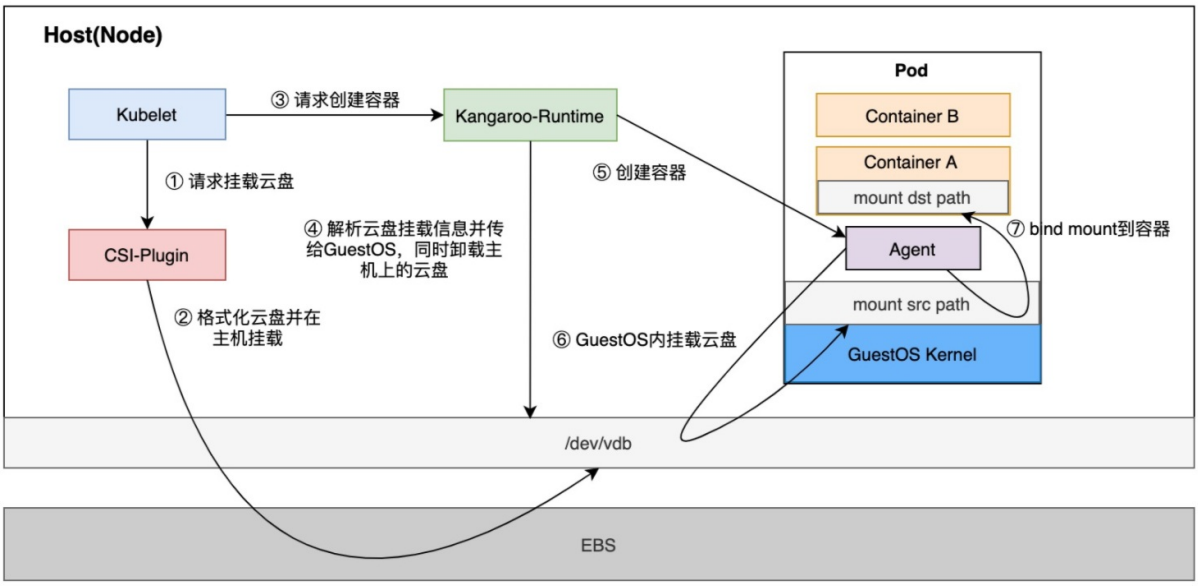
背景信息

virtio-fs是一个共享的文件系统。容器服务Kubernetes版（ACK）安全沙箱使用virtio-fs可以把Volume、Secret、ConfigMap等共享到虚拟机Guest OS内，从而可以用原生的方式通过Volume挂载云盘。但这种方式下，云盘是挂载到主机上的，在容器内要经过virtio-fs才能读写主机上的云盘，由此会带来一些性能损耗。

安全容器提供了安全沙箱云盘直挂的功能。该功能会先卸载主机上的云盘挂载点，然后在Guest OS内挂载云盘，最后把云盘bind mount到容器内，从而能够在容器内直接读写云盘，达到接近原生的性能。



实现原理



实现安全沙箱的云盘直挂沙箱功能的流程说明如下。

步骤序号	说明
①	Kubelet请求CSI-Plugin挂载云盘。
②	CSI-Plugin格式化云盘，然后在主机上挂载云盘。
③	Kubelet请求Kangaroo-Runtime创建容器。
④	Kangaroo-Runtime解析云盘挂载信息并传给GuetOS，同时卸载主机上的云盘。

步骤序号	说明
⑤	Kangaroo-Runtime请求Agent创建容器。
⑥	Agent把云盘挂载到GuestOS内。
⑦	Agent把GuestOS上的云盘bind mount到容器内。

示例

以下通过使用YAML模板创建资源对象举例说明如何在沙箱容器中挂载云盘。

1. 执行以下命令，使用模板创建资源对象。

```
cat <<EOF | kubectl create -f -
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-disk-ssd
parameters:
 type: cloud_ssd
provisioner: diskplugin.csi.alibabacloud.com
reclaimPolicy: Delete
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: disk-pvc-01
 namespace: default
spec:
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
   storage: 25Gi
 storageClassName: alicloud-disk-ssd
---
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  app: busybox
 name: busybox
 namespace: default
spec:
 replicas: 1
 selector:
  matchLabels:
   app: busybox
 template:
  metadata:
   labels:
    app: busybox
   annotations:
    storage.alibabacloud.com/enable_ebs_passthrough: "true"
  spec:
   containers:
```

```
    - name: busybox
      image: registry.cn-hangzhou.aliyuncs.com/acs/busybox:v1.29.2
      command:
      - tail
      - -f
      - /dev/null
      volumeMounts:
       - mountPath: "/data"
         name: disk-pvc
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  runtimeClassName: runv
  volumes:
   - name: disk-pvc
     persistentVolumeClaim:
      claimName: disk-pvc-01
EOF
```

> **说明**
> 默认Pod不启用云盘直挂功能，您需要在模板中通过Annotation开启云盘直通功能。
>
> ```
> annotations:
>     storage.alibabacloud.com/enable_ebs_passthrough: "true"
> ```

2. 执行以下命令，进入Pod内验证挂载点文件系统类型。

```
kubectl get pods
```

```
kubectl exec -it ${podid} sh
```

```
mount | grep /data | grep -vi virtio_fs
```

如果挂载类型不是virtio-fs，说明成功实现云盘直挂功能。

27.11. 安全沙箱兼容性说明

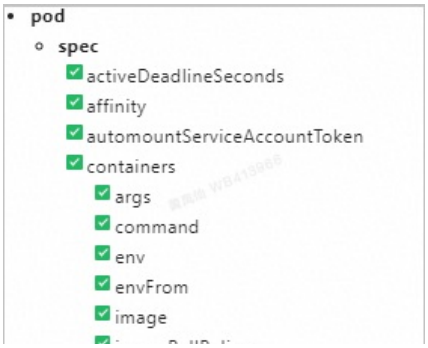
本文列举安全沙箱运行时兼容的Pod字段来帮助您更高效地使用安全沙箱运行时。

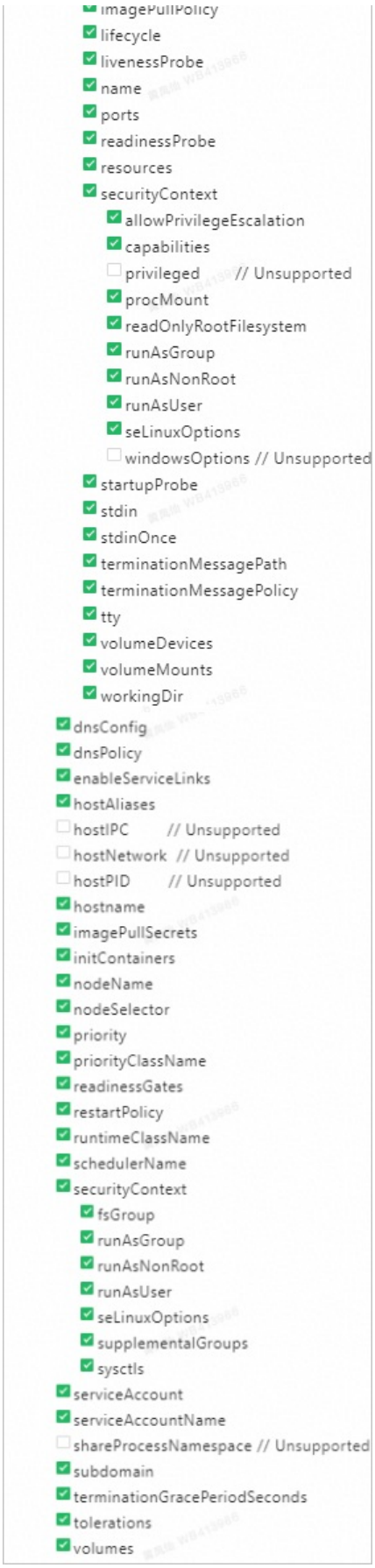
背景信息

安全沙箱作为一种新的容器运行时，除了在创建安全沙箱（runV）容器时不支持Pod层面的个别字段之外，其他兼容性与runC容器一致，如Pod（Container）网络、Service网络（ClusterIP、NodePort）、镜像等等。用户在使用安全沙箱容器运行时，无需改变任何开发模式、镜像打包等。

Pod兼容字段列表

在Pod API层面，安全沙箱运行时兼容的字段如下图所示（勾选的字段为兼容字段）。




imagePullPolicy
lifecycle
livenessProbe
name
ports
readinessProbe
resources
securityContext
allowPrivilegeEscalation
capabilities
privileged // Unsupported
procMount
readOnlyRootFilesystem
runAsGroup
runAsNonRoot
runAsUser
seLinuxOptions
windowsOptions // Unsupported
startupProbe
stdin
stdinOnce
terminationMessagePath
terminationMessagePolicy
tty
volumeDevices
volumeMounts
workingDir
dnsConfig
dnsPolicy
enableServiceLinks
hostAliases
hostIPC // Unsupported
hostNetwork // Unsupported
hostPID // Unsupported
hostname
imagePullSecrets
initContainers
nodeName
nodeSelector
priority
priorityClassName
readinessGates
restartPolicy
runtimeClassName
schedulerName
securityContext
fsGroup
runAsGroup
runAsNonRoot
runAsUser
seLinuxOptions
supplementalGroups
sysctls
serviceAccount
serviceAccountName
shareProcessNamespace // Unsupported
subdomain
terminationGracePeriodSeconds
tolerations
volumes

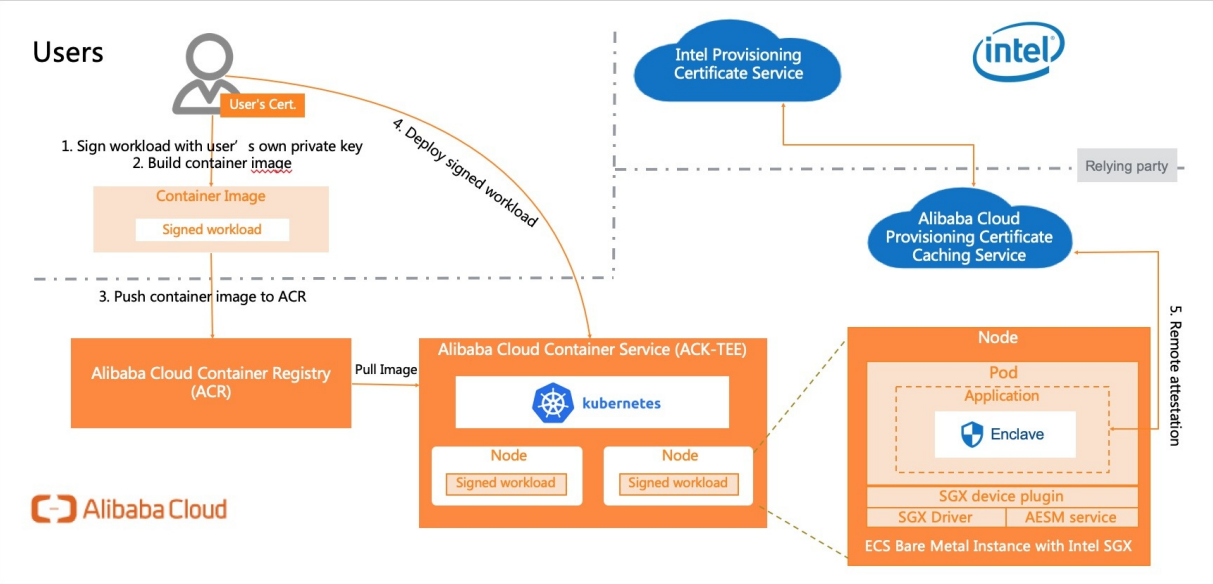
28.ACK-TEE机密计算

28.1. ACK-TEE机密计算介绍

本文介绍ACK-TEE（ACK-Trusted Execution Environment）机密计算的概念、核心功能、适用场景、产品方案，以及ACK-TEE和ACK安全沙箱的配合作用等。

