ALIBABA CLOUD

Alibaba Cloud

容器服务Kubernetes版

Serverless Kubernetes集群用户指南

文档版本：20211201

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.ASK概述

本文介绍阿里云Serverless Kubernetes（ASK）集群的产品简介、核心优势与阿里云容器服务Kubernetes版（ACK）集群对比及应用场景，帮助您快速地了解ASK集群。

产品简介

ASK集群是阿里云推出的无服务器Kubernetes容器服务。您无需购买节点即可直接部署容器应用，无需对集群进行节点维护和容量规划，并且根据应用配置的CPU和内存资源量进行按需付费。ASK集群提供完善的Kubernetes兼容能力，同时降低了Kubernetes使用门槛，让您更专注于应用程序，而不是管理底层基础设施。

ASK集群中的Pod基于阿里云弹性容器实例ECI运行在安全隔离的容器运行环境中。每个Pod容器实例底层通过轻量级虚拟化安全沙箱技术完全强隔离，容器实例间互不影响。

ASK集群包括ASK标准版集群和ASK Pro版集群。ASK Pro版集群是在ASK标准版基础上，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SLA的ASK集群。关于ASK Pro版集群的更多信息，请参见ASK Pro版集群概述。

核心优势

- **免运维**：低门槛快速创建Serverless集群，秒级部署容器应用。无需管理Kubernetes节点和服务器，聚焦业务应用。
- **秒级弹性**：无需担心集群节点的容量规划。根据应用负载，轻松灵活扩容应用所需资源。
- **原生兼容**：支持原生Kubernetes应用和生态，包括Service、Ingress、Helm等，无缝迁移Kubernetes应用。
- **安全隔离**：Pod基于ECI创建，应用Pod之间相互隔离防止互相干扰。
- **降低成本**：应用按需创建，按量计费，不运行不计费，没有资源闲置费用，同时Serverless带来更低的运维成本。
- **服务集成**：支持容器应用与阿里云基础服务无缝整合；支持与VPC中现有应用、数据库直接交互；支持容器与虚拟机应用的互联互通。

产品定价

集群中没有节点费用，Pod基于ECI按量计费，请参见ECI计费说明。

关于集群中SLB、PrivateZone等资源价格，请参见以下文档：

- ALB计费项概述
- 云解析PrivateZone产品计费

ASK与ACK集群的对比

应用场景

- **应用托管**

  ASK集群中无需管理和维护节点，无需容量规划，极大降低业务的基础设施管理和运维成本。

- **在线业务弹性**

  对于有着明显的波峰波谷特征的业务负载，例如在线教育、电子商务等行业，ASK集群的秒级伸缩能力可以显著降低计算成本，减少闲置资源浪费，平滑应对突发流量高峰。

- **数据计算**

  面对Spark等数据计算需求，ASK集群可以在短时间内启动大量Pod及时处理任务，计算结束时Pod自动释放停止计费，极大降低整体计算成本。更多信息，请参见通过ASK创建Spark计算任务。

- **CI/CD持续集成**

  基于ASK集群搭建Jenkins或Gitlab-Runner等持续集成环境，并快速完成应用源码编译、镜像构建和推送以及应用部署的流水线，各持续集成任务之间安全隔离互不影响，同时无需维护固定资源池，降低计算成本。更多信息，请参见ASK弹性低成本CI/CD。

- **定时任务**

  在ASK集群中运行定时任务，任务结束停止计费。无需维护固定资源池，避免资源闲置浪费。

2.ASK计费说明

阿里云Serverless Kubernetes（ASK）集群分为ASK标准版和ASK Pro版，不同类型集群的计费项和计费标准不同。本文分别介绍ASK标准版和ASK Pro版的计费说明。

ASK标准版计费说明

- 集群中没有节点费用，Pod基于ECI按量计费。更多信息，请参见ECI计费说明。
- 关于集群中SLB、PrivateZone等资源价格，请参见以下文档：
  - ALB计费项概述
  - 云解析PrivateZone产品计费

ASK Pro版计费说明

- ASK Pro版集群相较于ASK标准版集群会额外收取集群管理费，如下表所示。

计费方式	价格
按量计费	每个集群0.09美元/小时

  具体规则如下：
  - **计费出账**

    ASK Pro版集群费用的计费周期为1小时，即阿里云将在下一个小时就您上一个小时的服务使用进行计量、出具账单，出具账单后从您的阿里云账户中按账单金额扣划服务费用。账单出账时间通常在当前计费周期结束后10至30分钟内。账单开出后，如果您账户余额充足的话，系统将从您的账号中自动扣除账单中的费用数额。

    如果集群创建不足半小时，第一个整点不作为出账周期且不计费。例如：
    - 您在10:15:00创建了一台ASK Pro版集群，那么11:00:00为一个出账周期，账单出具时间在11:10:00~11:30:00之间。
    - 您在10:50:00创建了一台ASK Pro版集群，那么11:00:00将不会作为一个出账周期，12:00:00会作为一个出账周期。账单出具时间在12:10:00~12:30:00之间。
  - **欠费释放**

    如果您的账户余额不足以支付账单金额，您的ASK Pro版集群会处于欠费状态，您将无法访问集群API Server，但容器实例仍可继续运行。如果超过15天仍处于欠费状态，阿里云将暂停为您提供服务，删除您的ASK Pro版集群及集群中的容器实例。因集群删除而释放的容器实例，将不可被恢复。
- 集群中没有节点费用，Pod基于ECI按量计费。更多信息，请参见ECI计费说明。
- 关于集群中SLB、PrivateZone等资源价格，请参见以下文档：
  - ALB计费项概述
  - 云解析PrivateZone产品计费

相关文档

- ASK概述
- ASK Pro版集群概述

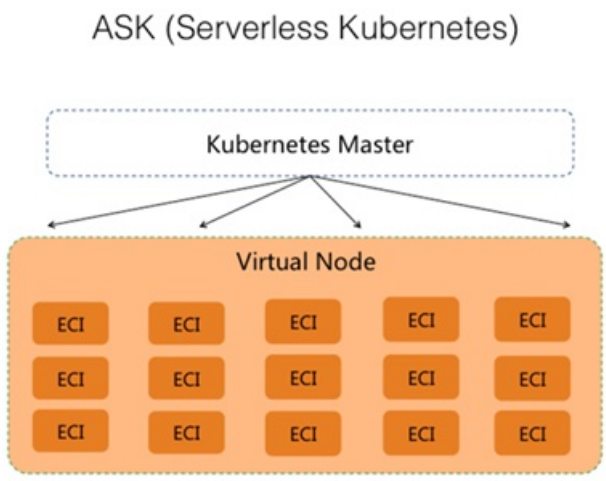
- 产品计费

3.功能简介

本文为您介绍阿里云Serverless Kubernetes（ASK）集群支持的功能。您可以通过了解这些功能来更有效地使用ASK集群。

虚拟节点

ASK集群中基于虚拟节点创建Pod。虚拟节点实现了Kubernetes与弹性容器实例ECI的无缝连接，让Kubernetes集群获得极大的弹性能力，而不必关心底层计算资源容量。



在集群中用户可以查看虚拟节点信息，但无需对虚拟节点进行任何操作，虚拟节点也不占用任何计算资源。有关虚拟节点详情，请参见通过部署ACK虚拟节点组件创建ECI Pod。

Pod安全隔离

ASK集群中的Pod基于阿里云弹性容器实例ECI运行在安全隔离的容器运行环境中。每个容器实例底层通过轻量级虚拟化安全沙箱技术完全强隔离，容器实例间互不影响。

ECI的容器实例底层运行在Alibaba Cloud Linux2操作系统之上，ASK集群作为Serverless容器服务，您无法访问ECI底层OS运行环境。

Pod配置

Pod基于弹性容器实例ECI创建，其支持原生的Kubernetes Pod功能，包括启动多个容器、设置环境变量、设置RestartPolicy、设置健康检查命令和挂载volumes、preStop等。同时支持执行命令 `kubectl logs` 访问容器日志和执行 `kubectl exec` 进入容器。

ASK使用了很多Annotation对Pod进行功能扩展，请参见ECI实例概述。

应用负载管理

- 支持Deployment、StatefulSet、Job/CronJob、Pod、CRD等原生Kubernetes负载类型。
- 不支持DaemonSet：Serverless集群中不支持节点相关的功能。

弹性伸缩

ASK集群中没有真实节点，所以无需考虑节点的容量规划，也无需考虑基于cluster-autoscaler的节点扩容，您只需要关注应用的按需扩容。建议您配置HPA或者CronHPA策略进行Pod的灵活按需扩容。

网络管理

集群中的ECI Pod默认使用Host网络模式，占用交换机VSwitch的一个弹性网卡ENI资源，与VPC内的ECS、RDS互联互通。

- Service
  - 支持创建LoadBalancer类型Service。
  - 不支持NodePort类型Service：Serverless集群中不支持节点相关的功能。
- Ingress
  - SLB Ingress：无需部署Controller直接使用基于SLB七层转发提供的Ingress能力，请参见ingress示例。
  - Nginx Ingress：部署Nginx Ingress Controller后可以创建Nginx Ingress，请参见ingress-nginx示例。
- **服务发现**

  如果您的集群内部应用需要Service的服务发现功能，请在创建集群时开启Privatezone。
- **弹性公网IP**

  支持给ECI Pod挂载EIP，可自动创建或者绑定到已有的EIP实例。

存储管理

Pod支持挂载阿里云块存储和文件存储。

- 阿里云块存储（Disk）
  - 使用flexvolume方式挂载：无需安装flexvolume插件。您可以选择指定diskId挂载，请参见disk-flexvolume-static.yaml示例；或者您也可以动态创建云盘，请参见disk-flexvolume-dynamic.yaml示例。
  - 使用PV/PVC动态创建云盘后挂载：安装disk-controller后即可动态创建云盘后挂载，请参见disk-pvc-dynamic.yaml示例。
- 阿里云文件存储（NAS）
  - 使用NFS volume：支持使用nfs方式挂载NAS目录，请参见nas-nfsvolume.yaml示例。
  - 使用flexvolume静态挂载：无需安装flexvolume插件，直接指定NAS挂载地址，请参见nas-flexvolume.yaml示例。
  - 使用PV/PVC静态挂载：安装disk-controller后即可使用PVC静态挂载NAS目录挂载，请参见nas-pvc.yaml示例。

日志管理

在ASK集群中无需部署logtail daemonset即可收集Pod的stdout和文件输出日志，请参见通过阿里云日志服务采集日志。

配置项及密钥管理

支持Secret和Configmap，以及通过volume挂载Secret和Configmap。

应用目录Chart管理

支持在应用目录部署Chart，连接Kubernetes生态应用。

4.快速入门

4.1. ASK使用快速入门

本文介绍如何在容器服务管理控制台快速创建阿里云Serverless Kubernetes（ASK）集群、使用镜像创建应用和查看容器。

前提条件

- 开通容器服务ACK，且授权默认角色和开通相关云产品。具体操作，请参见首次使用容器服务Kubernetes版。
- 登录弹性容器实例控制台，开通ECI服务。

步骤一：创建ASK集群

> **说明** 本文仅介绍创建ASK集群涉及的关键参数的设置和相关内容，在实际工作场景中，请您根据具体需求设置参数和安装相应组件。更多信息，请参见创建Serverless 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**创建集群**。
4. 单击**ASK集群**页签，然后完成集群配置。

配置项	描述
集群名称	填写集群的名称，本文集群名称为ask-hangzhou。 > **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	选择集群规格，支持**标准版**和**Pro版**。 本文选择标准版。
地域	选择集群所在的地域，本文选择为**华东1（杭州）**。
专有网络	设置集群的网络。Kubernetes集群仅支持专有网络。支持**自动创建**和**使用已有**的VPC。 ○ **自动创建**：集群会自动新建一个VPC，并在VPC中自动创建NAT网关以及配置SNAT规则。 ○ **使用已有**：您可以在已有VPC列表中选择所需的VPC和交换机。如需访问公网，例如下载容器镜像，要配置NAT网关，建议将容器镜像上传到集群所在区域的阿里云镜像服务，并通过内网VPC地址拉取镜像。 详情请参见创建和管理专有网络。 本文选择使用已有专有网络。