基础概念

为有强安全诉求的用户提供了基于硬件加密技术的云原生一站式机密计算容器平台，它可以帮助您保护数据使用（计算）过程中的安全性、完整性和机密性，同时简化了可信或机密应用的开发、交付和管理成本。机密计算可以让您把重要的数据和代码放在一个特殊的可信执行加密环境（Trusted Execution Environment，TEE）中，而不会暴露给系统其他部分。其他应用、BIOS、OS、Kernel、管理员、运维人员、云厂商、甚至除了CPU以外的其他硬件均无法访问机密计算平台数据，极大减少敏感数据的泄露风险，为您提供了更好的控制、透明度和隐秘性。



核心功能

- 保护并验证云端代码和数据的完整性。
- 加密数据和代码，避免在使用过程中被恶意窥探和窃取。
- 保障您对数据全生命周期的控制。

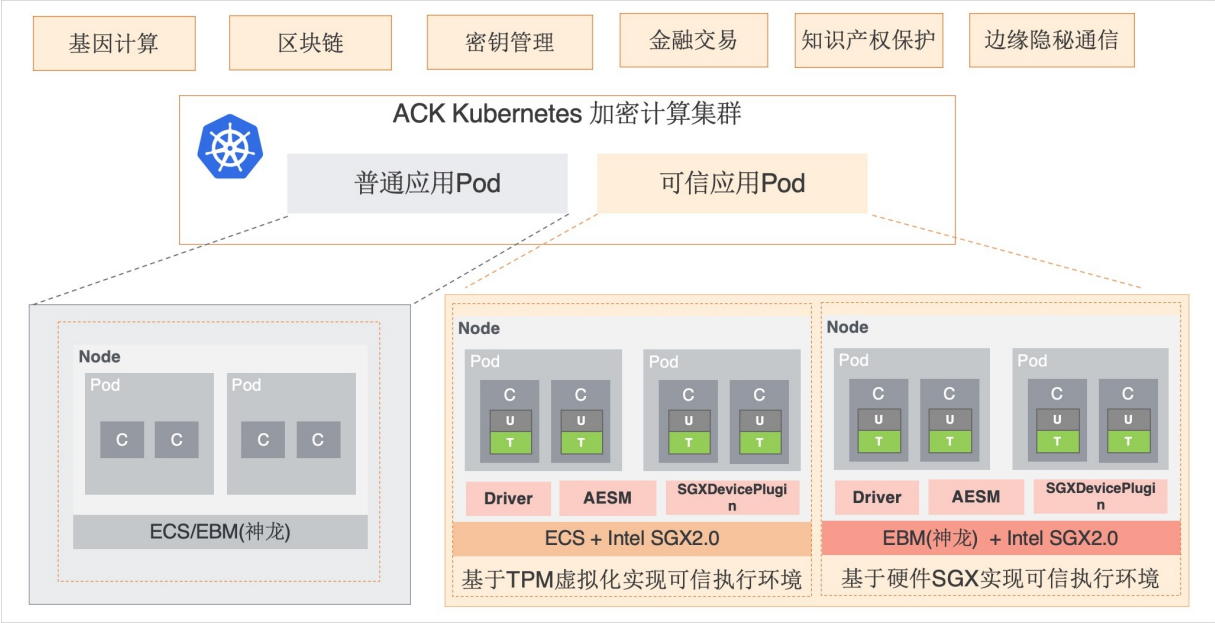
适用场景

- **区块链**
  增强事务处理、共识、智能合约以及密钥存储的隐私性和安全性。
- **密钥管理**
  把密钥管理功能放在Enclave中，提供类似硬件安全模块（HSM）的功能。
- **基因计算**
  敏感数据，对敏感数据的多方计算提供了隐私隔离。
- **金融**
  保护支付、交易的安全性。
- **AI**
  保护模型等核心机密数据，保护知识产权。
- **边缘计算**
  保护云、边、端三位一体相互通信的安全以及隐秘性。
- **数据共享与计算**

不同用户或厂商之间相互共享数据以便挖掘更大的数据经济价值，但不想把自己的数据泄露给对方。

产品方案

ACK-TEE1.1版的产品方案图如下所示。



ACK已经上线了基于Intel SGX2.0技术的加密计算托管集群，以便帮助您简化可信或机密应用的管理和交付成本。在公有云处理数据和代码时，保护您数据和代码的完整性和隐秘性，从可信计算基础中摆脱云厂商的影响。有关如何在ACK上直接创建加密计算托管集群，请参见创建加密计算托管集群。

创建加密计算托管集群说明如下：

- Worker节点需选择**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的型号。

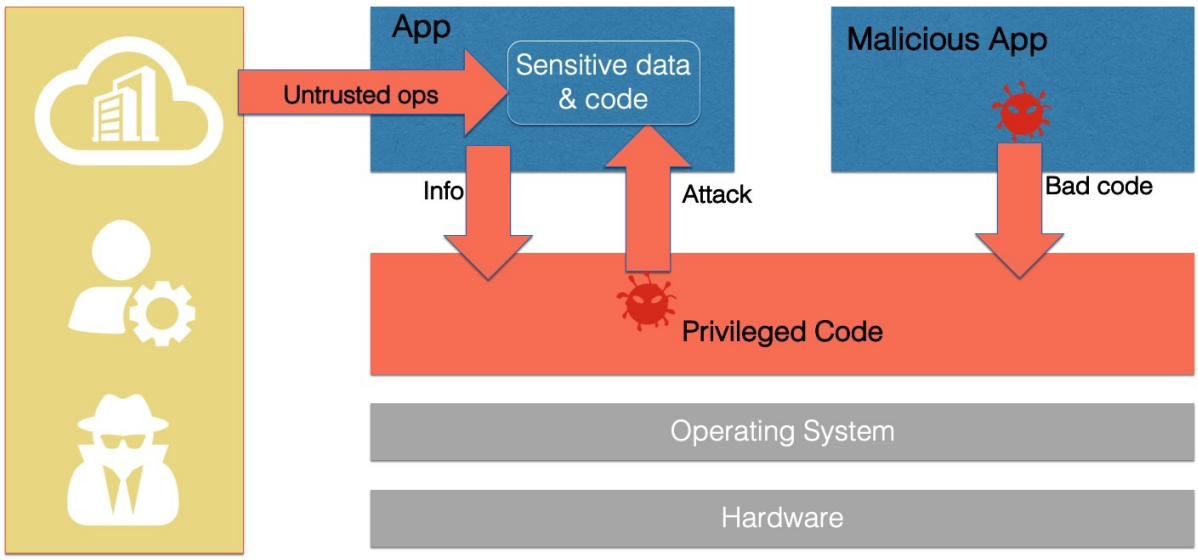
  > **说明** Intel IceLake仅支持基于Intel SGX DCAP的远程证明方式，不支持基于Intel EPID方式的远程证明方式，您的程序可能需要适配后才能正常使用远程证明功能。关于远程证明的更多信息，请参见attestation-services。

- 节点初始化时会自动安装SGX2.0驱动以及TEE-SDK。TEE SDK是阿里云提供的开发加密计算应用的开发者套件，该套件提供与Intel Linux SGX SDK一致的开发模型和编程接口。
- 为了便于SGX2.0应用访问AESM，集群默认安装了AESM DaemonSet。
- 通过自研的sgx-device-plugin，您可轻松实现对SGX节点EPC内存资源的发现、管理以及调度。

ACK-TEE机密计算和ACK安全沙箱配合工作的应用场景

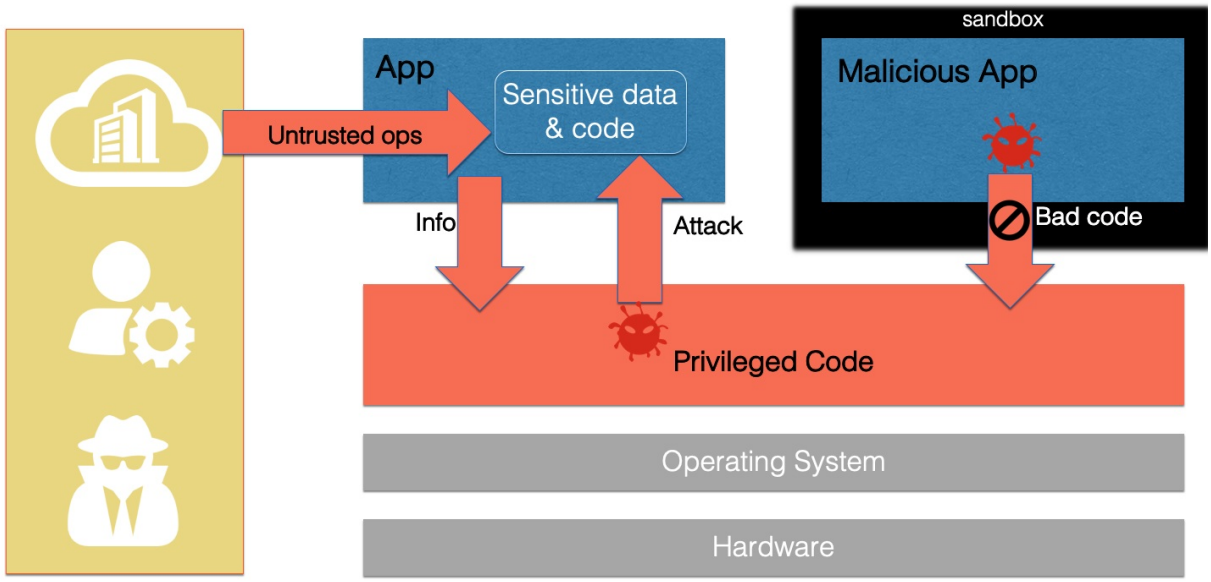
传统OS（操作系统）容器攻击模型

传统OS容器（或称为RunC容器）和宿主机共享Kernel，当内核出现漏洞，容器中恶意应用（如三方或您的应用）会利用漏洞逃逸并渗透到后端系统，危害其他应用和整个服务系统。



ACK安全沙箱隔离恶意应用并阻断攻击

ACK安全沙箱容器是基于轻量虚拟机技术（Kangaroo）实现的强隔离。每个Pod都有自己独立的OS和Kernel，这样恶意应用的攻击面从宿主机级别缩小至Pod级别，从而保护其他应用和后端系统。