配置项	描述
配置SNAT	设置是否为专有网络创建NAT网关并配置SNAT规则。 仅当**专有网络**选择为**自动创建**时，需要设置该选项。 **说明** 若您选择**自动创建**VPC，可选择是否自动配置SNAT。若选择不自动配置SNAT，您可自行配置NAT网关实现VPC安全访问公网环境，并且手动配置SNAT转发策略，否则VPC内实例将不能正常访问公网。 详情请参见创建公网NAT网关实例。 本文选中**为专有网络配置SNAT**。
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。 本文设置为**172.21.0.0/20**
API Server访问	ASK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，ASK会创建一个EIP，并挂载到SLB上。此时，Kubernetes API服务（即API Server）会通过EIP的6443端口暴露出来，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。 更多信息，请参见控制集群API Server的公网访问能力。 本文选中**使用EIP暴露API Server**。
服务协议	创建集群前，需阅读并选中**《无服务器Kubernetes服务协议》**。

5. 在**ASK集群**页面右侧，单击**创建集群**，在弹出的**当前配置确认**对话框中，单击**确定**。

后续步骤

- 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ASK集群。

- 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。

步骤二：使用镜像创建应用

> **说明** 本文仅介绍使用镜像创建应用涉及的关键参数设置，更多信息，请参见使用镜像创建应用。

第一步：配置应用基本信息

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. 在**应用基本信息**配置向导页面，设置应用的基本信息。

配置项	描述
应用名称	设置应用的名称，本文为serverless-app-deployment。
类型	定义资源对象的类型，可选择**无状态（Deployment）**、**有状态（StatefulSet）**等。本文选择**无状态（Deployment）**。

4. 单击**下一步**。

第二步：配置容器

1. 在**容器配置**配置向导页面的**基本配置**区域，完成容器的基本配置。

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**。 您还可以填写私有镜像。填写的格式为 `domainname/namespace/image name:tag` 。 本文选择使用个人版实例镜像，所属地域为**华东1（杭州）**。
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。 本文默认不指定镜像Tag。

2. 在**端口设置**区域，单击**新增**设置容器的端口。
   - 名称：设置容器端口名称。本文容器端口名称为nginx。

- 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。本文容器端口设置为80。
- 协议：支持TCP和UDP。

3. 单击**下一步**。

第三步：完成高级配置

1. 在**高级配置**向导页面的**访问设置**区域，设置暴露后端Pod的方式。

   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，本文为**serverless-app-svc**。
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。 本文服务端口和容器端口分别配置为8080和80。

   在**访问设置**区域，您可以看到创建完毕的服务，您可单击**变更**和**删除**进行二次配置。

2. 单击**创建**。

第四步：访问nginx应用

1. 在**创建完成**页签中单击**查看应用详情**。您可以看到新建的**serverless-app-deployment**出现在无状态列表下。

2. 访问nginx应用。

   i. 在集群管理页左侧导航栏中，选择**网络 > 服务**，可以看到新建的服务**serverless-app-svc**出现在服务列表下。

   ii. 单击新建服务的**外部端点**链接，即可在浏览器中访问外部端点，本示例为Nginx。

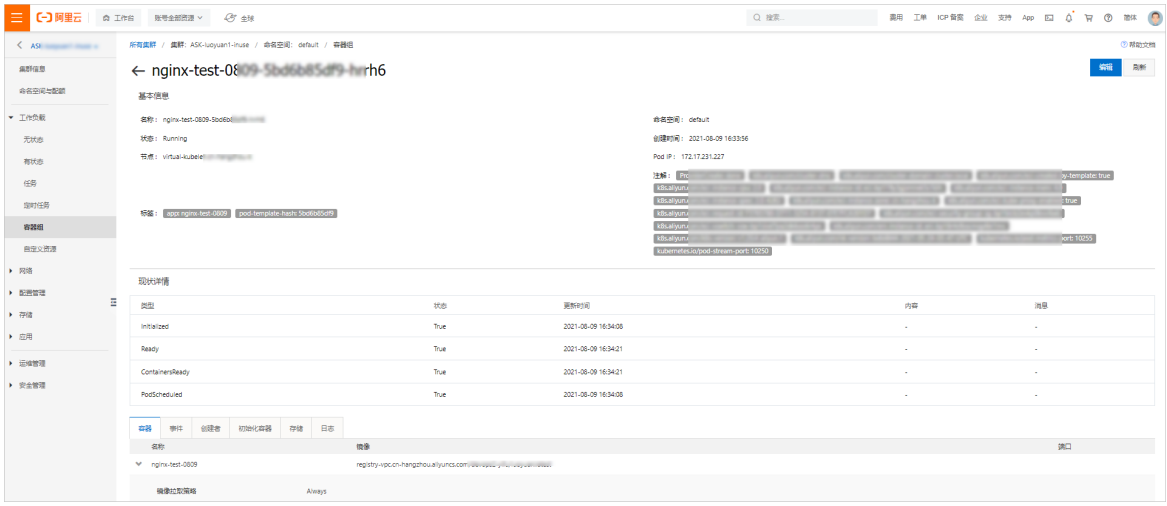
> **说明** 关于如何通过创建Ingress路由访问应用，请参见通过ALB Ingress访问服务。

步骤三：查看容器

1. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。
2. 选择目标容器组，单击右侧**操作**列下的**详情**。

> **说明** 您可对容器组进行更新和删除操作。对于通过模板部署创建的容器组，建议您通过模板进行管理，不要直接删除或修改Pod。

3. 进入容器组的详情页，您可查看该容器组的详情信息。



4.2. 通过ASK一键创建Nginx在线应用

ASK集群无需管理节点，无需进行节点的安全维护等运维操作，满足您对应用托管的免运维诉求，让您关注在应用而非底层基础设施管理。本文介绍如何通过ASK部署在线Web应用，实现应用免运维托管。

前提条件

创建Serverless Kubernetes集群

背景信息

ASK支持标准Kubernetes的语义和API，您可以一键创建Deployment、StatefulSet、Jobs、Service、Ingress或CRD等资源，也可以使用Helm部署各种Kubernetes生态应用。

操作步骤

1. 通过kubectl工具连接集群。
2. 使用以下样例创建名为*nginx.yaml*的YAML文件。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 ports:
 - port: 80
  protocol: TCP
 selector:
  app: nginx
 type: LoadBalancer
---
apiVersion: apps/v1 #对于不同的K8s版本这里需要使用不同的对应版本。
kind: Deployment
metadata:
 name: nginx-deploy
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image:  nginx:alpine
    ports:
    - containerPort: 80
    resources:
     requests:
      cpu: "2"
      memory: "4Gi"
```

> **说明** K8s版本和Deployment的apiVersion对应关系如下：
>
> - K8s 1.6版本之前：`extensions/v1beta1`
> - K8s 1.6版本到1.9版本之间：`apps/v1beta1`
> - K8s 1.9版本之后：`apps/v1`

3. 执行以下命令，部署Nginx示例应用。

```
kubectl apply -f nginx.yaml
```

预期输出：

```
service/nginx-service created
deployment.apps/nginx-deploy created
```

4. 查看Pod和Serivce状态，并通过SLB IP访问Nginx应用。
   i. 执行以下命令，查看Pod状态。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   nginx-deploy-55d8dcf755-bxk8n  1/1   Running  0     36s
   nginx-deploy-55d8dcf755-rchhq  1/1   Running  0     36s
   ```

   ii. 执行以下命令，查看Serivce状态。

   ```
   kubectl get svc
   ```

   预期输出：

   ```
   NAME         TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)     AGE
   kubernetes   ClusterIP   172.**.*.*    <none>       443/TCP     10d
   nginx-service  LoadBalancer  172.19.*.***  47.57.**.**  80:32278/TCP  39s
   ```

   iii. 执行以下命令，通过SLB IP访问Nginx应用。

   ```
   curl 47.57.**.**
   ```

   预期输出：

   ```
   <!DOCTYPE html>
   <html>
   <head>
   <title>Welcome to nginx!</title>
   ...
   </html>
   ```

5. 扩容Deployment。
   i. 执行以下命令，查看Deployment。

   ```
   kubectl get deploy
   ```

   预期输出：

   ```
   NAME       READY  UP-TO-DATE  AVAILABLE  AGE
   nginx-deploy  2/2   2       2       9m32s
   ```

   ii. 执行以下命令，扩容Deployment。

   ```
   kubectl scale deploy nginx-deploy --replicas=10
   ```

   预期输出：

   ```
   deployment.extensions/nginx-deploy scaled
   ```

iii. 执行以下命令，查看扩容后的Pod。

```
kubectl get pod
```

预期输出：

```
NAME              READY  STATUS   RESTARTS  AGE
nginx-deploy-55d8dcf755-8jlz2  1/1   Running  0     39s
nginx-deploy-55d8dcf755-9jbzk  1/1   Running  0     39s
nginx-deploy-55d8dcf755-bqhcz  1/1   Running  0     38s
nginx-deploy-55d8dcf755-bxk8n  1/1   Running  0     10m
nginx-deploy-55d8dcf755-cn6x9  1/1   Running  0     38s
nginx-deploy-55d8dcf755-jsqjn  1/1   Running  0     38s
nginx-deploy-55d8dcf755-lhp8l  1/1   Running  0     38s
nginx-deploy-55d8dcf755-r2clb  1/1   Running  0     38s
nginx-deploy-55d8dcf755-rchhq  1/1   Running  0     10m
nginx-deploy-55d8dcf755-xspnt  1/1   Running  0     38s
```

6. 配置HPA，可以让应用随着负载压力自动进行弹性扩容。当应用的CPU负载增高时，将水平扩容出更多的Pod副本。

i. 执行以下命令，将应用副本缩容到1个。

```
kubectl scale deploy nginx-deploy --replicas=1
```

预期输出：

```
deployment.extensions/nginx-deploy scaled
```

ii. 执行以下命令，查看Pod。

```
kubectl get pod
```

预期输出：

```
NAME              READY  STATUS   RESTARTS  AGE
nginx-deploy-55d8dcf755-rchhq  1/1   Running  0     16m
```

iii. 执行以下命令，配置HPA。

```
kubectl autoscale deployment nginx-deploy --cpu-percent=50 --min=1 --max=10
```

预期输出：

```
horizontalpodautoscaler.autoscaling/nginx-deploy autoscaled
```

iv. 执行以下命令，查看HPA。

```
kubectl get hpa
```

预期输出：

```
NAME      REFERENCE         TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-deploy  Deployment/nginx-deploy  0%/30%  1     10     1     35s
```

5.ECI Pod

5.1. ECI实例概述

本文主要介绍弹性容器实例ECI Pod的配置组成，包括安全隔离、CPU/Memory资源和规格配置、镜像拉取、存储、网络、日志收集等，及ECI的使用限制。

前提条件

- 您需要创建在阿里云容器服务Kubernetes版（ACK）集群中部署虚拟节点，或者创建ASK（Serverless Kubernetes）集群。详情参见步骤一：在ACK集群中部署ack-virtual-node组件和创建Serverless Kubernetes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。

安全隔离

弹性容器实例ECI作为安全可靠的Serverless容器运行环境，每个ECI实例底层通过轻量级安全沙箱技术完全强隔离，实例间互不影响。同时实例在调度时尽可能分布在不同的物理机上，进一步保障了高可用性。

CPU/Memory资源和规格配置

ECI支持多种规格配置的方式来申请资源和计费。

- **指定CPU和Memory**
  - **指定容器的CPU和Memory**：用户可以通过Kubernetes标准方式配置单个容器的CPU和Memory（`resources.limit`），如不指定则默认单个容器的资源是1C2G。ECI的资源则是Pod内所有容器所需资源的总和。（注意我们会对不支持的规格进行自动规整，例如所有容器相加的资源为2C3G，那么将会自动规整为2C4G。超过4C将不会自动规整。）
  - **指定Pod的CPU和Memory**：用户可以通过Annotation的方式配置整个Pod的CPU和Memory，这种情况将不再把所有容器资源相加，而以指定的资源创建和计费。此时可以不用指定内部容器的`request`和`limit`，各容器可以最大程度的使用Pod内计算资源。

  目前ECI Pod支持的CPU和Memory规格有：

vCPU	Memory
.25 vCPU	0.5 GB, 1 GB
.5 vCPU	1 GB, 2 GB
1 vCPU	2 GB, 4 GB, 8 GB
2 vCPU	2 GB, 4 GB, 8 GB, 16 GB
4 vCPU	4 GB, 8 GB, 16 GB, 32 GB
8 vCPU	8 GB, 16 GB, 32 GB, 64 GB
12 vCPU	12 GB, 24 GB, 48 GB, 96 GB
16 vCPU	16 GB, 32 GB, 64 GB, 128 GB

vCPU	Memory
24 vCPU	48 GB, 96 GB, 192 GB
32 vCPU	64 GB, 128 GB, 256 GB
52 vCPU	96 GB, 192 GB, 384 GB
64 vCPU	128 GB, 256 GB, 512 GB

CPU和Memory配置其计费时长从挂载外部存储/下载容器镜像开始至ECI实例停止运行（进入Succeeded/Failed状态）结束。

计费公式：ECI Pod实例费用=（ECI实例CPU核数xCPU单价+ECI实例内存大小x内存单价）xECI实例运行时长，实例按秒计费。当前计费单价如下：

计费项	价格	小时价
CPU（vCPU * 秒）	0.000049 元	0.1764 元/小时
内存（GB * 秒）	0.00000613 元	0.0221 元/小时

- **指定Pod的ECS规格**

  您可以根据需要，指定ECI实例底层使用的ECS规格族，获得各规格族的指定能力，例如指定使用ecs.sn1ne规格族来使用网络增强能力。更多相关信息，请参见ECS实例规格族和各地域ECS按量。

  > **说明** 指定ECS规格创建ECI实例时，计算资源的费用按ECS规格进行计算。目前支持的ECS实例规格族如下：
  >
  > - 通用型：g6e、g6、g5、sn2ne
  > - 计算型：c6e、c6a、c6、c5、sn1ne
  > - 内存型：r6e、r6、r5、se1ne、se1
  > - 密集计算型：ic5
  > - 高主频计算型：hfc6、hfc5
  > - 高主频通用型：hfg6、hfg5
  > - GPU计算型：gn6i、gn6v、gn5i、gn5
  > - 大数据网络增强型：d1ne
  > - 本地SSD型：i2、i2g
  > - 突发性能型：t6、t5
  > - 共享型：s6、xn4、n4、mn4、e4

  计费公式：ECI实例费用=ECI实例指定的ECS规格单价 x ECI实例运行时长，实例按秒计费。

  当指定Pod的ECS规格时，ECI可以选择使用预留实例券对ECI按规格创建实例进行抵扣。详情请参见预留实例券概述。

  ECI使用预留实例券后的费用，跟您使用ECS包月实例的费用相近。

- **抢占式实例**

  您可以通过配置Annotation使用spot可抢占式实例，大幅降低计算成本。

镜像拉取

ECI Pod默认每次启动后使用内部的containerd从远端拉取容器镜像。如果镜像为公共镜像，则需要开通VPC的NAT网关，或者给ECI Pod挂载EIP。建议您将容器镜像存储在阿里云镜像仓库（ACR），通过VPC网络减小镜像拉取时间。另外对于ACR上的私有镜像，实现了免密拉取功能方便您使用。

同时支持了镜像快照功能。通过把容器镜像缓存到快照中，然后通过快照快速启动容器，避免再次从远端拉取镜像，适合大镜像的场景。

存储

支持多种使用存储的方式：

- Flexvolume：
  - 挂载NAS Volume：同标准的Flexvolume用法相同，指定 `nas volume Id` 进行挂载。
  - 挂载Disk Volume：同标准的Flexvolume用法相同，指定 `disk volume Id` 进行挂载。
  - 随ECI实例自动创建 `disk volume` ：为了提供更灵活的云盘挂载能力，ECI支持创建时挂载Flexvolume动态创建一个云盘，可以指定 `disk volume` 的Size大小，也可以配置当ECI实例结束时是否保留 `disk volume` 。
- NFS：参考示例。
- PV/PVC：参考示例。

网络

ECI Pod默认使用Host网络模式，占用交换机VSwitch的一个弹性网卡ENI资源。

在Kubernetes集群环境中，ECI Pod与ECS节点上的Pod互联互通，方法如下：

- LoadBalancer Service挂载ECI Pod：也可以支持service同时挂载ECS节点上的Pod和ECI Pod。
- 访问ClusterIP Service：ECI Pod可以访问集群中的clusterIP地址。
- 挂载EIP：支持给ECI Pod挂载EIP，可自动创建或者绑定到已有的EIP实例。

日志收集

用户可以直接配置Pod的Env收集 `stdout` 或者文件日志到阿里云日志服务SLS中。一般情况无需再部署一个 `logtail sidecar` 容器。

支持Annotation列表

> **注意** Annotation需要配置在Pod Spec中，而不是Deployment Spec中。

Annotation	解释	示例
k8s.aliyun.com/eci-use-specs	表示允许的实例规格，可以配置多个。当前规格没有库存时依次尝试下一个规格创建。支持CPU-MEM格式（${cpu}-${mem}Gi）、ECS规格格式。	"k8s.aliyun.com/eci-use-specs": "2-4Gi,4-8Gi,ecs.c6.xlarge"
k8s.aliyun.com/eci-vswitch	设置Pod的虚拟交换机。	"k8s.aliyun.com/eci-vswitch" : "${your_vsw_id}"

Annotation	解释	示例
k8s.aliyun.com/eci-security-group	设置Pod的安全组。	"k8s.aliyun.com/eci-security-group"："${your_security_group_id}"
k8s.aliyun.com/eci-resource-group-id	设置Pod所在的资源组	"k8s.aliyun.com/eci-resource-group-id"："${your_resource_group_id}"
k8s.aliyun.com/eci-ram-role-name	设置Pod的RamRole，赋予在ECI实例内部可以访问阿里云产品能力。	"k8s.aliyun.com/eci-ram-role-name"："${your_ram_role_name}"
k8s.aliyun.com/eci-image-snapshot-id	指定已有ImageCacheID，加速ECI Pod创建。	k8s.aliyun.com/eci-image-snapshot-id: "${your_image_cache_id}"
k8s.aliyun.com/eci-image-cache	根据用户已有的镜像缓存，自动匹配镜像缓存。默认为false。	k8s.aliyun.com/eci-image-cache: "true"
k8s.aliyun.com/eci-with-eip	创建弹性公网IP，绑定到ECI Pod。	"k8s.aliyun.com/eci-with-eip": "true"
k8s.aliyun.com/eip-bandwidth	设置弹性公网IP带宽，如果不指定默认为5M。	"k8s.aliyun.com/eci-with-eip": "true""k8s.aliyun.com/eip-bandwidth": 10
k8s.aliyun.com/eci-eip-instanceid	给Pod绑定已有的弹性公网IP。	"k8s.aliyun.com/eci-eip-instanceid": "${your_eip_Instance_Id}"
k8s.aliyun.com/eci-spot-strategy	SpotAsPriceGo：系统自动出价，跟随当前市场实际价格。 SpotWithPriceLimit：设置抢占实例价格上限。	k8s.aliyun.com/eci-spot-strategy: "SpotAsPriceGo"
k8s.aliyun.com/eci-spot-price-limit	只有k8s.aliyun.com/eci-spot-strategy设置为SpotWithPriceLimit时有效。设置实例每小时最高价格，支持最多3位小数。	k8s.aliyun.com/eci-spot-price-limit: "0.250"
k8s.aliyun.com/eci-ntp-server	设置ntp server，支持设置多个。	k8s.aliyun.com/eci-ntp-server: 100.100.5.1,100.100.5.2 # 设置您的NTP服务器地址。
k8s.aliyun.com/eci-set-diskvolume	把volume（emptyDir或者hostPath）转换为动态创建云盘。格式为”$volumeName:$type:$size“。	k8s.aliyun.com/eci-set-diskvolume: "cache-volume:ext4:500Gi"

ECI限制

ECI和虚拟节点目前已经兼容了大部分Pod功能，以下功能暂不支持。

- 不支持在虚拟节点上运行DaemonSet Pod。
- 不支持hostPath/hostPid。
- 不支持privileged权限开放。
- 不支持NodePort类型的Service。
- 不支持Network Policy。

5.2. 通过指定CPU和内存创建ECI Pod

在创建ECI Pod时，您可以指定单个容器或ECI Pod的CPU与内存规格。本文分别介绍如何指定ECI实例内容器规格和ECI Pod规格。

指定ECI实例内容器规格

通过Kubernetes标准方式配置单个容器的CPU和内存，ECI的资源则是Pod内所有容器所需资源的总和。

每个ECI实例支持最多20个容器，实例内每个容器的资源支持自定义配置，但汇总到ECI实例级别需要满足CPU和内存约束。

- 对于未满足的情况，ECI会执行自动规整操作，计费按照规整后CPU和内存值进行计费。
- 对于大于4 vCPU的情况，为了减少因操作失误造成规整，浪费计费资源，您需要严格声明规格资源，否则接口会返回规格非法错误。

ECI Pod支持的CPU和内存规格，如下表所示。

vCPU	内存
0.25 vCPU	0.5 GB, 1 GB
0.5 vCPU	1 GB, 2 GB
1 vCPU	2 GB, 4 GB, 8 GB
2 vCPU	2 GB, 4 GB, 8 GB, 16 GB
4 vCPU	4 GB, 8 GB, 16 GB, 32 GB
8 vCPU	8 GB, 16 GB, 32 GB, 64 GB
12 vCPU	12 GB, 24 GB, 48 GB, 96 GB
16 vCPU	16 GB, 32 GB, 64 GB, 128 GB
24 vCPU	48 GB, 96 GB, 192 GB
32 vCPU	64 GB, 128 GB, 256 GB
52 vCPU	96 GB, 192 GB, 384 GB
64 vCPU	128 GB, 256 GB, 512 GB

使用示例

基于Kubernetes原生方式，直接定义对应 `container` 的 `request` 即可。

```
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
    ports:
    - containerPort: 80
    resources:
     requests:
      cpu: "500m"
      memory: "1024Mi"
   - name: busybox
    image: busybox:latest
    ports:
    - containerPort: 80
    resources:
     requests:
      cpu: "500m"
      memory: "1024Mi"
```

指定ECI Pod规格

在指定ECI Pod级别的CPU和内存模式下，对于指定的CPU和内存规格，ECI会尝试使用多种ECS规格进行支撑，以提供比ECS单规格更好的库存和弹性能力。这具有以下优势：

- ECI实例内容器可以不用限制资源上限。在定义实例内容器资源时，可以不用指定 `request` 和 `limit` ，各容器可以最大程度的共享申请的资源。
- 在基因计算和Istio场景下，业务框架会自动给Pod添加Sidecar容器。通过显式指定ECI实例规格，ECI可以无缝的对接这类业务框架。

使用示例

通过在Pod定义中设置annotations: k8s.aliyun.com/eci-use-specs，可以配置多个规格，以逗号分割。

```
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:
    k8s.aliyun.com/eci-use-specs : "2-4Gi"
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
   - name: busybox
    image: busybox:latest
    ports:
    - containerPort: 80
    resources:
     limits:
      cpu: "500m"
      memory: "1024Mi"
```

5.3. 通过指定ECS规格创建ECI Pod

在某些业务场景下，存在着特殊的规格需求，例如：GPU、增强的网络能力、高主频、本地盘等。ECI支持通过指定ECS规格进行创建。本文介绍如何通过指定ECS规格创建ECI Pod。

规格说明

ECI指定规格完全参考ECS规格定义。ECI单价与对应规格的ECS价格保持一致，按秒计费。详情请参见ECS价格计算器。

您可以通过ECS 实例规格可购买地域总览，查询每个地域和可用区具体支持的ECS规格信息。目前支持的实例规格族如下所示：

- 通用型（1:4）实例规格族g6、g5、sn2ne（网络增强）
- 计算型（1:2）实例规格族c6、c5、sn1ne（网络增强）
- 内存型（1:8）实例规格族r6、r5、se1ne（网络增强）
- 密集计算型（1:1）实例规格族ic5
- 高主频计算型（1:2）实例规格族hfc6、hfc5
- 高主频通用型（1:4）实例规格族hfg6、hfg5

- GPU计算型实例规格族gn6i、gn6v、gn5i、gn5（不支持本地存储）
- 突发性能实例规格族t6、t5

使用示例

通过在Pod定义中设置annotations: k8s.aliyun.com/eci-use-specs，可以配置多个规格，以逗号分割。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-use-specs : "ecs.c5.large"  #根据需要替换ECS规格
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