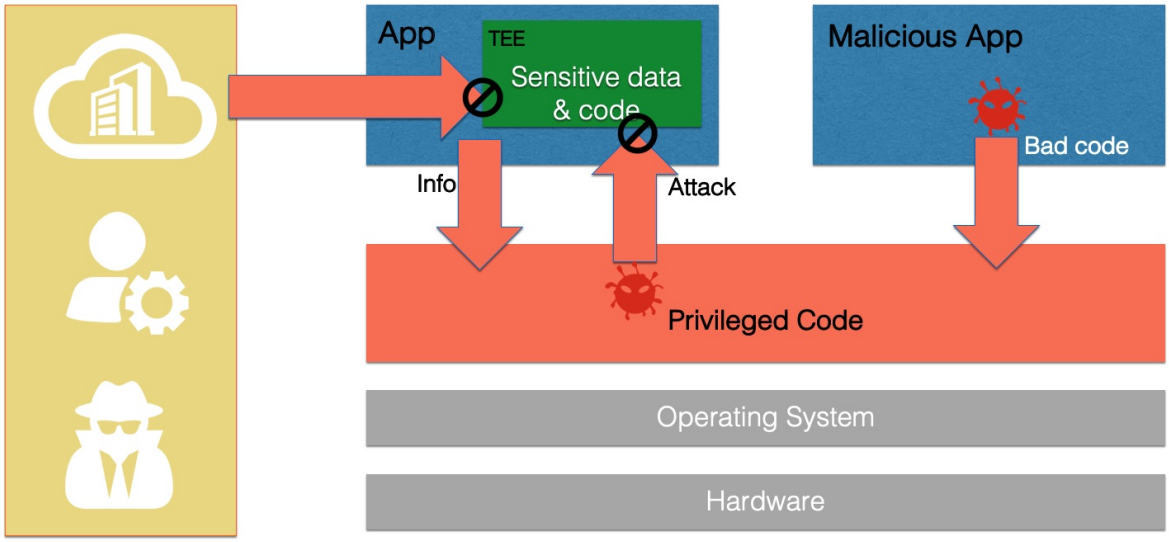


ACK-TEE加密保护运行中的应用

ACK-TEE是ACK基于Kubernetes提供的机密计算解决方案，用于保护您的敏感代码和数据，如IP保护、密钥保护、隐秘通信等。

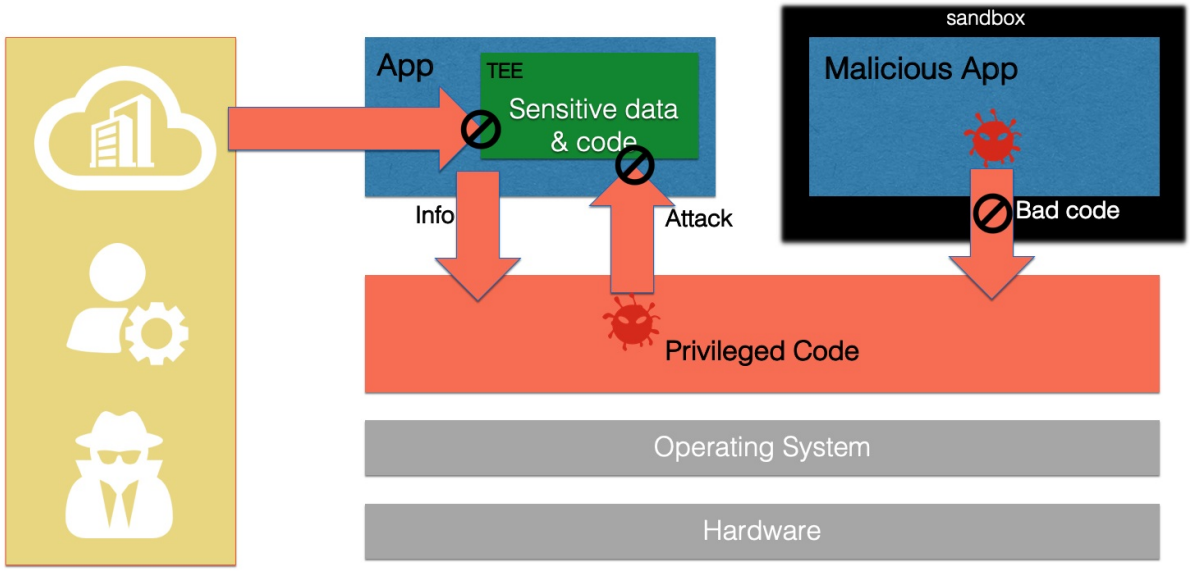
云计算为您和企业提供了极大的便利，然而数据泄露成了上云过程中最大的担忧之一。数据泄露可能来源：

- 黑客攻击
- 云厂商的不可信任
- 云基础设施安全缺陷
- 运维和管理人员



安全沙箱和ACK-TEE配合隔离恶意应用并保护敏感应用

ACK安全沙箱和ACK-TEE定位完全不同，它们之间可以相互配合，这样在您的集群中，既可以隔离恶意应用，又可以保护敏感的应用和数据，做到真正的双向保护。



相关文档

- 创建加密计算托管集群
- 创建安全沙箱托管版集群

28.2. 创建加密计算托管集群

本文介绍如何在控制台创建阿里云容器服务安全加密计算托管集群（Alibaba Cloud Container Service for Kubernetes-Trusted Execution Environment，简称ACK-TEE）。

前提条件

您需要开通容器服务和访问控制（RAM）服务。

登录容器服务管理控制台和RAM管理控制台开通相应的服务。

> **说明**
> 您在使用集群过程中，请注意以下限制：
>
> - 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。
> - ACK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。如果您需要提高配额，请提交工单申请。
>   - 关于ACK集群配额限制的详情，请参见ACK集群配额限制。
>   - 每个账户初始默认状况下VPC路由条目不超过48条，意味着您的ACK集群的网络模式是Flannel时，集群的路由条目最大不能超过48个（网络模式是Terway则不受该影响）。如集群需要更多路由条目数，您需要对目标VPC提交工单申请提高配额 。
>   - 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。
> - 创建加密计算Kubernetes集群时，以下界面参数需要按照表中的要求配置，否则创建的集群将无法支持运行Intel SGX应用。
>
> | 参数 | 配置 |
> | --- | --- |
> | **可用区** | 目前只有**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例规格支持加密计算集群，请确保所选可用区有这些实例规格。 |
> | **容器运行时** | Containerd 1.4.4或更高版本。 |
> | **实例规格** | 选择**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例规格。
说明 Intel IceLake仅支持基于Intel SGX DCAP的远程证明方式，不支持基于Intel EPID方式的远程证明方式，您的程序可能需要适配后才能正常使用远程证明功能。关于远程证明的更多信息，请参见attestation-services。 |
> | **操作系统类型** | **AliyunLinux 2.xxxx 64位UEFI版。** |
> | **网络插件** | **Flannel。** |

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击页面右上角的**集群模板**。
4. 在**选择集群模板**页面的**托管集群**区域选择**加密计算托管集群**，并单击**创建**。
5. 在**ACK托管版**页面，完成集群配置。
   i. 完成集群基础选项配置。

配置项	描述

配置项	描述
集群名称	填写集群的名称。 **说明** 集群名称应包含1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
集群规格	选择集群规格，支持**标准版**和**Pro版**。
地域	选择集群所在的地域。
付费类型	容器服务Kubernetes版支持**按量付费**和**包年包月**两种节点付费类型。选择**包年包月**时，需设置以下参数。 **说明** 设置**付费类型**为**包年包月**时，仅ECS云服务器和SLB负载均衡按照包年包月的方式计费，其他云资源仍然按照按量付费的方式付费。关于云资源的详细介绍，请参见云产品资源计费。 ■ **购买时长**：目前支持选择1、2、3、6个月和1~3年。 ■ **自动续费**：设置是否自动续费。
账号全部资源组	将鼠标悬浮于页面上方的**账号全部资源**，选择资源组。在控制台页面顶部选择的资源组可过滤出该资源组内的专有网络及对应的虚拟交换机。在创建集群时，只显示过滤的专有网络实例及专有网络对应的虚拟交换机实例。 
Kubernetes版本	显示当前ACK支持的Kubernetes版本。
加密计算集群	选中**开启**。
容器运行时	容器运行时仅支持Containerd。更多信息，请参见如何选择Docker运行时、Containerd运行时、或者安全沙箱运行时？。
专有网络	设置集群的网络，您可以选择普通VPC和共享VPC。 ■ 共享VPC：VPC的所有者账号（资源所有者）可以将其账号下的VPC内的交换机资源共享给其组织内的其他账号使用。 ■ 普通VPC：不具备共享功能的VPC。 **说明** 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。如果没有您需要的专有网络，可以通过单击**创建专有网络**进行创建，请参见创建和管理专有网络。
虚拟交换机	设置虚拟交换机。 您可以在已有虚拟交换机列表中，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，请参见使用交换机。
网络插件	加密计算仅支持**Flannel**。

配置项	描述
Pod网络CIDR	网络插件选择**Flannel**时，需要配置**Pod网络CIDR**。 **Pod网络CIDR**指定Flannel网络插件需要配置Pod网络CIDR，网段不能和VPC及VPC已有Kubernetes集群使用的网段重复，创建成功后不能修改。而且Service地址段不能和Pod地址段重复，有关Kubernetes网络地址段规划的信息，请参见Kubernetes集群网络规划。
Service CIDR	设置**Service CIDR**。您需要指定**Service CIDR**，网段不能与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，请参见Kubernetes集群网络规划。
节点IP数量	如果您选择的网络模式为**Flannel**，您需设置**节点IP数量**。 **说明** ■ **节点IP数量**是指可分配给一个节点的IP数量，建议保持默认值。 ■ 根据您所选择的专有网络和节点IP数量，ACK将为您推荐可用的Pod网络CIDR和Service CIDR，并给出相应配置下集群内可允许部署的主机数量以及每台主机可容纳的Pod数量。请您根据集群规模的实际需求，在推荐配置的基础上进行修改。
配置SNAT	创建集群时，默认不开通公网。如果您选择的VPC不具备公网访问能力，选中**为专有网络配置SNAT**后，ACK将为您创建NAT网关并自动配置SNAT规则。
API Server访问	ACK默认为API Server创建一个内网SLB实例，您可修改SLB实例规格。更多信息，请参见实例规格。 **注意** 删除默认创建的SLB实例将会导致无法访问API Server。 您可设置是否开放**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，ACK会创建一个EIP，并挂载到SLB上。此时，Master节点的6443端口（对应API Server）暴露出来，您可以在外网通过kubeconfig连接并操作集群。 ■ 如果选择不开放，则不会创建EIP，您只能在VPC内部用kubeconfig连接并操作集群。
RDS白名单	设置RDS白名单。将节点IP添加到RDS实例的白名单中。 **说明** 允许白名单RDS访问Kubernetes集群，RDS必须在当前集群的VPC内。

配置项	描述
安全组	支持选择**自动创建普通安全组**、**自动创建企业级安全组**、**选择已有安全组**。有关安全组的详细内容，请参见安全组概述。 说明 ■ 只有白名单用户可以使用**选择已有安全组**功能。请提交工单申请。 ■ 指定已有安全组时，系统默认不会为安全组配置额外的访问规则，可能会导致访问异常，请自行管理安全组规则。关于如何管理安全组规则，请参见最小化集群访问规则。
集群删除保护	设置是否启用集群删除保护。为防止通过控制台或API误释放集群。
资源组	创建的集群将归属于选择的资源组。一个资源只能归属于一个资源组。根据不同的业务场景，您可以将资源组映射为项目、应用或组织等概念。更多信息，请参见资源组。

ii. 完成集群高级选项配置。

配置项	描述
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。
标签	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。
集群本地域名	设置是否配置集群本地域名。 **说明** 默认域名为**cluster.local**，可自定义域名。域名由两段组成，每段不超过63个字符，且只能使用大小写字母和数字，不能为空。
自定义证书SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。 具体操作，请参见自定义集群API Server证书SAN。
服务账户令牌卷投影	开启**服务账户令牌卷投影**功能以降低在Pod中使用Service Account遇到的安全性问题，可使得kubelet支持基于Pod粒度的Token签发，并且支持Token audience和过期时间的配置。详情请参见部署服务账户令牌卷投影
Secret落盘加密	根据需要选择是否勾选**选择KMS密钥**。

6. 单击**下一步：节点池配置**，完成Worker节点配置。

i. 选择Worker实例。

> **说明** 加密计算场景下，Worker节点只有**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例规格支持加密计算集群运行Intel SGX2.0应用。

- 如果您选择**新增实例**，需要进行以下配置。

配置项	描述
节点池名称	填写节点池名称。 > **说明** 节点池名称为1~63个字符，可包含数字、汉字、英文字符或短划线（-）。
实例规格	目前仅**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例规格支持运行Intel SGX2.0应用。
已选规格	已选的实例规格，可以选一个或多个规格。
数量	新增Worker实例（ECS实例）的数量。
系统盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。 > **说明** > - 支持选中**开启云盘备份**以备份云盘数据。 > - **ESSD云盘**支持自定义**性能级别**。ESSD云盘容量越大，可供选择的性能级别越高（460 GiB容量以上可选PL2，1260 GiB以上可选PL3）。更多信息，请参见容量范围与性能级别的关系。
挂载数据盘	支持**ESSD云盘**、**SSD云盘**和**高效云盘**。挂载数据盘时，支持云盘**加密**和**开启云盘备份**。节点的数据盘加密时只能使用默认的CMK。
操作系统	仅支持AliyunLinux 2.xxxx 64位UEFI版。

登录方式
密钥对
	- 设置密钥。
	  - **密钥对**：如您已经创建密钥对，在下拉列表中选择目标密钥对。
	  - **新建密钥对**：此项用于您还未创建密钥对。创建密钥对，请参见创建SSH密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。
	- 设置密码。
	  - **登录密码**：设置节点的登录密码。
	  - **确认密码**：确认设置的节点登录密码。
	> **说明** 密码为8~30个字符，且必须同时包含三项（大写字母、小写字母、数字和特殊符号），其中特殊字符不包括下划线（_）。

- 如果您选择**添加已有实例**，则至少添加两个以上的Worker实例。您需要配置**操作系统**、**登录方式**和**密钥对**，配置描述参见上文。

ii. 设置高级选项。

配置项	描述
实例保护	设置是否启用实例保护。 **说明** 为防止通过控制台或API误释放集群节点，默认启用实例保护。
实例自定义数据	自定义脚本，在节点初始化之后会执行该脚本。 **说明** Windows支持bat和powershell两种格式，在Base64编码前，第一行为[bat]或者 [powershell]。Linux支持shell脚本，更多的格式请参见cloud-init和ECS实例自定义数据概述。 如果您使用的自定义脚本长度大于1 KB，建议您将脚本上传到OSS，通过OSS内网端点拉取脚本执行。
自定义镜像	**注意** 不使用自定义镜像，否则无法保证提供加密计算服务。
自定义节点名称	是否开启**自定义节点名称**。 节点名称由前缀，节点IP地址子串及后缀三部分组成： ■ 前缀和后缀均可由.分隔的一个或多个部分构成，每个部分可以使用小写字母、数字和短划线（-），且首尾必须为小写字母和数字。 ■ IP地址段长度指截取节点IP地址末尾的位数，取值范围5~12。
CPU Policy	设置CPU policy。 ■ none：默认策略，表示启用现有的默认CPU亲和方案。 ■ static：允许为节点上具有某些资源特征的Pod赋予增强的CPU亲和性和独占性。
污点（Taints）	为集群内所有Worker节点添加污点。

7. 单击**下一步：组件配置**，完成组件配置。

配置项	描述
Ingress	设置是否安装Ingress组件。默认**不安装**。 ○ 若选择**Nginx Igress**，则默认选中**安装Ingress组件**。关于Nginx Igress的更多使用说明，请参见Ingress高级用法。 ○ 若选择**ALB Ingress（公测）**，则默认安装ALB Ingress Controller组件。关于如何在ACK集群中通过ALB Ingress访问服务，请参见通过ALB Ingress访问服务。 **说明** 如果您勾选**创建Ingress Dashboard**，则需要同步安装日志服务组件。

配置项	描述
服务发现	设置是否安装NodeLocal DNSCache组件，默认安装NodeLocal DNSCache。NodeLocal DNSCache用于运行DNS缓存代理以提升域名解析性能和稳定性。关于NodeLocal DNSCache的更多信息，请参见在ACK集群中使用NodeLocal DNSCache。
存储插件	设置存储插件，支持Flexvolume和CSI。Kubernetes集群通过Pod可自动绑定阿里云云盘、NAS、OSS存储服务。请参见存储管理-Flexvolume和存储管理-CSI。
监控插件	设置是否启用云监控插件。默认选中**在ECS节点上安装云监控插件**和**使用Prometheus监控服务**，前者用于在云监控控制台查看所创建ECS实例的监控信息。
报警配置	选中**使用默认报警模板配置报警**，开启默认报警规则。详细介绍，请参见容器服务报警管理。
日志服务	设置是否启用日志服务，您可使用已有Project或新建一个Project。默认选中**使用日志服务**。创建应用时，您可通过简单配置，快速使用日志服务，详情参见通过日志服务采集Kubernetes容器日志。 默认选中**安装node-problem-detector并创建事件中心**，您可以选择是否在日志服务控制台中添加事件中心。更多信息，请参见创建并使用Kubernetes事件中心。
工作流引擎	设置是否使用AGS。 **说明** 当前只有白名单用户可以使用该功能。 ○ 如果选中AGS，则创建集群时系统自动安装AGS工作流插件。 ○ 如果不选中，则需要手动安装AGS工作流插件，请参见AGS命令行帮助。

8. 单击**下一步：确认配置**，确认配置信息并选中**服务协议**。
9. 单击**创建集群**，启动部署。

   您还可以看到集群创建的日志信息。

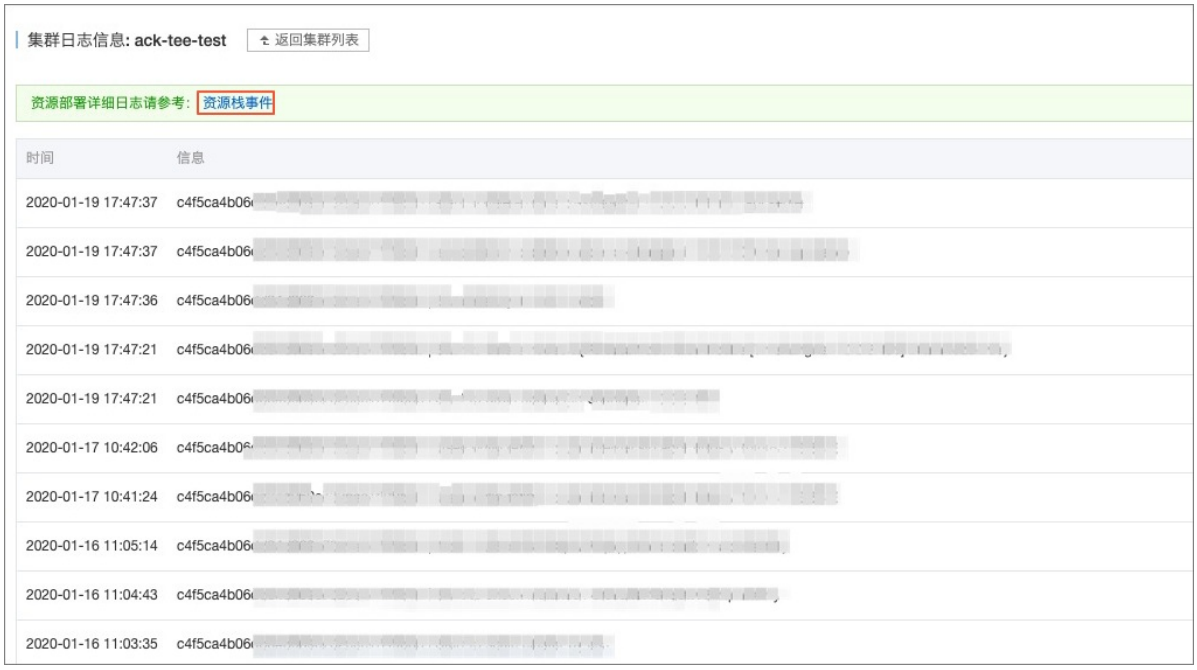
   说明 一个包含多节点的Kubernetes集群的创建时间一般约为十分钟。

执行结果

- 集群创建成功后，您可以在容器服务管理控制台的**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，单击集群名称或者操作列中的**详情**，然后单击**基本信息**和**连接信息**页签，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。
    绑定EIP和**解绑EIP**功能仅支持托管版Kubernetes集群。
    - 绑定EIP：您可以选择在已有EIP列表中绑定EIP或者新建EIP。
      绑定EIP操作为导致API Server短暂重启，请避免在此期间操作集群。
    - 解绑EIP：解绑EIP后您将无法通过公网访问API Server。
      解绑EIP操作会导致API Server短暂重启，请避免在此期间操作集群。
  - **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。
  - **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` 。

    > **说明** 单击**重新绑定域名**，您可以重新绑定访问域名。

- 您可以通过kubectl工具连接集群，执行 `kubectl get node` 查看集群的节点信息。

```
使用 Aliyun CLI 来管理云资源        |  cloudshell://tutorial/aliyun-cli

For more tutorials, visit https://api.aliyun.com/#/lab
shell@Alicloud:~$ source use-k8s-cluster c4f5ca4b06dd34d969e732cee1f7f9bd5

Type "kubectl" to manage your kubenetes cluster c4f5ca4b06dd34d969e732cee1f7f9bd5

shell@Alicloud:~$ kubectl get node
NAME                       STATUS   ROLES    AGE    VERSION
cn-shanghai.[redacted]     Ready    <none>   3d6h   v1.14.8-aliyun.1
cn-shanghai.[redacted]     Ready    <none>   3d6h   v1.14.8-aliyun.1
cn-shanghai.[redacted]     Ready    <none>   3d5h   v1.14.8-aliyun.1
cn-shanghai.[redacted]     Ready    <none>   3d5h   v1.14.8-aliyun.1
cn-shanghai.[redacted]     Ready    <none>   3d5h   v1.14.8-aliyun.1
shell@Alicloud:~$
```

相关文档

- ACK-TEE机密计算介绍
- 通过SDK开发和构建SGX2.0应用

28.3. 创建加密计算节点池

通过创建加密计算节点池使已有集群具备ACK TEE机密计算能力，ACK TEE机密计算可以让您把重要数据和代码放在一个特殊的可信执行加密环境中，避免在使用过程中被恶意窥探和窃取，减少敏感数据泄露风险。本文介绍如何创建加密计算节点池。

前提条件

- 已创建一个托管版ACK集群，请参见创建Kubernetes托管版集群。且创建的集群必须符合以下要求：
  - 集群的网络插件必须选择Flannel。
  - 集群的容器运行时必须选择Containerd。
- 所在的Region和可用区可以购买到**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例规格。

> **说明** Intel IceLake仅支持基于Intel SGX DCAP的远程证明方式，不支持基于Intel EPID方式的远程证明方式，您的程序可能需要适配后才能正常使用远程证明功能。关于远程证明的更多信息，请参见attestation-services。

背景信息

ACK-TEE机密计算是阿里云容器服务Kubernetes版ACK（Container Service for Kubernetes）基于Intel SGX2.0提供的可信应用或用于交付和管理机密计算应用的云原生一站式机密计算平台，帮助您保护数据使用中的安全性、完整性和机密性。机密计算可以让您把重要的数据和代码放在一个特殊的可信执行加密环境（Trusted Execution Environment，TEE）中，而不会暴露给系统其他部分。其他应用、BIOS、OS、Kernel、管理员、运维人员、云厂商、甚至除了CPU以外的其他硬件均无法访问机密计算平台数据，极大减少敏感数据的泄露风险，为您提供了更好的控制、透明度和隐秘性。您可以在符合条件的已有Kubernetes托管版集群中，通过新建ACK-TEE机密计算节点池，快速为集群引入机密计算能力。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在集群管理页左侧导航栏中，选择**节点管理 > 节点池**。
4. 在**节点池**页面右上角，单击**创建节点池**。