5.4. 使用抢占式实例

抢占式实例是一种低成本竞价型实例，您可以对阿里云当前闲置的资源出价，获得资源后运行容器，直到因为其他客户的更高出价而被回收容器资源，从而降低部分场景下使用ECI实例的成本。本文介绍如何使用抢占式实例。

背景信息

抢占式实例相对于按量付费实例价格有一定的折扣，实际价格随供求波动，并按实际使用时长进行收费。更多信息，请参见抢占式实例概述。

抢占式实例创建成功后拥有一小时的保护周期，即在创建成功后第一个小时内，即使市场价格浮动超过了出价，抢占式实例也不会被释放，您可以在该抢占式实例上正常运行业务。超过保护周期后，每5分钟检测一次实例规格的当前市场价格和库存，如果某一时刻的市场价格高于出价或实例规格库存不足，抢占式实例会被释放。

- 抢占式ECI实例运行1小时后将会因为价格的波动、库存等原因被系统回收。
- 系统回收会提前3分钟产生准备释放的事件。
- 释放的ECI的资源会被回收，但是实例信息会依然保留，回收后实例不再收费并且状态变为Expired（已过期）。

在使用抢占式实例时，您需要遵循以下建议：

- 选择一个合理的出价，您的出价应该足够高，而且要充分考虑到市场价格的波动。之后您的抢占请求才会被接受处理，而且创建后才不会因为价格因素被释放。另外，出价还必须符合您根据自身业务评估后的预期。
- 建议您使用不受抢占式实例释放影响的存储介质来保存您的重要数据。例如，您可以使用独立创建的云盘（不能设置为随实例一起释放）存储数据，或者NAS等外部存储。

抢占式ECI实例依然支持两种创建方式：

- ECS InstanceType形式。

  计费会以该规格的按量市场价以及实时折扣为准。

- CPU/内存形式。

  ECI会自动匹配满足价格要求的ECS InstanceType，并以此规格作为计费的原始市场价，即实例的计费折扣都是基于该规格的市场价，而非对应的ECI的CPU/Memory的按量价格。该方式等效于传入InstanceType方式创建。

应用场景

抢占式实例适用于无状态的应用场景，例如可弹性伸缩的Web站点服务、图像渲染、大数据分析和大规模并行计算等。应用程序的分布度、可扩展性和容错能力越高，越适合使用抢占式实例节省成本和提升吞吐量。

您可以在抢占式实例上部署以下业务：

- 实时分析业务
- 大数据计算业务
- 可弹性伸缩的业务站点
- 图像和媒体编码业务
- 科学计算业务
- 地理空间勘测分析业务
- 网络爬虫业务
- 测试业务

使用方法

您可以通过在Pod层面设置annotations: k8s.aliyun.com/eci-spot-strategy和annotations: k8s.aliyun.com/eci-spot-price-limit。

- SpotAsPriceGo：系统自动出价，跟随当前市场实际价格。
- SpotWithPriceLimit：设置抢占实例价格上限。

具体配置方式：

- 设置 `k8s.aliyun.com/eci-spot-strategy` 为 `"SpotAsPriceGo"` 。
- 只有 `k8s.aliyun.com/eci-spot-strategy` 设置为 `"SpotWithPriceLimit"` 时， `k8s.aliyun.com/eci-spot-price-limit` 才会有效。

  设置实例每小时最高价格，支持最多3位小数。

  - `k8s.aliyun.com/eci-spot-strategy: "SpotAsPriceGo"`
  - `k8s.aliyun.com/eci-spot-price-limit: "0.250"`

Deployment示例

- SpotAsPriceGo策略

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-use-specs: "2-4Gi" #根据需要替换您接受的ECS规格。
    k8s.aliyun.com/eci-spot-strategy: "SpotAsPriceGo" #随市场出价策略。
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

- SpotWithPriceLimit策略

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-use-specs: "ecs.c5.large" #根据需要替换您接受的ECS规格。
    k8s.aliyun.com/eci-spot-strategy: "SpotWithPriceLimit" #最高价限定策略。
    k8s.aliyun.com/eci-spot-price-limit: "0.250" #小时最高单价。
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

5.5. 使用GPU实例

GPU实例内置了对应的Docker镜像，因此使用ECI GPU实例时无需安装Tensorflow、CUDA Toolkit等软件。本文介绍如何使用ECI GPU实例。

背景信息

当前ECI GPU支持的驱动版本为NVIDIA 460.73.01，可支持的CUDA Tookit版本为11.2。

ECI支持通过指定ECS GPU规格来创建ECI GPU实例。支持的ECS GPU规格，如下所示：

- GPU计算型实例规格族gn6v（NVIDIA V100），例如：ecs.gn6v-c8g1.2xlarge。
- GPU计算型实例规格族gn6i（NVIDIA T4），例如：ecs.gn6i-c4g1.xlarge。
- GPU计算型实例规格族gn5（NVIDIA P100），例如：ecs.gn5-c4g1.xlarge。
- GPU计算型实例规格族gn5i（NVIDIA P4），例如：ecs.gn5i-c2g1.large。

完整的ECS GPU规格定义，请参见实例规格族。

使用方法

在Pod定义中增加 `annotations: k8s.aliyun.com/eci-use-specs` 。

- 在Pod的 `metadata` 中添加指定规格的 `annotations` 。
- 在Container的 `resources` 中声明GPU资源。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-gpu-demo
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-use-specs: ecs.gn5i-c4g1.xlarge
  spec:
   containers:
   - name: nginx
    image: registry-vpc.cn-beijing.aliyuncs.com/eci_open/nginx:1.15.10
    resources:
      limits:
       nvidia.com/gpu: '1'
    ports:
    - containerPort: 80
```

5.6. 使用AMD实例

AMD实例依托神龙架构，将大量虚拟化功能卸载到专用硬件，降低虚拟化开销，提供稳定可预期的超高性能。本文介绍如何使用AMD实例。

适用场景

- 视频编解码
- 高网络包收发场景
- Web前端服务器
- 大型多人在线游戏（MMO）前端
- 测试开发，例如DevOps

规格说明

ECI支持通过指定ECS AMD规格进行实例的创建，支持ECS计算型实例规格族c6a，例如：`ecs.c6a.large`。更多信息，请参见实例规格族。

使用方法

Pod声明中增加`annotations: k8s.aliyun.com/eci-use-specs`。

在Pod的`metadata`中添加指定规格的`annotations`。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-use-specs : "ecs.c6a.xlarge" #根据需要替换ECS规格
  spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

5.7. 使用预留实例券

当业务是长时间运行的在线业务时，使用预留实例券可以抵扣ECI费用，从而降低长时间运行实例的费用。本文介绍如何使用预留实例券。

准备工作

1. 根据需要购买预留实例券。具体操作，请参见购买预留实例券。
2. 查看和管理预留实例券。具体操作，请参见拆分预留实例券。

> **说明** 预留实例券根据规则匹配按量ECI实例，具体匹配描述，请参见预留实例券与实例的匹配。

使用方式

预留实例券仅支持根据指定ECS规格创建ECI实例，具体操作，请参见通过指定ECS规格创建ECI Pod。

在Pod的 `templte` 中加入以下 `annotations` （本例中预留实例券购买的是 `ecs.c5.large` 实例）：

```
annotations:
  k8s.aliyun.com/eci-instance-type : "ecs.c5.large" #根据需要替换ECS规格ecs.规格族.规格大小，例如：（ecs.c6.3xlarge）
```

> **说明** `annotations` 需要添加到Pod的 `spec` 中，请根据实际需求填写ECS规格。ECS规格列表请参见实例规格族。

Deployment示例

`annotations` 需要添加到Pod的 `metadata` 中。

> **说明** 预留实例券（如果是可用区级别的）所在可用区应该与Kubernetes集群所在可用区一致，否则Serverless Kubernetes集群中创建出来的ECI实例与预留实例券无法匹配。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-instance-type : "ecs.c5.large" #根据需要替换ECS规格spec:
   containers:
   - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

查看预留实例券账单和抵扣信息

1. 登录ECS管理控制台。
2. 在左侧导航栏，选择**实例与镜像 > 预留实例券**。
3. 在顶部菜单栏左上角处，选择地域。
4. 单击对应预留实例券实例**操作**列中的**查看账单**。
5. 在**资源实例管理**页面，单击**使用明细**页签。

   可以查看对应预留实例券的抵扣明细，记录在每个小时的出账周期内该预留实例券抵扣的ECS或ECI实例信息。

> **说明** 预留实例券有计算力的概念，1计算力可以简单理解为1 vCPU，抵扣时长（小时）等于计算力乘以小时。

5.8. 创建多可用区的ECI Pod

当您为了应对突发流量而进行业务的快速水平扩容，或者启动大量实例进行Job任务处理时，可能会遇到可用区对应规格实例库存不足或者指定的交换机IP耗尽等情况，从而导致ECI实例创建失败。使用ASK的多可用区特性可以提高ECI实例的创建成功率。

背景信息

- ECI会把创建Pod的请求分散到所有的vSwitch中，从而达到分散压力的效果。
- 如果创建Pod请求在某一个vSwitch中遇到没有库存的情况，会自动切换到下一个vSwitch继续尝试创建。

使用方式

1. 创建ASK集群时选择已有专有网络，并配置多个不同可用区的vSwitch。创建集群请参见创建Serverless Kubernetes集群。

2. 修改集群可用区配置。

   当集群创建成功后，需要修改集群的vSwitch配置。您无需重建集群，执行以下命令，根据实际需求修改 `kube-system/eci-profile configmap` 中的 `vswitch` 字段，修改后即时生效。