   您还可以在**节点池**页面右上角，单击**创建托管节点池**或者**自动弹性伸缩配置**创建托管节点池或者具有弹性功能的节点池。

5. 在**创建节点池**页面，设置创建节点池的配置项。

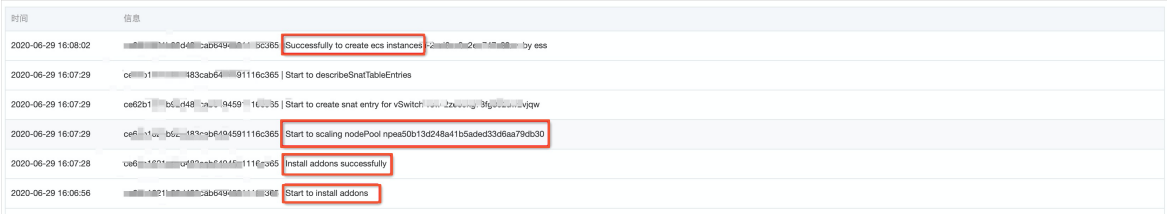
   有关配置项的详细说明，请参见创建Kubernetes托管版集群。以下为创建机密计算节点池的重点配置项说明。

配置项	描述
加密计算集群	选择**加密计算**。 
容器运行时	只能选择Containerd运行时。
自动伸缩	选择是否开启自动弹性伸缩。开启后，节点池会根据资源使用自动弹性扩容节点。
实例规格	选择**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例规格。
数量	设置节点池初始节点数量。如不需要创建节点，可以填写为0。
操作系统	只能选择AliyunLinux 2.xxxx 64位UEFI。
节点标签	您可以为集群节点添加标签。
ECS标签	您可以为ECS实例添加标签。

6. 单击**确认配置**。

   在**节点池**页面，如果节点池**状态**显示**初始化中**，则说明节点池正在创建中。

   在**集群列表**页面，单击目标集群**操作**列下的**查看日志**，在集群日志页面可以查看详细的创建加密计算节点池的日志信息。



   创建完成后，**状态**显示为**已激活**。



后续步骤

创建机密计算节点池成功后，您可以创建并部署SGX2.0应用，详情请参见通过SDK开发和构建SGX2.0应用。

28.4. 通过SDK开发和构建SGX2.0应用

本文介绍如何基于TEE-SDK开发和构建，并部署一个名为 `helloworld` 的SGX2.0应用，这个应用在可信区周期性地生产消息并传递给不可信代码（缓冲区），然后输出到终端。

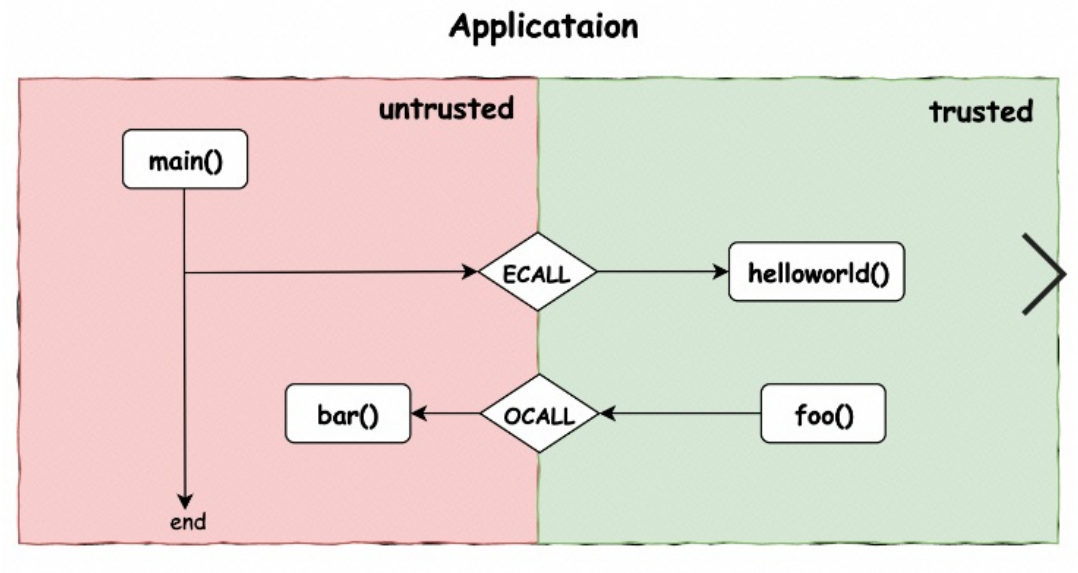
前提条件

- 创建加密计算托管集群。
- 开发编译环境依赖：
  - Alibaba Cloud Linux 2。
  - Intel SGX Driver。

- TEE-SDK：是阿里云提供的开发加密计算应用的开发者套件，该套件提供与Intel Linux SGX SDK一致的开发模型和编程接口，兼容Intel SGX SDK和Intel SGX PSW。
- AESM service。

基本原理

SGX程序原理图



SGX程序基本原理说明如下：

- SGX2.0应用由两部分组成：
  - untrusted不可信区
    如果代码和数据运行在普通非加密内存区域，程序 `main` 入口必须在非可信区。SGX程序原理图中的 `main()` 和 `bar()` 函数均在非可信区。
  - trusted可信区
    如果代码和数据运行在硬件加密内存区域，此区域由CPU创建且只有CPU有权限访问。SGX程序原理图中的 `helloworld()` 和 `foo()` 函数运行在可信区。
- 非可信区只能通过ECALL函数调用可信区内的函数。
- 可信区只能通过OCALL函数调用非可信区的函数。
- ECALL函数和OCALL函数通过EDL文件声明。

示例

本示例中SGX2.0应用 `helloworld` 所涉及的应用编译、镜像构建以及部署等示例源码位于github。

示例代码目录结构

```
sgx-device-plugin/samples/hello_world/
                ├── Dockerfile
                ├── Makefile
                ├── README.md
                └── src
                    ├── App
                    │   ├── App.cpp
                    │   └── App.h
                    ├── Enclave
                    │   ├── Enclave.config.xml
                    │   ├── Enclave.cpp
                    │   ├── Enclave.edl
                    │   ├── Enclave.h
                    │   ├── Enclave.lds
                    │   └── Enclave_private.pem
                    └── Makefile
```

`src` 代码目录和文件说明如下：

- App目录存放不可信区域代码，包括main入口、OCALL函数内具体逻辑代码等。
- Enclave目录存放可信区域代码，包括ECALL函数内具体逻辑代码。

文件	说明
Enclave.edl	EDL(Enclave Description Language)文件。
Enclave.lds	Enclave linker script。
Enclave_private.pem	enclave.so的签名密钥。
Enclave.config.xml	Enclave配置文件，如堆栈大小、是否开启Debug等。
Enclave.h和Enclave.cpp	应用安全区代码实现。

1. 在命令行工具连接您创建的加密计算托管集群并编译hello_world。

   在项目目录下执行命令。

   ```
   export SGX_SDK=/opt/alibaba/teesdk/intel/sgxsdk make
   ```

   在项目根目录下会生成一个名为*hello_world*的Binary文件。

```
GEN => App/Enclave_u.h
CC  <= App/Enclave_u.c
CXX <= App/App.cpp
LINK => app
GEN => Enclave/Enclave_t.h
CC  <= Enclave/Enclave_t.c
CXX <= Enclave/Enclave.cpp
LINK => enclave.so
<EnclaveConfiguration>
  <ProdID>0</ProdID>
  <ISVSVN>0</ISVSVN>
  <StackMaxSize>0x40000</StackMaxSize>
  <HeapMaxSize>0x100000</HeapMaxSize>
  <TCSNum>10</TCSNum>
  <TCSPolicy>1</TCSPolicy>
  <!-- Recommend changing 'DisableDebug' to 1 to make the enclave undebuggable for enclave release -->
  <DisableDebug>0</DisableDebug>
  <MiscSelect>0</MiscSelect>
  <MiscMask>0xFFFFFFFF</MiscMask>
</EnclaveConfiguration>
tcs_num 10, tcs_max_num 10, tcs_min_pool 1
The required memory is 3960832B.
The required memory is 0x3c7000, 3868 KB.
Succeed.
SIGN => enclave.signed.so
The project has been built in debug hardware mode.
```

执行命令 `./hello_world` 运行*hello_world*应用。

```
./hello_world
```

返回结果如下：

```
Wed May  6 06:53:33 2020
Hello world From SGX Enclave!
Wed May  6 06:53:34 2020
Hello world From SGX Enclave!
...
```

编译流程和编译后的代码目录结构说明如下：

- 编译基本流程（Makefile）。
  a. 通过**sgx_edger8r**工具在*App/*目录下生成不可信代码（`Enclave_u.c` 和 `Enclave_u.h`），这部分生成的代码主要会调用 `ECALL（sgx_ecall）` 。
  b. 编译不可信部分Binary：*app*。
  c. 通过**sgx_edger8r**工具在*Enclave/*目录下生成可信代码（ `Enclave_t.c` 和 `Enclave_t.h` ）。
  d. 编译可信动态链接库（enclave.so）。
  e. 通过**sgx_sign**工具签名可信动态链接库（enclave.signed.so）。
  f. 结束。
- 编译后的代码目录结构。

```
sgx-device-plugin/samples/hello_world/src/
                        ├──── hello_world    #[generated]
                        ├──── App
                        │   ├──── App.cpp
                        │   ├──── App.h
                        │   ├──── App.o      #[generated]
                        │   ├──── Enclave_u.c #[generated]
                        │   ├──── Enclave_u.h #[generated]
                        │   └──── Enclave_u.o #[generated]
                        ├──── Enclave
                        │   ├──── Enclave.config.xml
                        │   ├──── Enclave.cpp
                        │   ├──── Enclave.edl
                        │   ├──── Enclave.h
                        │   ├──── Enclave.lds
                        │   ├──── Enclave.o    #[generated]
                        │   ├──── Enclave_private.pem
                        │   ├──── Enclave_t.c  #[generated]
                        │   ├──── Enclave_t.h  #[generated]
                        │   └──── Enclave_t.o  #[generated]
                        ├──── enclave.signed.so #[generated]
                        ├──── enclave.so      #[generated]
                        └──── Makefile
```

2. 构建镜像并部署 helloworld 应用。

   编译出的应用程序必须基于alibabatee/centos_sgx:7构建镜像，镜像里面已安装了SGX程序所依赖的动态链接库。
   Dockerfile文件内容如下。

   ```
   FROM alibabatee/centos_sgx:7
   COPY src/hello_world src/enclave.signed.so /app/
   WORKDIR /app
   ENTRYPOINT ["/app/hello_world"]
   ```

   i. 执行以下命令编译并构建镜像。

   ```
   cd sgx-device-plugin/samples/hello_world
   TARGET_IMAGE=registry-vpc.cn-shanghai.aliyuncs.com/${namespace}/${image_name}:${image_tag} make image
   docker push registry-vpc.cn-shanghai.aliyuncs.com/${namespace}/${image_name}:${image_tag}
   ```

ii. 执行以下命令部署 helloworld 应用。

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
 name: helloworld
 namespace: default
spec:
 replicas: 2
 selector:
  matchLabels:
   app: helloworld
 template:
  metadata:
   labels:
    app: helloworld
  spec:
   containers:
   - command:
    - /app/hello_world
    image: {{TARGET_IMAGE}}
    imagePullPolicy: Always
    name: helloworld
    resources:
     limits:
      cpu: 250m
      memory: 512Mi
      alibabacloud.com/sgx_epc_MiB: 2
    volumeMounts:
    - mountPath: /var/run/aesmd/aesm.socket
     name: aesmsocket
   volumes:
   - hostPath:
     path: /var/run/aesmd/aesm.socket
     type: Socket
    name: aesmsocket
EOF
```

代码文件说明

文件路径	说明	示例代码

文件路径	说明	示例代码

Encalve/Enclave.edl	EDL中声明了一个公共ECALL函数，每个SGX应用的EDL必须至少声明一个：
	• `public` 类型的ECALL函数。
	• `trusted {...}` 内声明ECALL函数。
	• `untrusted {...}` 内申明OCALL函数。
	由于本例中安全区不需要向非安全区调用（OCALL），所以只声明了一个ECALL函数 `ecall_hello_from_enclave` ，这个ECALL函数目的是在安全区创建一个Buffer并填充 `helloworld` ，然后这个Buffer的内容拷贝到非安全的Buffer中，非安全区调用 `printf` 打印这个非安全Buffer内容。
		enclave {
		trusted {
		public void ecall_hello_from_enclave([out, size=len] char* buf, size_t len);
		};
		};

Enclave/Enclave.lds	无。	enclave.so
		{
		global:
		g_global_data_sim;
		g_global_data;
		enclave_entry;
		g_peak_heap_used;
		local:
		*;
		};

Enclave/Enclave.config.xml	无。	<EnclaveConfiguration> <ProdID>0</ProdID> <ISVSVN>0</ISVSVN> <StackMaxSize>0x40000</StackMaxSize> <HeapMaxSize>0x100000</HeapMaxSize> <TCSNum>10</TCSNum> <TCSPolicy>1</TCSPolicy> <!-- Recommend changing 'DisableDebug' to 1 to make the enclave undebuggable for enclave release --> <DisableDebug>0</DisableDebug> <MiscSelect>0</MiscSelect> <MiscMask>0xFFFFFFFF</MiscMask> </EnclaveConfiguration>

文件路径	说明	示例代码

Enclave/Enclave.h	这个头文件内容基本为空。	`#ifndef _ENCLAVE_H_`
		`#define _ENCLAVE_H_`
		`#endif`

Enclave/Enclave.cpp	无。	`#include "Enclave.h"`
		`#include "Enclave_t.h" /* print_string */`
		`#include <string.h>`
		`void ecall_hello_from_enclave(char *buf, size_t len)`
		`{`
		`  const char *hello = "Hello world";`
		`  size_t size = len;`
		`  if(strlen(hello) < len)`
		`  {`
		`  size = strlen(hello) + 1;`
		`  }`
		`  memcpy(buf, hello, size - 1);`
		`  buf[size-1] = '\0';`
		`}`

Enclave/Enclave_private.pem	生成签名密钥。	`openssl genrsa -out Enclave/Enclave_private.pem -3 3072`

文件路径	说明	示例代码

App/App.h	无。	#ifndef _APP_H_
		#define _APP_H_
		#include <assert.h>
		#include <stdio.h>
		#include <stdlib.h>
		#include <stdarg.h>
		#include "sgx_error.h" /* sgx_status_t */
		#include "sgx_eid.h" /* sgx_enclave_id_t */
		#ifndef TRUE
		# define TRUE 1
		#endif
		#ifndef FALSE
		# define FALSE 0
		#endif
		# define TOKEN_FILENAME "enclave.token"
		# define ENCLAVE_FILENAME "enclave.signed.so"
		extern sgx_enclave_id_t global_eid; /* global enclave id */
		#if defined(__cplusplus)
		extern "C" {
		#endif
		#if defined(__cplusplus)
		}
		#endif
		#endif /* !_APP_H_ */

		#include <stdio.h>
		#include <string.h>
		#include <assert.h>
		#include <time.h>
		#include <ctime>
		# include <unistd.h>
		# include <pwd.h>
		# define MAX_PATH FILENAME_MAX
		#include "sgx_urts.h"
		#include "App.h"
		#include "Enclave_u.h"
		/* Global EID shared by multiple threads */
		sgx_enclave_id_t global_eid = 0;
		int initialize_enclave(void)
		{
		  sgx_status_t ret = SGX_ERROR_UNEXPECTED;
		  char enclavefile[256];
		  getcwd(enclavefile, sizeof(enclavefile));
		  strcat(enclavefile, "/enclave.signed.so");

文件路径	说明	示例代码

App/App.cpp	无。	/* Call sgx_create_enclave to initialize an enclave instance */
		/* Debug Support: set 2nd parameter to 1 */
		ret = sgx_create_enclave(enclavefile, SGX_DEBUG_FLAG, NULL, NULL, &global_eid, NULL);
		if (ret != SGX_SUCCESS) {
		printf("Failed to create enclave, ret code: %d, enclave file: %s\n", ret, enclavefile);
		return -1;
		}
		return 0;
		}
		tm* get_time() {
		time_t rawtime;
		struct tm * timeinfo;
		time (&rawtime);
		timeinfo = localtime (&rawtime);
		return timeinfo;
		}
		/* Application entry */
		int SGX_CDECL main(int argc, char *argv[])
		{
		(void)(argc);
		(void)(argv);
		const size_t max_buf_len = 100;
		char buffer[max_buf_len] = {0};
		/* Initialize the enclave */
		if(initialize_enclave() < 0){
		printf("Enter a character before exit ...\n");
		getchar();
		return -1;
		}
		/* Enclave calls */
		while(1) {
		ecall_hello_from_enclave(global_eid, buffer, max_buf_len);
		printf("%s%s\n", asctime(get_time()), buffer);
		fflush(stdout);
		sleep(1);
		}
		/* Destroy the enclave */
		sgx_destroy_enclave(global_eid);
		printf("Info: SampleEnclave successfully returned.\n");
		printf("Enter a character before exit ...\n");
		getchar();
		return 0;
		}

参考链接

- ACK-TEE机密计算介绍
- 创建加密计算托管集群
- Intel® Software Guard Extensions Developer Guide

28.5. 在ACK-TEE集群中使用Inclavare Containers机密容器

Inclavare Containers是工业界首个面向机密计算场景的开源容器运行时，它在基于硬件的可信执行环境中启动受保护的容器，以防止不受信任的实体（例如：云服务商）访问您的敏感数据。在ACK-TEE集群中，您可以基于Inclavare Containers灵活、便捷地部署机密容器。本文主要为您介绍如何在ACK-TEE集群中基于Inclavare Containers部署和使用机密容器（简称Inclavare Containers机密容器）。

前提条件

- 创建加密计算托管集群。

> **说明** 目前只有规格族为安全增强计算型c7t、安全增强通用型g7t、安全增强内存型r7t的实例支持ACK-TEE集群，请确保所选可用区有这些实例。

- 步骤二：选择集群凭证类型。

背景信息

在ACK-TEE集群中虽然也可以使用常规技术部署和使用机密容器，但您需要掌握机密计算领域的专业知识，并按照SGX SDK应用开发规范开发和构建镜像。而Inclavare Containers能帮您省去这些复杂过程，降低机密计算的使用门槛，灵活对接不同类型的Enclave运行时形态，同时保持和普通容器一样的使用体验。更多信息，请参见inclavare-containers。

步骤一：搭建Inclavare Containers机密容器运行环境

> **说明** 由于Pod的调度具有随机性，如果在部署Inclavare Containers机密容器时，您不通过打标签的方式定义被调度Pod所属的节点，则需要在每一个节点搭建Inclavare Containers机密容器的运行环境。

1. 执行以下命令，安装 `rune` 和 `shim-rune` 。更多信息，请参见rune和shim-rune。

   > **说明**
   > - `rune` 是符合OCI Runtime规范的命令行工具，用于在容器里创建和运行Enclave的命令行工具。更多信息，请参见runtime-container。
   > - `shim-rune` 是为容器运行时 `rune` 提供的 `shim` ，在普通 `shim` 功能的基础上额外提供了Enclave签名和管理功能。

   ```
   yum-config-manager --add-repo https://mirrors.openanolis.org/inclavare-containers/alinux2-repo && \
    rpm --import https://mirrors.openanolis.org/inclavare-containers/alinux2-repo/RPM-GPG-KEY-rpm-sign && \
    yum install -y rune shim-rune
   ```

2. 配置容器引擎Containerd。

   > **说明** ACK-TEE机密计算容器引擎Containerd默认的OCI Runtime是runC，而运行机密容器需要用新的容器运行时runE。

i. 执行以下命令，配置OCI Runtime runE运行时。

```
cd /etc/containerd/ && \
 sed -i 's/\(default_runtime_name = \)"runc"/\1"rune"/' config.toml && \
 sed -i '/\[plugins."io.containerd.grpc.v1.cri".containerd.runtimes\]/a\\t[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.rune]' config.toml && \
 sed -i '/\[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.rune\]/a\\t  runtime_type = "io.containerd.rune.v2"' config.toml
```

ii. 执行以下命令，重启Containerd。

```
systemctl restart containerd.service
```

iii. 执行以下命令，查看Containerd是否启动成功。

```
systemctl status containerd.service
```

预期输出：

```
containerd.service - containerd container runtime
  Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; vendor preset: disabled)
  Active: active (running) since 二 2021-06-15 19:16:19 CST; 5s ago
    Docs: https://containerd.io
 Process: 212462 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
Main PID: 212464 (containerd)
   Tasks: 288
  Memory: 6.3G
  ...
```

3. 执行以下命令，安装Occlum软件栈。

> **说明** Occlum是Inclavare Containers当前支持的Enclave运行时，而Inclavare Containers需要Enclave运行时的配合才能运行机密容器。

```
yum install -y occlum-pal occlum-rdfsbase-dkms
```

4. 执行以下命令，查看 `occlum-pal` 是否安装成功。

```
ls /opt/occlum/build/lib/
```

预期输出：

```
libocclum-pal.so.0.21.0
```

5. 执行以下命令，查看 `rdfsbase` 驱动是否安装成功。

```
lsmod | grep enable_rdfsbase
```

预期输出：

```
enable_rdfsbase     16384  0
```

步骤二：构建Inclavare Containers机密容器镜像

> **说明**
> - 步骤1~5：基于Occlum的开发镜像构建并打包Hello World可信应用。
> - 步骤6：构建包含Hello World可信应用的机密容器镜像。

1. 执行以下命令，运行Occlum开发环境 `occlum-app-builder` 。

   > **说明** `docker.io/occlum/occlum` 镜像版本需要与安装的Occlum版本一致。

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Pod
   metadata:
    labels:
     run: occlum-app-builder
    name: occlum-app-builder
    namespace: default
   spec:
    hostNetwork: true
    containers:
    - command:
     - sleep
     - infinity
     image: docker.io/occlum/occlum:0.21.0-centos8.2
     imagePullPolicy: IfNotPresent
     securityContext:
      privileged: true
     name: occlum-app-builder
   EOF
   ```

2. 执行以下命令，登录到Occlum开发环境 `occlum-app-builder` 容器。

   ```
   kubectl exec -it occlum-app-builder -c occlum-app-builder -- /bin/bash
   ```

3. 在Occlum开发环境 `occlum-app-builder` 内安装Docker。

   i. 安装Docker。具体安装操作，请参见Install Docker。

   ii. Docker安装完成后，执行以下命令启动Docker服务。

   ```
   nohup dockerd -b docker0 --storage-driver=vfs &
   ```

   > **说明** 在默认情况下Systemd未安装在容器中，因此您无法通过Systemd管理Docker服务，需要用 `nohup dockerd -b docker0 --storage-driver=vfs &` 来启动Docker服务。

   iii. 执行以下命令，确认Docker服务成功启动。

   ```
   docker ps
   ```

   预期输出：

   ```
   CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS   NAMES
   ```

4. 使用以下C代码样例，编写Hello World代码。

```
cat << EOF > hello_world.c
#include <stdio.h>
#include <unistd.h>
void main(void)
{
  while (1) {
    printf("Hello World!\n");
    fflush(stdout);
    sleep(5);
  }
}
EOF
```