```
kubectl -n kube-system edit cm eci-profile
```

```
apiVersion: v1
data:
 kube-proxy: "true"
 privatezone: "true"
 quota-cpu: "192000"
 quota-memory: 640Ti
 quota-pods: "4000"
 region: cn-hangzhou
 resourcegroup: ""
 securitygroup: sg-xxx
 vpc: vpc-xxx
 vswitch: vsw-xxx,vsw-yyy,vsw-zzz
kind: ConfigMap
```

常见问题

- 如何在ASK集群中加入其他Node?

  ASK集群没有Node节点的概念。如果您需要让ECI可以部署到其他可用区，可以在kube-system下名为eci-profile的configmap中，追加一个不同可用区的VSW，逗号分隔即可。

- 如何指定Pod部署到不同的可用区?

  部署Pod的时候默认是根据可用区下是否有足够的资源进行随机调度。

  匹配VK，通过三个Annotation实现对多可用区和多规格支持:

  - - k8s.aliyun.com/eci-schedule-strategy: "VSwitchOrdered"
  - - k8s.aliyun.com/eci-vswitch: "vsw-11111, vsw-22222"
  - - k8s.aliyun.com/eci-use-specs: "ecs.c5.4xlarge, ecs.c6.4xlarge,2-4Gi"

  VSwitchOrdered会根据指定的k8s.aliyun.com/eci-vswitch列表顺序创建ECI。通过k8s.aliyun.com/eci-schedule-strategy设置VSwitchRandom或VSwitchOrdered，在创建ECI时实现随机调度到可用区或者按照注解中的vsw顺序来调度。

5.9. ECI Pod Annotation

Kubernetes集群通过虚拟节点创建Pod到ECI时，为充分使用ECI提供的功能，在不改变Kubernetes语义的前提下，您可以根据需求为Pod添加Annotation。 本文为您介绍ECI实例支持的Annotation及其配置示例。

ECI实例目前支持的Annotation如下表所示。

> **说明**
>
> 下表列举的Annotation仅适用于创建到虚拟节点上的Pod，即ECI实例，调度到ECS上的Pod不受这些Annotation影响。

参数	示例值	描述	相关文档
k8s.aliyun.com/eci-security-group	sg-bp1dktddjsg5nktv****	安全组ID。	配置安全组
k8s.aliyun.com/eci-vswitch	vsw-bp1xpiowfm5vo8o3c****	交换机ID，支持指定多个交换机实现多可用区功能。	多可用区创建实例
k8s.aliyun.com/eci-schedule-strategy	VSwitchOrdered	多可用区调度策略。取值范围： • VSwitchOrdered：按顺序 • VSwitchRandom：随机	
k8s.aliyun.com/eci-ram-role-name	AliyunECIContainerGroupRole	RAM角色，赋予ECI访问阿里云产品的能力。	授权RAM角色
k8s.aliyun.com/eci-use-specs	2-4Gi,4-8Gi,ecs.c6.xlarge	ECI实例规格，支持指定多规格，包括指定vCPU和内存，或者ECS规格。	多规格创建实例
k8s.aliyun.com/eci-spot-strategy	SpotAsPriceGo	抢占式实例策略。取值范围： • SpotAsPriceGo：系统自动出价，跟随当前市场实际价格。 • SpotWithPriceLimit：设置抢占实例价格上限。	创建抢占式实例
k8s.aliyun.com/eci-spot-price-limit	0.5	抢占式实例价格。仅当k8s.aliyun.com/eci-spot-strategy设置为SpotWithPriceLimit时有效。	
k8s.aliyun.com/eci-cpu-option-core	2	CPU物理核心数。	自定义CPU选项
k8s.aliyun.com/eci-cpu-option-ht	1	每核线程数。	
k8s.aliyun.com/eci-reschedule-enable	"true"	是否开启ECI重调度。	无，详见下文。

参数	示例值	描述	相关文档
k8s.aliyun.com/pod-fail-on-create-err	"true"	创建失败的ECI实例是否体现Failed状态。	无，详见下文。
k8s.aliyun.com/eci-image-snapshot-id	imc-2zebxkiifuyzzlhl****	指定镜像缓存ID。 说明 使用镜像缓存支持手动指定和自动匹配两种方式，建议使用自动匹配方式。	使用镜像缓存CRD加速创建Pod
k8s.aliyun.com/eci-image-cache	"true"	自动匹配镜像缓存。 说明 使用镜像缓存支持手动指定和自动匹配两种方式，建议使用自动匹配方式。	
k8s.aliyun.com/acr-instance-id	cri-j36zhodptmyq****	ACR企业版实例ID。	配置ACR企业版免密
k8s.aliyun.com/eci-eip-instanceid	eip-bp1q5n8cq4p7f6dzu****	EIP实例ID。	为ECI实例绑定EIP
k8s.aliyun.com/eci-with-eip	"true"	是否自动创建并绑定EIP。	
k8s.aliyun.com/eip-bandwidth	5	EIP带宽。	
k8s.aliyun.com/eip-common-bandwidth-package-id	cbwp-2zeukbj916scmj51m****	共享带宽包ID。	

参数	示例值	描述	相关文档
k8s.aliyun.com/eip-isp	BGP	EIP线路类型，仅按量付费的EIP支持指定。取值范围： • BPG：BGP（多线）线路 • BGP_PRO：BGP（多线）精品线路	
k8s.aliyun.com/eip-internet-charge-type	PayByBandwidth	EIP的计量方式。取值范围： • PayByBandwidth：按带宽计费 • PayByTraffic：按流量计费	
k8s.aliyun.com/eci-enable-ipv6	"true"	是否分配IPv6。 说明 目前仅支持为Pod分配1个IPv6地址，k8s.aliyun.com/eci-enable-ipv6和k8s.aliyun.com/eci-ipv6-count的配置效果一致，您可以选择其一。	配置IPv6地址
k8s.aliyun.com/eci-ipv6-count	1	IPv6个数，目前仅支持配置1个。 说明 目前仅支持为Pod分配1个IPv6地址，k8s.aliyun.com/eci-enable-ipv6和k8s.aliyun.com/eci-ipv6-count的配置效果一致，您可以选择其一。	
kubernetes.io/ingress-bandwidth	40M	入方向带宽。	ECI实例进行带宽限速
kubernetes.io/egress-bandwidth	20M	出方向带宽。	
k8s.aliyun.com/eci-extra-ephemeral-storage	50Gi	临时存储空间大小。	自定义临时存储空间大小
k8s.aliyun.com/eci-core-pattern	/pod/data/dump/core	Core dump文件保存目录。	查看Core dump文件

参数	示例值	描述	相关文档
k8s.aliyun.com/eci-ntp-server	100.100.*.*	NTP Server。	为Pod配置NTP服务

设置安全组

Virtual Kubelet启动时，将通过环境变量设置默认的安全组。所有创建在虚拟节点上的Pod默认使用Virtual Kubelet配置的。如果您有特殊需求，可以通过添加Annotation的方式，为Pod设置特定的安全组。

配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo
 labels:
  app: nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      k8s.aliyun.com/eci-security-group: "sg-bp1dktddjsg5nktv****"    #设置安全组。
    labels:
      app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
   nodeName: virtual-kubelet
```

多可用区创建Pod

在创建Pod时，您可以指定多个交换机实现多可用区功能。系统将根据资源的库存情况，选择合适的可用区创建Pod。更多信息，请参见多可用区创建实例。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-vswitch: "vsw-bp1xpiowfm5vo8o3c****,vsw-bp1rkyjgr1xwoho6k****"  #指定多个交换机ID。
  k8s.aliyun.com/eci-schedule-strategy: "VSwitchOrdered"  #设置多可用区调度策略。
 name: nginx-test
spec:
 containers:
 - name: nginx
  image: nginx:latest
```

设置RAM角色

您可以通过添加Annotation的方式为Pod设置RAM角色，授予Pod访问阿里云产品的能力。

> **注意**
>
> 请确保RAM角色的受信服务为云服务器。

配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: set-ram-role
 labels:
  app: vk
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      k8s.aliyun.com/eci-ram-role-name : "AliyunECIContainerGroupRole"  #设置RAM角色。
    labels:
      app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
   nodeName: virtual-kubelet
```

指定实例规格创建Pod

您可以通过添加Annotation的方式为Pod指定允许使用的实例规格。在创建Pod时，如果遇到库存不足的情况，将按照顺序遍历指定的实例规格，保证创建成功率。更多信息，请参见多规格创建实例。

> **说明**
>
> 实例规格支持直接指定vCPU和内存，或者指定具体的ECS规格。如果业务有特殊的规格需求，例如：GPU、高主频、本地盘等，请明确指定ECS规格。更多信息，请参见指定ECS规格创建实例。

配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: vk-cos-use
 labels:
  app: cos
spec:
 replicas: 1
 selector:
  matchLabels:
   app: cos
 template:
  metadata:
    annotations:
      "k8s.aliyun.com/eci-use-specs": "2-4Gi,4-8Gi,ecs.c6.xlarge" #支持指定多个规格，规格可以是vCPU和内存，或者具体的ECS规格。
    labels:
      app: cos
  spec:
   containers:
   - name: u1
    image: "registry-vpc.cn-beijing.aliyuncs.com/lxx/cos-4g"
   nodeName: virtual-kubelet
```

创建抢占式实例

对于无状态应用、Job任务等，使用抢占式实例运行，可以有效地节约实例使用成本。您可以通过添加Annotation的方式，创建抢占式ECI示例。更多信息，请参见创建抢占式实例。

配置示例如下：

```
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
   annotations:
    k8s.aliyun.com/eci-use-specs : "ecs.c5.large"  #根据需要替换ECS实例规格。
    k8s.aliyun.com/eci-spot-strategy: "SpotWithPriceLimit"  #采用自定义设置价格上限的策略。
    k8s.aliyun.com/eci-spot-price-limit: "0.250"  #设置每小时价格上限。
  spec:
  #  nodeSelector:
  #    env: test-team
   containers:
   - name: nginx
    image: nginx:1.7.9 # replace it with your exactly <image_name:tags>
    ports:
    - containerPort: 80
```

自定义设置CPU选项

对于一台ECI实例，CPU选项由CPU物理核心数和每核线程数决定。根据您创建ECI实例的方式，部分ECI实例支持自定义CPU选项。更多信息，请参见自定义CPU选项。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-use-specs : "ecs.c6.2xlarge"   #指定支持自定义CPU选项的ECS规格。
  k8s.aliyun.com/eci-cpu-option-core: "2"          #自定义设置CPU物理核心数为2。
  k8s.aliyun.com/eci-cpu-option-ht: "1"            #自定义设置每核线程数为1，即关闭超线程。
 name: nginx-test
spec:
 containers:
 - name: nginx
  image: nginx:latest
 restartpolicy: Always
```

设置ECI重调度

调度Pod到虚拟节点时，可能会碰到调度失败的情况，您可以通过添加Annotation的方式为Pod开启重调度，即使异步调度失败了，仍会一直保持调度，不返回失败。

配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: set-eci
 labels:
  app: vk
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      k8s.aliyun.com/eci-reschedule-enable: "true"   #开启ECI重调度。
    labels:
      app: nginx
  spec:
   containers:
   - name: nginx
     image: nginx:latest
   nodeName: virtual-kubelet
```

Pod创建失败置为Failed

默认情况下，每个Pod在创建时，如果遇到错误，会重试一定次数，如果还是Failed，则Pod会处于Pending状态。对于一些Job类型的任务，您可能希望直接体现Failed状态。此时，您可以通过添加Annotation的方式，设置Pod遇到创建失败时，体现Failed状态。

配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: set-pod-fail-on-create-err
 labels:
  app: vk
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      k8s.aliyun.com/pod-fail-on-create-err: "true" #设置Pod创建失败时，状态为Failed。
    labels:
      app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
   nodeName: virtual-kubelet
```

使用镜像缓存CRD加速创建Pod

使用镜像缓存技术可以加速创建Pod，您可以通过添加Annotation的方式，为Pod指定使用的镜像缓存，或者开启自动匹配镜像缓存。更多信息，请参见使用镜像缓存CRD加速创建Pod。

配置示例如下：

- 指定镜像缓存

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-image-snapshot-id: imc-2ze5tm5gehgtiiga****  #指定使用的镜像缓存。
 name: nginx-imagecache-id
spec:
 containers:
 - image: nginx:1.7.9
  imagePullPolicy: IfNotPresent
  name: nginx
  resources:
   limits:
    cpu: 300m
    memory: 200Mi
   requests:
    cpu: 200m
    memory: 100Mi
 nodeName: virtual-kubelet
```

- 自动匹配

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-image-cache: "true"  #开启自动匹配镜像缓存。
 name: nginx-auto-match
spec:
 containers:
 - image: nginx:1.7.9
  imagePullPolicy: IfNotPresent
  name: nginx
  resources:
   limits:
    cpu: 300m
    memory: 200Mi
   requests:
    cpu: 200m
    memory: 100Mi
 nodeName: virtual-kubelet
```

指定ACR企业版实例

阿里云容器服务ACR支持免密拉取，您可以通过添加Annotation的方式指定ACR企业版实例，从对应的镜像仓库中拉取镜像。更多信息，请参见配置ACR企业版免密。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/acr-instance-id: cri-j36zhodptmyq****  #指定ACR企业版实例。
 name: cri-test
spec:
 containers:
 - image: test****-registry.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0  #使用公网拉取镜像。
  imagePullPolicy: Always
  name: nginx
 restartPolicy: Never
```

绑定弹性公网IP

如果Pod有公网通信的需求，您可以为其绑定EIP。更多信息，请参见如何给Pod挂载弹性公网IP。

配置示例如下：

- 自动创建

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 annotations:
  k8s.aliyun.com/eci-with-eip: "true"   #开启自动创建EIP。
  k8s.aliyun.com/eip-bandwidth: "10"   #设置带宽，默认为5，单位为Mbps。
spec:
 containers:
 - image: registry-vpc.cn-hangzhou.aliyuncs.com/jovi/nginx:alpine
  imagePullPolicy: Always
  name: nginx
  ports:
  - containerPort: 80
   name: http
   protocol: TCP
 restartPolicy: OnFailure
```

- 指定EIP

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 annotations:
  k8s.aliyun.com/eci-eip-instanceid: "eip-bp1q5n8cq4p7f6dzu****"  #指定EIP。
spec:
 containers:
 - image: registry-vpc.cn-hangzhou.aliyuncs.com/jovi/nginx:alpine
  imagePullPolicy: Always
  name: nginx
  ports:
  - containerPort: 80
   name: http
   protocol: TCP
 restartPolicy: OnFailure
```

配置IPv6

相比IPv4，IPv6不仅可以解决网络地址资源有限的问题，还可以解决多种接入设备连入互联网障碍的问题。您可以通过添加Annotation的方式为Pod分配IPv6地址。更多信息，请参见配置IPv6地址。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 annotations:
  k8s.aliyun.com/eci-enable-ipv6: "true"  #开启自动分配IPv6。
spec:
 containers:
 - name: nginx
  image: nginx
 nodeName: virtual-kubelet
```