5. 构建并打包Hello World可信应用。
   i. 执行下面命令，使用Occlum工具链编译程序。

   ```
   occlum-gcc -o hello_world hello_world.c
   ```

   ii. 执行下面命令，初始化 `occlum_instance` 。

   ```
   occlum new occlum_instance
   ```

   iii. 执行下面命令，生成Occlum文件系统镜像和Occlum SGX Enclave。

   ```
   cd occlum_instance && \
     cp ../hello_world image/bin/ && \
     openssl genrsa -aes128 -out occlum_key.pem -3 3072 && \
     occlum build --sign-key occlum_key.pem
   ```

   iv. 执行下面命令，打包Hello World可信应用。

   ```
   occlum package occlum_instance.tar.gz
   ```

6. 构建机密容器镜像。
   i. 使用以下Dockerfile模板样例，创建名为*Dockerfile*的文件。

   ```
   cat << EOF >Dockerfile
   FROM scratch
   ADD occlum_instance.tar.gz /
   ENTRYPOINT ["/bin/hello_world"]
   EOF
   ```

   ii. 执行以下命令，构建并上传镜像。

   ```
   docker build -f Dockerfile -t "<$TARGET_IMAGE>" .
   docker login -p <$password> -u <$username>
   docker push "<$TARGET_IMAGE>"
   ```

   > **说明**
   > - 变量*<$password>*、*<$username>*分别表示登录 `docker hub` 的用户名和密码。
   > - 变量*<$TARGET_IMAGE>*表示镜像地址。

步骤三：使用Inclavare Containers机密容器

1. 从Occlum开发环境 `occlum-app-builder` 中退出，在集群中执行以下命令，创建RuntimeClass对象runC和runE，从而您就可以在ACK-TEE集群中部署机密容器。

```
cat << EOF | kubectl apply -f -
apiVersion: node.k8s.io/v1beta1
handler: runc
kind: RuntimeClass
metadata:
 name: runc
---
apiVersion: node.k8s.io/v1beta1
handler: rune
kind: RuntimeClass
metadata:
 name: rune
EOF
```

2. 执行以下命令，部署机密容器。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
 labels:
  run: occlum-helloworld
 name:  occlum-helloworld
 namespace: default
spec:
 restartPolicy: Always
 runtimeClassName: rune
 containers:
  - command:
    - /bin/hello_world
    env:
     - name: ENCLAVE_TYPE
       value: intelSgx
     - name: RUNE_CARRIER
       value: occlum
     - name: ENCLAVE_RUNTIME_LOGLEVEL
       value: info
     - name: ENCLAVE_RUNTIME_PATH
       value: /opt/occlum/build/lib/libocclum-pal.so.0.21.0
     - name: ENCLAVE_RUNTIME_ARGS
       value: occlum_instance
    image: <$TARGET_IMAGE>
    imagePullPolicy: IfNotPresent
    name: hello-world-client
 dnsPolicy: ClusterFirst
EOF
```

上述YAML中涉及的参数含义如下：

参数	描述
ENCLAVE_TYPE	指定要使用的机密计算硬件的类型。
RUNE_CARRIER	`shim-rune` 创建和运行Occlum应用。
ENCLAVE_RUNTIME_LOGLEVEL	指定运行时的日志级别；允许设置的值有：trace、debug、info、warning、error、fatal、panic和off。

参数	描述
ENCLAVE_RUNTIME_PATH	指定启动Enclave运行时的路径。
ENCLAVE_RUNTIME_ARGS	指定启动Enclave运行时的参数。
TARGET_IMAGE	在步骤二第6步中自定义的容器镜像名称。

验证在ACK-TEE集群中使用Inclavare Containers机密容器是否成功运行

执行以下命令，查看容器运行日志。

```
kubectl logs -f occlum-helloworld
```

预期输出：

```
Hello World!
Hello World!
Hello World!
```

如果容器日志每隔5s打印一行 `Hello World!` ，则表示Inclavare Containers环境安装成功且机密容器也被正确运行了。

> **说明** 如果您有关于任何Inclavare Containers机密容器的问题，请在issues中留言 。

28.6. 在ACK-TEE集群中使用Inclavare Containers机密容器实现远程证明

Inclavare Containers实现了一个针对云场景的通用且跨平台的远程证明架构EAA（Enclave Attestation Architecture）。EAA能够向您证明其敏感的工作负载是运行在基于机密技术硬件的可信执行加密环境（Trusted Execution Environment，简称TEE）中的。本文介绍如何在部署了Inclavare Containers的ACK-TEE集群中实现远程证明。

前提条件

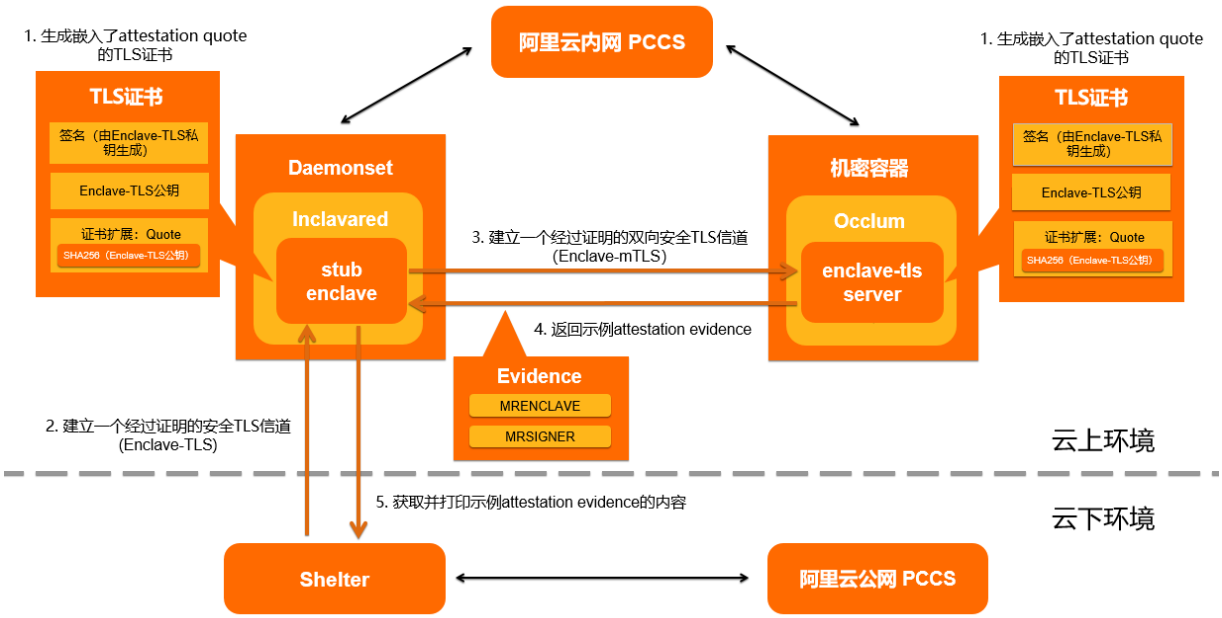
- 创建加密计算托管集群

  > **说明** 目前只有**规格族**为**安全增强计算型c7t**、**安全增强通用型g7t**、**安全增强内存型r7t**的实例支持ACK-TEE集群，请确保所选可用区有这些实例。更多信息，请参见实例规格族。

- 创建一台装有CentOS 8.2或Ubuntu 18.04操作系统的ECS，用于运行远程证明客户端程序Shelter。具体操作，请参见创建ECS实例。
- 通过kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。
- 在ACK-TEE集群上的所有节点上，部署Inclavare Containers机密容器环境。具体操作，请参见在ACK-TEE集群中使用Inclavare Containers机密容器。

背景信息

EAA远程证明以关联了带有硬件可执行环境的Quote的TLS证书为信任根，确保通信双方的通信信道的安全性是完全基于硬件可信执行环境的。具体工作流程和架构图如下：



当您想验证工作负载是否运行在可信平台上时，可以启动Shelter验证工具发送验证请求给Inclavared对平台进行验证。具体工作流程如下：

1. 当Inclavared接收到Shelter的验证请求之后，将请求发送给机密容器，Inclavared和机密容器分别产生带有硬件可执行环境的Quote的TLS证书。
2. Inclavared和Shelter之间基于Enclave-TLS建立经过证明的安全信道。
3. Inclavared与机密容器之间基于Enclave-TLS建立经过双向证明过的安全信道。
4. Inclavared转发机密容器提供的硬件可执行环境信息和敏感信息给Shelter。
5. Shelter对Enclave运行时的度量值进行验证，并返回验证结果。

该架构包含以下组件：

组件名称	角色	功能
机密容器	工作负载	在Occlum中运行服务端程序enclave-tls-server，并基于Enclave-TLS，响应来自Inclavared的请求，并返回机密容器的 attestation evidence （ mrenclave 值和 mrsigner 值）。更多信息，请参见occlum和enclave-tls。
Inclavared	证明者	负责转发下游的机密容器和上游的客户端验证者程序Shelter之间的流量，且受到经过证明的Enclave-TLS信道的保护。
Shelter	线下的远程证明验证者	1. 记录Enclave运行时的启动度量值。 2. 建立可信TLS信道与Inclavared进行通信。 3. Shelter对Enclave运行时的度量值进行验证，以便您能够明确知道自己的工作负载是否在被认证的TEE中加载。
阿里云证书缓存服务PCCS（Provisioning Certificate Caching Service）	远程证明服务	阿里云SGX远程证明服务完全兼容Intel® SGX ECDSA远程证明服务和Intel® SGX SDK，因此阿里云vSGX实例（即g7t、c7t或r7t实例的简称）能够通过远程证明来获得远程提供商或生产者的信任。更多信息，请参见Intel® SGX ECDSA远程证明服务和Intel® SGX SDK。

使用限制

- 仅支持对DaemonSet应用进行远程证明，即必须在该集群的所有节点上有且仅有一个DaemonSet应用。
- 为了能让远程证明客户端程序Shelter直接访问到工作负载，集群中每个节点都要绑定EIP。具体操作，请参见步骤四：绑定节点EIP。

步骤一：部署Inclavared组件

1. 执行以下命令，部署名称为 `inclavared` 的DaemonSet应用。

```
cat <<-EOF | kubectl apply -f -
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: inclavared
spec:
 selector:
  matchLabels:
   k8s-app: inclavared
 template:
  metadata:
   labels:
    k8s-app: inclavared
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: alibabacloud.com/tee-hardware-category
          operator: In
          values:
          - intel-sgx1
          - intel-sgx2
   containers:
   - image: docker.io/inclavarecontainers/inclavared:latest
    imagePullPolicy: IfNotPresent
    securityContext:
     privileged: true
    name: inclavared
    command:
     - /usr/local/bin/inclavared
    args:
     - --listen
     - 0.0.0.0:1236
     - --xfer
     - 127.0.0.1:1234
     - --attester
     - sgx_ecdsa
     - --verifier
     - sgx_ecdsa_qve
     - --mutual
    volumeMounts:
    - mountPath: /dev/sgx/enclave
     name: dev-enclave
    - mountPath: /dev/sgx/provision
     name: dev-provision
    - mountPath: /var/run/aesmd
     name: run-dir
    resources:
```

```
      requests:
        cpu: 100m
        memory: 100Mi
      limits:
        cpu: 1
        memory: 1000Mi
  tolerations:
  - effect: NoSchedule
    key: alibabacloud.com/sgx_epc_MiB
    operator: Exists
  volumes:
  - hostPath:
      path: /var/run/aesmd
      type: DirectoryOrCreate
    name: run-dir
  - hostPath:
      path: /dev/sgx_enclave
    name: dev-enclave
  - hostPath:
      path: /dev/sgx_provision
    name: dev-provision
  hostNetwork: true
EOF
```

2. 执行以下命令，查看Inclavared是否部署成功。

```
kubectl get daemonset inclavared
```

预期输出：

```
NAME     DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
inclavared  2     2    2  2      2     <none>     7d22h
```

> **说明** 当Inclavared启动后，Inclavared在端口监听地址 `0.0.0.0:1236` 监听来自远程认证客户端Shelter的请求，且会把请求转发给当前节点的工作负载。

步骤二：部署工作负载occlum-attestation-app

1. 执行以下命令，部署名称为 `occlum-attestation-app` 的DaemonSet应用。