设置出入方向带宽

CI支持配置流入和流出的网络带宽值。您可以通过添加Annotation的方式，为Pod指定出方向和入方向的带宽值进行限速。更多信息，请参见ECI实例进行带宽限速。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: eci-qos
 annotations:
  kubernetes.io/ingress-bandwidth: 40M  #设置入方向带宽。
  kubernetes.io/egress-bandwidth: 10M  #设置出方向带宽。
spec:
 containers:
 - name: nginx
  image: nginx:latest
  command: ["bash","-c","sleep 100000"]
```

设置临时存储空间大小

ECI实例默认提供20 GiB的免费存储空间，如果该存储空间大小无法满足您的需求，您可以通过添加Annotation的方式，自定义增加临时存储空间大小。更多信息，请参见自定义临时存储空间大小。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: test
 annotations:
  k8s.aliyun.com/eci-extra-ephemeral-storage: "50Gi" #自定义设置临时存储空间大小。
spec:
 containers:
 - name: nginx
  image: nginx:latest
  imagePullPolicy: IfNotPresent
 restartPolicy: Always
 nodeName: virtual-kubelet
```

设置Core dump文件的保存目录

容器进程异常退出会产生Core dump文件，默认情况下，Core dump文件的名称为core.pid，保存在当前目录下。您可以通过添加Annotation的方式，自定义设置Core dump文件保存目录。更多信息，请参见查看Core dump文件。

配置示例如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: test
 annotations:
  k8s.aliyun.com/eci-core-pattern: "pod/data/dump/core" # 设置Core dump文件保存目录。
spec:
 containers:
 - image: nginx:latest
  name: test-container
  volumeMounts:
  - mountPath: /pod/data/dump/
   name: default-volume
 volumes:
 - name: nfs
  nfs:
   server: 143b24****-gfn3.cn-beijing.nas.aliyuncs.com
   path: /dump/
   readOnly: false
```

为Pod设置NTP服务

您可以为Pod添加k8s.aliyun.com/eci-ntp-server的Annotation来配置NTP服务。更多信息，请参见为Pod配置NTP服务。

配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: set-ngnix-ntp
 labels:
  app: vk
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      k8s.aliyun.com/eci-ntp-server: 100.100.5.*,100.100.5.* #设置NTP服务器地址。
    labels:
      app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
   nodeName: virtual-kubelet
```

5.10. 为Pod配置NTP服务

本文主要介绍如何为运行在virtual kubelet上的Pod配置NTP服务。当您在部署应用时，如果需要Pod内的容器能与NTP服务进行时间同步，您可以参考本文进行配置。

前提条件

已将virtual-kubelet升级到最新版本。

背景信息

不同类型的kubernetes集群升级virtual kubelet到最新版本的方式如下：

- Serverless kubernetes：由管理员统一负责升级。
- 托管版kubernetes：您需要自行升级。
- 专有版kubernetes：您需要自行升级。
- 自建kubernetes：您需要自行升级。

操作步骤

您需要在Pod的annotations中增加 `k8s.aliyun.com/eci-ntp-server` 注解，设置需要配置的NTP服务的IP地址。

1. 创建配置NTP服务的yaml文件。

   ```
   vim set-ntp-pod.yaml
   ```

   以下为yaml文件的内容示例：

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-ntp-server: 10.10.5.1 # 设置您的NTP服务的IP地址
 name: set-custom-ntp
spec:
 nodeName: virtual-kubelet
 containers:
  - image: centos:latest
    command:
     - sleep
     - "3600"
    imagePullPolicy: IfNotPresent
    name: centos
```

2. 将yaml文件中的配置应用到Pod。

```
kubectl apply -f set-ntp-pod.yaml
```

验证结果

登录到容器，验证NTP服务是否设置成功。

1. 获取Pod信息。

```
kubectl get pod/set-custom-ntp
```

返回示例如下：

```
NAME          READY  STATUS   RESTARTS  AGE
set-custom-ntp  1/1     Running  0         7m20s
```

2. 进入容器。

```
kubectl exec set-custom-ntp -it -- bash
```

3. 查询容器的时间来源。

```
chronyc sources
```

如果返回了NTP服务的IP地址，则表示设置成功。返回示例如下：

```
210 Number of sources = 1
MS Name/IP address      Stratum Poll Reach LastRx Last                sample
===============================================================================
^* 10.10.5.1             2      6   377    35    +40us[ +135us] +/-  14ms
```

5.11. 为Pod配置时区

本文主要为您介绍如何为运行在virtual-kubelet上的Pod配置不同的时区，当用户使用Pod部署应用时，希望Pod能指定不同地点的时区。您可以参考此文档。

前提条件

升级 virtual-kubelet 到最新版本，升级方式：

- 阿里云Serverless kubernetes集群：由管理员统一负责升级
- 阿里云托管kubernetes集群：用户更新
- 阿里云专有kubernetes集群：用户更新
- 阿里云自建kubernetes集群：用户更新

Yaml示例

您需要先创建一个configmap，导入你需要指定的时区，其他时区使用/usr/share/zoneinfo/Asia/目录下文件。

```
$ kubectl create configmap tz --from-file=/usr/share/zoneinfo/Asia/Shanghai
```

创建ECI pod实例，主要就是把configmap mount到/etc/localtime/Shanghai目录下即可。

```
$ cat set-timezone.yaml
apiVersion: v1
kind: Pod
metadata:
 name: timezone
spec:
 containers:
 - name: timezone
  image: registry-vpc.cn-beijing.aliyuncs.com/eci_open/busybox:1.30
  command: [ "sleep", "10000" ]
  volumeMounts:
   - name: tz
     mountPath: /etc/localtime
     subPath: Shanghai
 volumes:
  - name: tz
    configMap:
     name: tz
 nodeSelector:
  type: virtual-kubelet
 tolerations:
 - key: virtual-kubelet.io/provider
  operator: Exists
```

创建Pod：

```
$ kubectl apply -f set-timezone.yaml
```

验证

登录到容器，验证时区是否设置成功。

```
$ kubectl get pod/timezone
NAME          READY  STATUS   RESTARTS  AGE
set-timezone  1/1    Running  0         7m20s
$ kubectl exec timezone -it -- sh
/ # date -R
Fri, 01 May 2020 10:00:11 +0800
/ #
```

已经在容器中成功设置了时区。

5.12. 查看Core dump文件

本文介绍如何设置Core dump文件的保存目录，以便在容器异常终止时查看分析Core dump文件，找出问题原因。

背景信息

在Linux中，如果程序突然异常终止或者崩溃时，操作系统会将程序当时的内存状态记录下来，保存在一个文件中，这种行为就叫做Core dump。此时，您可以查看分析Core dump文件，找出问题原因。

Linux中支持Core dump（Action为Core）的Signal如下图所示。

Signal	Standard	Action	Comment
SIGABRT	P1990	Core	Abort signal from abort(3)
SIGALRM	P1990	Term	Timer signal from alarm(2)
SIGBUS	P2001	Core	Bus error (bad memory access)
SIGCHLD	P1990	Ign	Child stopped or terminated
SIGCLD	-	Ign	A synonym for SIGCHLD
SIGCONT	P1990	Cont	Continue if stopped
SIGEMT	-	Term	Emulator trap
SIGFPE	P1990	Core	Floating-point exception
SIGHUP	P1990	Term	Hangup detected on controlling terminal or death of controlling process
SIGILL	P1990	Core	Illegal Instruction
SIGINFO	-		A synonym for SIGPWR
SIGINT	P1990	Term	Interrupt from keyboard
SIGIO	-	Term	I/O now possible (4.2BSD)
SIGIOT	-	Core	IOT trap. A synonym for SIGABRT
SIGKILL	P1990	Term	Kill signal
SIGLOST	-	Term	File lock lost (unused)
SIGPIPE	P1990	Term	Broken pipe: write to pipe with no readers; see pipe(7)
SIGPOLL	P2001	Term	Pollable event (Sys V). Synonym for SIGIO
SIGPROF	P2001	Term	Profiling timer expired
SIGPWR	-	Term	Power failure (System V)
SIGQUIT	P1990	Core	Quit from keyboard
SIGSEGV	P1990	Core	Invalid memory reference
SIGSTKFLT	-	Term	Stack fault on coprocessor (unused)
SIGSTOP	P1990	Stop	Stop process
SIGTSTP	P1990	Stop	Stop typed at terminal
SIGSYS	P2001	Core	Bad system call (SVr4); see also seccomp(2)
SIGTERM	P1990	Term	Termination signal
SIGTRAP	P2001	Core	Trace/breakpoint trap
SIGTTIN	P1990	Stop	Terminal input for background process
SIGTTOU	P1990	Stop	Terminal output for background process
SIGUNUSED	-	Core	Synonymous with SIGSYS
SIGURG	P2001	Ign	Urgent condition on socket (4.2BSD)
SIGUSR1	P1990	Term	User-defined signal 1
SIGUSR2	P1990	Term	User-defined signal 2
SIGVTALRM	P2001	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	P2001	Core	CPU time limit exceeded (4.2BSD); see setrlimit(2)
SIGXFSZ	P2001	Core	File size limit exceeded (4.2BSD); see setrlimit(2)

更多信息，请参见Core dump file。

功能概述

对于ECI来说，因为VM是托管的，所以无法指定Core dump文件的文件名和文件目录。默认情况下，Core dump文件的名称为core.pid，保存在当前目录下。

```
/ # cd /pod/
/pod # ls
data
/pod # sleep 10
^\Quit (Core dumped)（按Ctrl+\触发）
/pod # ls
core.45  data
```

即使容器进程异常退出产生了Core dump文件，但Core dump文件会随容器退出而丢失，因此也无法离线查看Core dump文件，进行后续的分析。

针对上述情况，ECI支持自定义设置Core dump文件的保存目录。您可以将Core dump文件保存到外挂存储中，即使容器退出了，也可以获取到Core dump文件进行离线查看和分析。

配置方式如下：

> **注意**
>
> 配置的路径不能以 | 开头，即不能通过Core dump来配置可执行程序。

- Kubernetes

```
annotations:
  k8s.aliyun.com/eci-core-pattern: "/xx/xx/core"
```

- OpenAPI

```
CorePattern = "/xx/xx/core"
```

配置示例

对于Core dump文件，一般是为了离线分析，因此在设置Core dump文件的保存目录时，一般会选择外挂存储，而不是保存在容器本地目录。目前支持的外挂存储包括云盘、NFS等。下文以NFS为例进行介绍。

1. 挂载NFS，并设置Core dump文件的保存目录。

   如下示例，将NFS挂载到容器的/pod/data/dump/目录，然后设置Core dump文件的保存目录为：/pod/data/dump/core。此时，容器的所有Core dump文件将自动保存到该目录下，并同步到NFS。即使该容器释放了，您仍可以从NFS中获取Core dump文件进行分析。

   ```
   apiVersion: v1
   kind: Pod
   metadata:
    name: test
    annotations:
     k8s.aliyun.com/eci-core-pattern: "pod/data/dump/core"
   spec:
    containers:
    - image: nginx:latest
     name: test-container
     volumeMounts:
     - mountPath: /pod/data/dump/
       name: default-volume
    volumes:
    - name: default-volume
     nfs:
       server: 143b24****-gfn3.cn-beijing.nas.aliyuncs.com
       path: /dump/
       readOnly: false
   ```

2. 在容器任意的目录下触发Core dump。

   如下示例，可以看到设置的Core dump文件的名称和保存目录已经生效。

```
/ # ls
bin  dev  etc  home pod  proc root sys  tmp  usr  var
/ # sleep 10
^\Quit (core dumped)（按Ctrl+\触发）
/ # ls
bin  dev  etc  home pod  proc root sys  tmp  usr  var
/ # cd /pod/data/dump/
/pod/data/dump # ls
core.12
```

3. 释放该台ECI实例。
4. 将NFS挂载到新的一台ECI实例上，然后登录实例查看Core dump文件。

   如下示例，可以看到Core dump文件并没有丢失，您可以进行查看分析。