```
cat <<-EOF | kubectl apply -f -
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: occlum-attestation-app
spec:
 selector:
  matchLabels:
   k8s-app: occlum-attestation-app
 template:
  metadata:
   labels:
    k8s-app: occlum-attestation-app
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
```

```
      nodeSelectorTerms:
      - matchExpressions:
        - key: alibabacloud.com/tee-hardware-category
          operator: In
          values:
          - intel-sgx1
          - intel-sgx2
    containers:
    - image: docker.io/inclavarecontainers/occlum-ecdsa-server:0.21.0
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: true
      name: occlum-attestation-app
      command:
        - /bin/enclave-tls-server
      args:
        - --ip
        - "127.0.0.1"   #本地机的IP地址。
        - --port
        - "1234"
        - --mutual
      env:
        - name: ENCLAVE_TYPE
          value: intelSgx
        - name: RUNE_CARRIER
          value: occlum
        - name: ENCLAVE_RUNTIME_LOGLEVEL
          value: info
        - name: ENCLAVE_RUNTIME_PATH
          value: /opt/occlum/build/lib/libocclum-pal.so.0.21.0
        - name: ENCLAVE_RUNTIME_ARGS
          value: occlum_workspace_server
        - name: OCCLUM_RELEASE_ENCLAVE
          value: "1"
      workingDir: /run/rune
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
        limits:
          cpu: 1
          memory: 1000Mi
    tolerations:
    - effect: NoSchedule
      key: alibabacloud.com/sgx_epc_MiB
      operator: Exists
    hostNetwork: true
EOF
```

2. 执行以下命令，查看 occlum-attestation-app 是否部署成功。

```
kubectl get daemonset occlum-attestation-app
```

预期输出：

```
NAME                     DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
occlum-attestation-app  2        2        2      2           2          <none>         7d22h
```

从上述输出信息可知 `occlum-attestation-app` 已部署成功，工作负载中的 `enclave-tls-server` 程序，在端口监听地址 `127.0.0.1:1234` 监听来自客户端Inclavared的连接请求。

步骤三：修改集群安全组配置

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群信息管理页面，单击**集群资源**页签，然后单击安全组右侧的链接。
5. 修改默认安全组规则，允许入方向 `1236` 端口可访问。具体操作，请参见修改安全组规则。

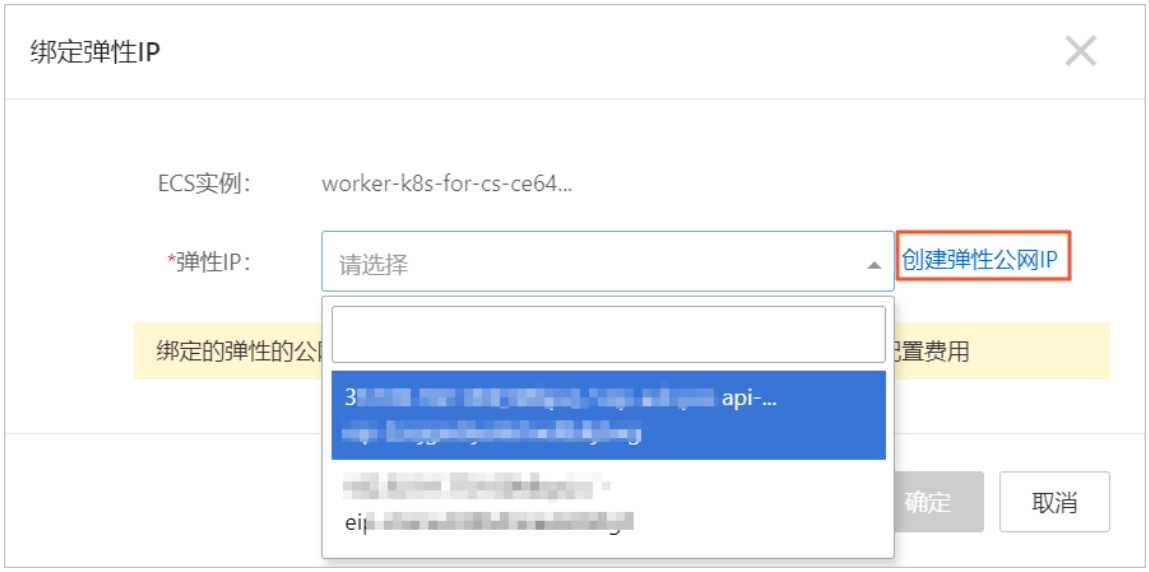
步骤四：绑定节点EIP

> **说明** 如果ECS实例已有公网IP，请跳过该步骤。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理 > 节点**。
5. 在**节点**页面，单击目标节点的**实例ID**名称。
6. 在云服务器ECS控制台，选择**实例与镜像 > 实例**，单击**实例详情**页签。
7. 单击**绑定弹性IP**为实例分配公网IP。



- 如果已有弹性IP，在**绑定弹性IP**对话框中，从**弹性IP**下拉列表中选择目标弹性IP。
- 如果没有弹性IP，在**绑定弹性IP**对话框中，单击**创建弹性公网IP**。具体操作，请参见申请EIP。



8. 单击**确定**。

步骤五：安装远程证明客户端程序Shelter

1. 安装SGX PSW环境。

   > **说明** Shelter在验证嵌有SGX认证信息的TLS证书过程中，需要依赖SGX PSW提供的动态库。更多信息，请参见SGX PSW。

   - 如果您使用的是CentOS 8.2系统，执行以下命令，安装SGX PSW。

     ```
     yum install -y yum-utils && \
       wget -c https://download.01.org/intel-sgx/sgx-linux/2.13/distro/centos8.2-server/sgx_rpm_local_repo.tgz && \
       tar xzf sgx_rpm_local_repo.tgz && \
       yum-config-manager --add-repo sgx_rpm_local_repo && \
       yum makecache && rm -f sgx_rpm_local_repo.tgz && \
       yum install --nogpgcheck -y libsgx-dcap-quote-verify \
       libsgx-dcap-default-qpl libsgx-dcap-ql \
       libsgx-uae-service
     ```

   - 如果您使用的是Ubuntu 18.04系统，执行以下命令，安装SGX PSW。

     ```
     apt-get update -y && apt-get install -y wget gnupg && \
       echo "deb [arch=amd64] https://download.01.org/intel-sgx/sgx_repo/ubuntu bionic main" | tee /etc/apt/sources.list.d/intel-sgx.list && \
       wget -qO - https://download.01.org/intel-sgx/sgx_repo/ubuntu/intel-sgx-deb.key | apt-key add - && \
       apt-get update -y && \
       apt-get install -y libsgx-dcap-quote-verify=1.10.100.4-bionic1 \
       libsgx-dcap-default-qpl=1.10.100.4-bionic1 \
       libsgx-dcap-ql=1.10.103.1-bionic1 \
       libsgx-uae-service=2.13.100.4-bionic1
     ```

2. 配置阿里云PCCS公网URL。

   阿里云SGX远程证明服务采用区域化部署，您可以通过访问实例所在地域的阿里云SGX远程证明服务来获得最佳的稳定性。您需要手动修改*/etc/sgx_default_qcnl.conf*文件以适配实例所在地域的阿里云SGX远程证明服务。

> **说明** 仅中国内地地域支持阿里云SGX远程证明服务。更多信息，请参见地域和可用区。

- 如果vSGX实例已分配公网IP，需要对*/etc/sgx_default_qcnl.conf*的内容做以下修改。

```
# PCCS server address
PCCS_URL=https://sgx-dcap-server.<Region-ID>.aliyuncs.com/sgx/certification/v3/
# To accept insecure HTTPS cert, set this option to FALSE
USE_SECURE_CERT=TRUE
```

- 如果vSGX实例只有VPC内网IP，需要对*/etc/sgx_default_qcnl.conf*的内容做以下修改。

```
# PCCS server address
PCCS_URL=https://sgx-dcap-server-vpc.<Region-ID>.aliyuncs.com/sgx/certification/v3/
# To accept insecure HTTPS cert, set this option to FALSE
USE_SECURE_CERT=TRUE
```

> **说明** 您需要将*<Region-ID>*替换为vSGX实例所在地域的ID。

3. 安装远程证明客户端程序Shelter。
   - 如果您使用的是CentOS 8.2系统，执行以下命令，安装远程证明客户端程序Shelter。

```
yum-config-manager --add-repo https://mirrors.openanolis.org/inclavare-containers/rpm-repo/ && \
  rpm --import https://mirrors.openanolis.org/inclavare-containers/rpm-repo/RPM-GPG-KEY-rpm-sign && \
  yum install -y shelter
```

   - 如果您使用的是Ubuntu 18.04系统，执行以下命令，安装远程证明客户端程序Shelter。

```
echo 'deb [arch=amd64] https://mirrors.openanolis.org/inclavare-containers/deb-repo bionic main' | tee /etc/apt/sources.list.d/inclavare-containers.list && \
  wget -qO - https://mirrors.openanolis.org/inclavare-containers/deb-repo/DEB-GPG-KEY.key | apt-key add - && \
  apt-get update -y && apt-get install -y shelter
```

4. 执行以下命令，查看Shelter是否安装成功。

```
which shelter
```

预期输出：

```
/usr/local/bin/shelter
```

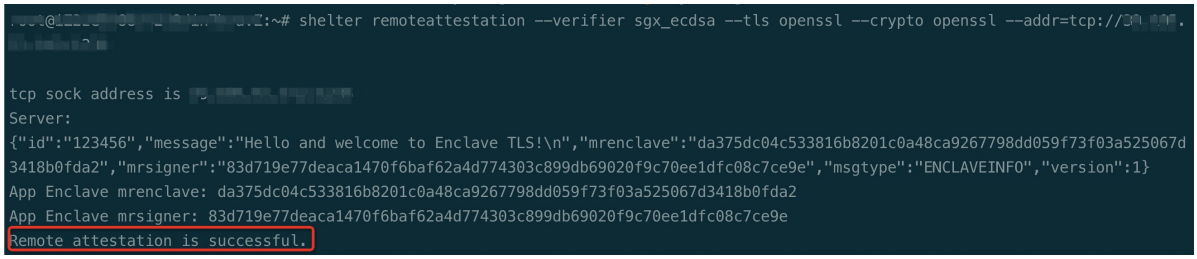
验证在ACK-TEE集群中使用Inclavare Containers机密容器实现远程证明是否成功

对集群中所有节点依次执行以下命令，运行远程证明客户端程序Shelter。

> **说明** 执行命令前请将*<$IP>*替换为节点的EIP。

```
shelter remoteattestation --verifier sgx_ecdsa --tls openssl --crypto openssl --addr=tcp://<$IP>:1236
```

- 如果在SGX环境中运行Shelter，预期输出如下。



如果输出结果中包含 **Remote attestation is successful** ，则表明您已成功实现远程证明。

- 如果在非SGX环境中运行Shelter，除了会输出上述包含 **Remote attestation is successful** 的信息，还会输出以下错误信息。

  ```
  [load_qve ../sgx_dcap_quoteverify.cpp:209] Error, call sgx_create_enclave for QvE fail [load_qve], SGXError:2006.
  [sgx_qv_get_quote_supplemental_data_size ../sgx_dcap_quoteverify.cpp:527] Error, failed to load QvE.
  ```

  上述输出的错误信息您可以忽略，因为其对最终的证明结果没有任何影响。
  出现该错误的原因是：Shelter在调用sgx_qv_get_quote_supplemental_data_size()时，总会先尝试加载QVE（Quote Verification Enclave）。如果加载失败（所有非SGX环境都无法成功加载QVE），Shelter会输出上述错误信息，然后会自动使用QVL（Quote Verification Library）进行后续证明逻辑。在非SGX环境中，验证过程都是由QVL处理。更多信息，请参见Intel® SGX ECDSA远程证明。