```
/ # cd /pod/data/dump/
/pod/data/dump # ls
core.12
```

6.集群

6.1. 创建Serverless Kubernetes集群

本文介绍如何在容器服务管理控制台快速创建阿里云Serverless Kubernetes（ASK）集群。

前提条件

- 开通容器服务ACK，且授权默认角色和开通相关云产品。具体操作，请参见首次使用容器服务Kubernetes版。
- 登录弹性容器实例控制台，开通ECI服务。

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**ASK集群**页签，然后完成集群配置。

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	选择集群规格，支持**标准版**和**Pro版**。 选中**Pro版**创建ASK Pro版集群。更多信息，请参见ASK Pro版集群概述。
地域	选择集群所在的地域。
Kubernetes版本	显示当前ASK支持的Kubernetes版本。
专有网络	设置集群的网络。Kubernetes集群仅支持专有网络。支持**自动创建**和**使用已有**的VPC。 ○ **自动创建**：集群会自动新建一个VPC，并在VPC中自动创建NAT网关以及配置SNAT规则。 ○ **使用已有**：您可以在已有VPC列表中选择所需的VPC和交换机。如需访问公网，例如下载容器镜像，要配置NAT网关，建议将容器镜像上传到集群所在区域的阿里云镜像服务，并通过内网VPC地址拉取镜像。 详情请参见创建和管理专有网络。
可用区	选择集群所在的可用区。

配置项	描述
配置SNAT	设置是否为专有网络创建NAT网关并配置SNAT规则。 仅当**专有网络**选择为**自动创建**时，需要设置该选项。 **说明** 若您选择**自动创建**VPC，可选择是否自动配置SNAT网关。若选择不自动配置SNAT，您可自行配置NAT网关实现VPC安全访问公网环境，并且手动配置SNAT，否则VPC内实例将不能正常访问公网。 详情请参见创建公网NAT网关实例。
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。
API Server访问	ASK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，ASK会创建一个EIP，并挂载到SLB上。此时，Kubernetes API服务（即API Server）会通过EIP的6443端口暴露出来，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。 更多信息，请参见控制集群API Server的公网访问能力。
服务发现	设置集群的服务发现，支持**不开启**、**PrivateZone**和**CoreDNS**三种方式。 **说明** ○ PrivateZone：基于阿里云专有网络VPC环境的私有DNS服务。该服务允许您在自定义的一个或多个VPC中将私有域名映射到IP地址。 ○ CoreDNS：是一个灵活可扩展的DNS服务器，也是Kubernetes标准的服务发现组件。

配置项	描述
Ingress	设置是否安装Ingress组件。Ingress支持**不安装**、**Nginx Ingress**和**ALB Ingress**三种方式。 ○ Nginx Ingress：基于社区版的ingress-nginx进行了优化，为您的Kubernetes集群提供灵活可靠的路由服务（Ingress）。更多信息，请参见Nginx Ingress概述。 ○ 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。更多信息，请参见ALB Ingress概述。
监控服务	设置是否安装metrics-server集群基础监控组件。metrics-server是阿里云容器服务Kubernetes版基于社区开源监控组件进行改造和增强的离线监控数据组件，提供查看集群离线监控数据功能，提供HPA和基础资源监控的能力。您可以通过Metrics API获取到采集的监控数据。
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。 不开启日志服务时，将无法使用集群审计功能。日志服务详情请参见快速入门。
Knative	设置是否开启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，详细介绍请参见概述。
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。
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	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。  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。

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。
服务协议	创建集群前，需阅读并选中**《无服务器Kubernetes服务协议》**。

5. 在页面右侧，单击**创建集群**，在弹出的**当前配置确认**页面，单击**确定**，启动部署。

后续步骤

- 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Serverless集群。

- 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。

6.2. 通过阿里云CLI创建Serverless Kubernetes集群

阿里云CLI是基于阿里云开放API建立的管理工具。您可以通过该工具调用阿里云开放API来管理阿里云产品。

前提条件

在使用阿里云CLI之前，您需要配置调用阿里云资源所需的凭证信息、地域、语言等。更多信息，请参见简介。

安装配置阿里云CLI和kubectl

Cloud Shell默认安装配置了阿里云CLI和账号信息，无需任何额外配置。如果您不使用Cloud Shell，您需要安装和配置如下组件。

1. 安装阿里云CLI。
   - 在Linux上安装阿里云CLI，具体操作，请参见在Linux上安装阿里云CLI。
   - 在macOS上安装阿里云CLI。
     - 在macOS上安装阿里云CLI。
     - 您也可以通过包管理器工具安装CLI。请参考Homebrew，安装包管理工具后，执行以下命令安装CLI。

       ```
       brew install aliyun-cli
       ```
   - 在Windows上安装阿里云CLI，具体操作，请参见在Windows上安装阿里云CLI。
2. 配置阿里云CLI。执行以下命令，创建环境变量，用于存放身份认证信息。

   ```
   aliyun configure
   ```

   预期输出：

   ```
   Configuring profile 'default' in 'AK' authenticate mode...
   Access Key Id []: ************
   Access Key Secret []: ************
   Default Region Id []: cn-beijing
   Default Output Format [json]: json (Only support json)
   Default Language [zh|en] en:
   Saving profile[default] ...Done.
   Configure Done!!!
   ..............88888888888888888888 ........=88888888888888888D=..............
   ...........88888888888888888888 ..........D88888888888888888888I...........
   .........,8888888888888ZI: ............................=Z88D8888888888D..........
   .........+88888888 ..................................................88888888D..........
   .........+88888888 .......Welcome to use Alibaba Cloud.......O8888888D..........
   .........+88888888 ..............  ************* ..............O8888888D..........
   .........+88888888 .... Command Line Interface(Reloaded) ....O8888888D..........
   .........+88888888.....................................................88888888D..........
   ..........D888888888888DO+. ............................?ND888888888888D..........
   ............O888888888888888888888.............D888888888888888888888=............
   ..............  .:D888888888888888888.........78888888888888888888O ..............
   ```

安装和设置kubectl客户端，具体操作，请参见Install and Set Up kubectl。

创建Serverless Kubernetes集群

1. 创建一个工作目录，并且在工作目录下创建 `create.json` 文件。

   `create.json` 文件请求示例如下：

```
POST /clusters HTTP/1.1
<公共请求头>
{
  "cluster_type":"Ask",
  "name":"test-ask",
  "region_id":"cn-hangzhou",
  "endpoint_public_access":false,
  "private_zone":false,
  "nat_gateway":true,
  "tags":[
    {
      "key":"k-aa",
      "value":"v-aa"
    }
  ],
  "deletion_protection":false,
  "addons":[
    {
      "name":"logtail-ds"
    }
  ],
  "zone_id":"cn-hangzhou-i"
}
```

该示例文件描述了Serverless Kubernetes集群配置信息。更多信息，请参考创建Serverless Kubernetes集群。

2. 执行以下命令创建Serverless Kubernetes集群。

```
aliyun cs  POST /clusters --header "Content-Type=application/json" --body "$(cat create.json)"
```

预期输出：

```
{
  "cluster_id": "************************",
  "instanceId": "************************",
  "request_id": "**********-****-****-****-************",
  "task_id": "*-************"
}
```

3. 创建成功后，执行以下命令查看集群实例。

```
aliyun cs GET /clusters/<YOUR-CLUSTER-ID>
```

预期输出：

```
{
  "-": "PayByTraffic",
  "cluster_healthy": "",
  "cluster_id": "************************",
  "cluster_spec": "",
  "cluster_type": "Ask",
  "created": "2020-07-23T10:02:18+08:00",
  "current_version": "v1.16.6-aliyun.1",
  "data_disk_category": "cloud",
  "data_disk_size": 0,
  "deletion_protection": false,
  "docker_version": "",
  "enabled_migration": false,
  "external_loadbalancer_id": "lb-*********",
  "gw_bridge": "",
  "init_version": "v1.16.6-aliyun.1",
  "instance_type": "",
  "name": "test-serverless-k8s",
  "need_update_agent": false,
  "network_mode": "vpc",
  "node_status": "",
  "private_zone": false,
  "profile": "ask.v2",
  "region_id": "cn-beijing",
  "resource_group_id": "rg-*********",
  "security_group_id": "sg-*********",
  "size": 0,
  "state": "running",
  "subnet_cidr": "172.16.0.0/16",
  "swarm_mode": false,
  "tags": [
    {
      "key": "env",
      "value": "test"
    }
  ],
  "updated": "2020-07-23T10:05:11+08:00",
  "vpc_id": "vpc-*********",
  "vswitch_cidr": "",
  "vswitch_id": "vsw-*********",
  "worker_ram_role_name": ""
}
```


4. 执行以下命令获取当前的集群配置信息。

```
KUBECONFIG=<YOUR-LOCAL-KUBECONFIG-PATH>
aliyun cs GET /k8s/$cluster/user_config | jq -r '.config' > $KUBECONFIG
```

预期输出：

```
NAME        STATUS  AGE
default     Active  7m43s
kube-node-lease  Active  7m45s
kube-public    Active  7m45s
kube-system    Active  7m45s
```

验证Serverless Kubernetes集群

1. 执行以下命令，部署Nginx应用。

   ```
   kubectl run nginx --image=registry-vpc.cn-shenzhen.aliyuncs.com/acs-sample/nginx:latest
   ```

   预期输出：

   ```
   deployment.apps/nginx created
   ```

2. 执行以下命令，查询Nginx应用状态。

   ```
   kubectl get deploy
   ```

   预期输出：

   ```
   NAME  READY  UP-TO-DATE  AVAILABLE  AGE
   nginx  1/1   1       1       58s
   ```

删除资源

- 如果您需要删除Nginx应用，请执行以下命令。

  ```
  kubectl delete deploy nginx
  ```

  预期输出：

  ```
  deployment.extensions "nginx" deleted
  ```

- 您可以执行以下命令删除包含Serverless Kubernetes集群和相关的VPC等资源。

  ```
  aliyun cs DELETE /clusters/************
  ```

相关文档

- 概述
- 什么是阿里云CLI?
- 什么是云命令行?

6.3. 删除集群

您可以通过容器服务管理控制台删除不再使用的集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，在目标集群右侧**操作**列下选择**更多 > 删除**。

4. 如果目标集群已开启删除保护，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

   i. 在集群信息管理页面，单击**基本信息**页签。

   ii. 关闭**删除保护**开关。

   iii. 返回**集群列表**页面。

   iv. 在**集群列表**页面中，在目标集群右侧**操作**列下选择**更多 > 删除**。

5. 在弹出的**删除集群**对话框中，阅读并选中**我已知晓以上信息并确认删除集群**，单击**确定**。

> **注意** 如果您的集群挂载了ECI Pod，您需要先在**无状态**页面删除ECI Pod，然后再通过以上方法删除集群。

6.4. 管理和访问集群

6.4.1. 通过kubectl连接Kubernetes集群

如果您需要从客户端计算机连接到Kubernetes集群，请使用Kubernetes命令行客户端kubectl。本文介绍如何通过kubectl连接Kubernetes集群。

操作步骤

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

   安装和设置的详细信息，请参见安装和设置kubectl。

3. 配置集群凭据。

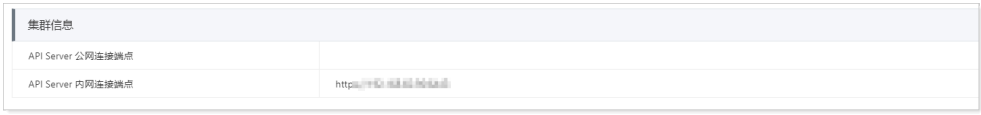
   您可以在集群信息页面查看集群凭据。

   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. （可选）选择**基本信息**页签，您可以在**集群信息**处查看集群的连接地址。



v. 选择**连接信息**页签，复制集群凭据到本地文件中，您可创建并将集群凭据保存到*$HOME/.kube/config*（kubectl预期凭据所在的位置）。或者命名一个新的文件，如*/tmp/kubeconfig*，并执行命令 `export KUBECONFIG=/tmp/kubeconfig` 。

vi. 执行上述操作后，您可执行以下命令，确认集群连接情况。

```
kubectl get pod
```

```
No resources found.
```

后续步骤

配置完成后，您可使用kubectl从本地计算机访问Kubernetes集群。

6.4.2. 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

ASK使用快速入门

背景信息

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

操作步骤

1. 登录容器服务管理控制台。
2. 在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.

6.4.3. 通过公网访问集群API Server

您可以使用弹性公网IP EIP（Elastic IP Address）暴露ASK集群的API Server。使用EIP暴露集群API Server后，您将获得从公网访问集群API Server的能力。本文介绍如何在创建集群时绑定EIP。

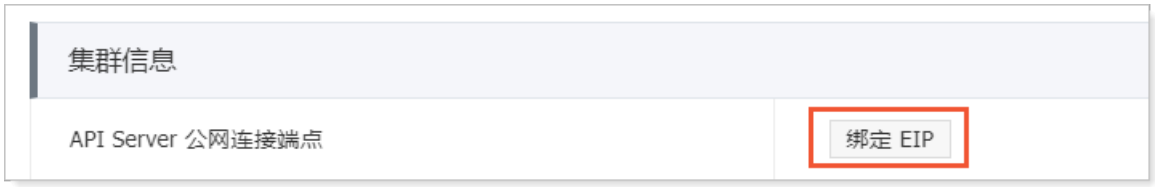
创建集群时绑定EIP

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ASK使用快速入门。

后续绑定EIP

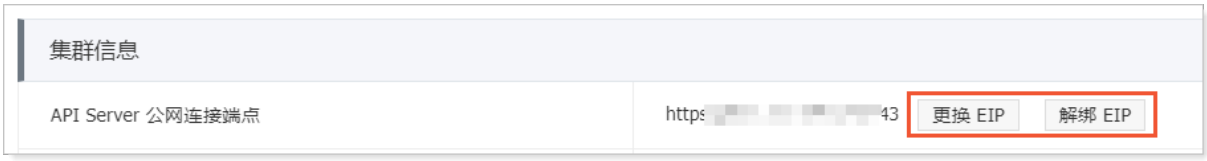
如果您创建集群时未选择为API Server开放公网，您可以通过以下步骤为集群关联EIP。

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. 在控制台左侧导航栏中，选择**集群 > 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**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**管理**。
6. 在**基本信息**页签的**集群信息**区域，单击**绑定EIP**。



后续步骤

绑定EIP后，您还可以参见上述步骤更换EIP或者解绑EIP。



6.5. 集群管理最佳实践

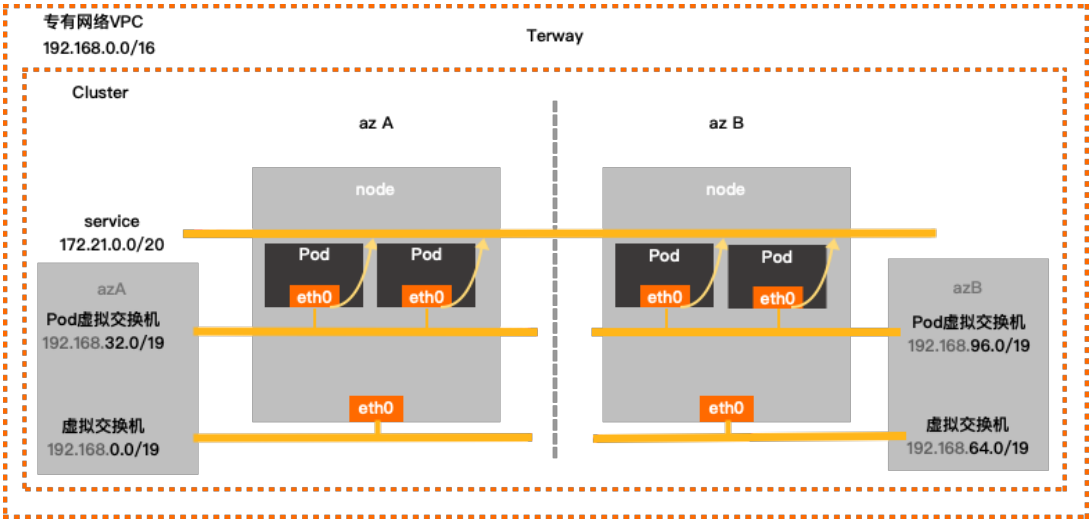
6.5.1. Kubernetes集群网络规划

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。

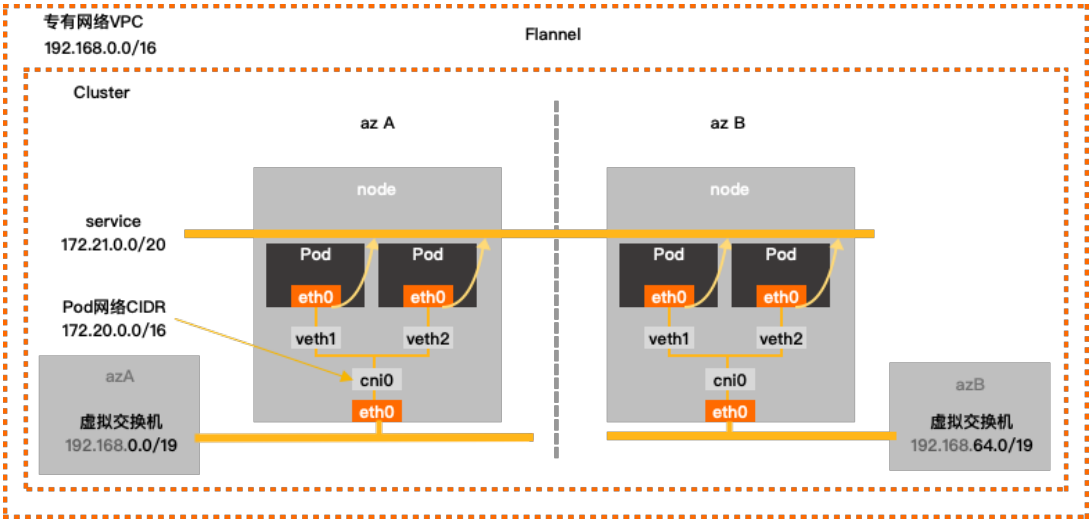
专有网络VPC网段和Kubernetes网段关系

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两种模式，两种模式的网络关系如下图所示。

Terway模式



Flannel模式



配置Terway网络模式和Flannel网络模式时，需要设置相关参数的网段，其注意事项如下。

配置参数	Terway网络模式	Flannel网络模式
专有网络	您在创建VPC时需要选择网段，只能从10.0.0.0/8、172.16.0.0/12、192.168.0.0/16三者当中选择一个。	
虚拟交换机	ECS使用的交换机，用于节点间网络通信。在VPC里创建交换机时指定的网段，必须是当前VPC网段的子集（可以和VPC网段一样，但不能超过）。配置网段时，请注意： • 虚拟交换机是VPC交换机。 • 交换机下ECS所分配到的地址，就是从这个交换机网段内获取的。 • 一个VPC下，可以创建多个交换机，但交换机网段不能重叠。 • 虚拟交换机和Pod虚拟交换机需要在一个可用区下。关于可用区的概念，请参见地域和可用区。	ECS使用的交换机，用于节点间网络通信。在VPC里创建交换机时指定的网段，必须是当前VPC网段的子集（可以和VPC网段一样，但不能超过）。配置网段时，请注意： • 虚拟交换机是VPC交换机。 • 交换机下ECS所分配到的地址，就是从这个交换机网段内获取的。 • 一个VPC下，可以创建多个交换机，但交换机网段不能重叠。

配置参数	Terway网络模式	Flannel网络模式
Pod虚拟交换机	Pod地址从该交换机分配，用于Pod网络通信。Pod是Kubernetes内的概念，每个Pod具有一个IP地址。在VPC里创建交换机时指定的网段，必须是当前VPC网段的子集。配置网段时，请注意： • Pod虚拟交换机是VPC交换机。 • Terway网络模式下，Pod分配的Pod IP 就是从这个交换机网段内获取的。 • 该地址段不能和**虚拟交换机**网段重叠。 • 该地址段不能和**Service CIDR**网段重叠。 • 虚拟交换机和Pod虚拟交换机需要在一个可用区下。关于可用区的概念，请参见地域和可用区。	选择Flannel网络模式时，无需设置此参数。
Pod网络CIDR	选择Terway网络模式时，无需设置此参数。	Pod网络CIDR，Pod地址从该地址段分配，用于Pod网络通信。Pod是Kubernetes内的概念，每个Pod具有一个IP地址。配置网段时，请注意： • 非VPC交换机，为虚拟网段。 • 该地址段不能和**虚拟交换机**网段重叠。 • 该地址段不能和**Service CIDR**网段重叠。 例如，VPC网段用的是172.16.0.0/12，Kubernetes的Pod地址段就不能使用172.16.0.0/16、172.17.0.0/16等，因为这些地址都包含在172.16.0.0/12里。
Service CIDR	Service地址段。Service是Kubernetes内的概念，对应的是Service类型为ClusterIP（`Type=ClusterIP`）时Service使用的地址，每个Service有自己的地址。配置网段时，请注意： • Service地址只在Kubernetes集群内使用，不能在集群外使用。 • Service地址段不能和**虚拟交换机**地址段重叠。 • Service地址段不能和**Pod虚拟交换机**地址段重叠。	Service地址段。Service是Kubernetes内的概念，对应的是Service类型为ClusterIP（`Type=ClusterIP`）时Service使用的地址，每个Service有自己的地址。配置网段时，请注意： • Service地址只在Kubernetes集群内使用，不能在集群外使用。 • Service地址段不能和**虚拟交换机**地址段重叠。 • Service地址段不能和**Pod网络CIDR**地址段重叠。

注意事项

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

集群节点规模	目的	VPC规划	可用区
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。

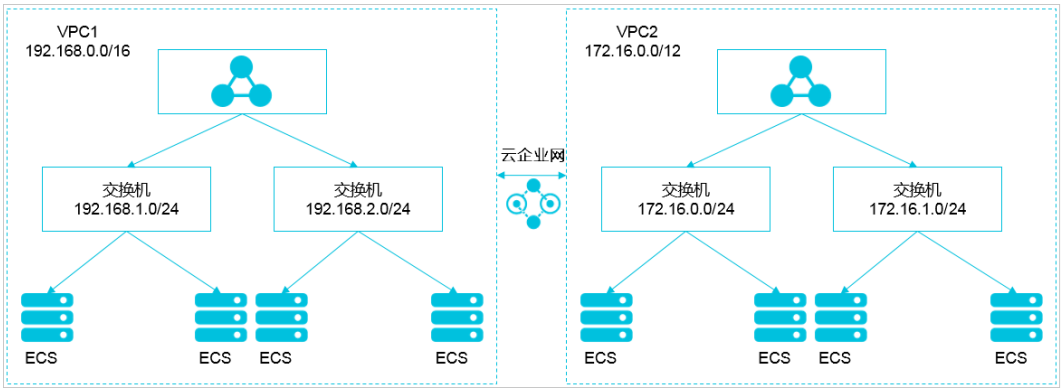
  - VPC地址是在创建VPC时已经确定。创建Kubernetes集群时，每个集群内的VPC地址段、Service地址段和Pod地址段彼此间不能重叠。
  - 所有Kubernetes集群之间的Pod地址段不能重叠，但Service地址段可以重叠。

- 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.ASK Pro集群

7.1. ASK Pro版集群概述

ASK Pro版集群是在ASK标准版基础上，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SLA的Serverless Kubernetes集群。

产品简介

ASK Pro版集群是在原ASK标准版集群的基础上发展而来的集群类型，继承了原标准版集群的所有优势，例如社区原生兼容、免运维、秒级弹性等。同时，相比原标准版进一步增强了集群的可靠性、安全性，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。
- 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%。

产品定价

ASK Pro版计费说明

集群对比

ASK Pro版集群和ASK标准版集群的对比详情如下表所示。

分类	功能	ASK	
		ASK Pro版集群	ASK标准版集群
集群规模	不涉及	最大10000 ECI实例	最大1000 ECI实例
SLA	不涉及	99.95%（支持赔付）	99.9%（不支持赔付）
API Server	自定义参数设置机制	✓	✗
	可用性监控	✓	✗

分类	功能	ASK	
		ASK Pro版集群	ASK标准版集群
ETCD	高频冷热备机制，异地容灾	✓	✗
	可观测性监控指标	✓	✗
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）	✓	✗

7.2. 使用阿里云KMS进行Secret的落盘加密

在ASK Pro版集群中，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KMS中管理的密钥对已创建的ASK Pro版集群中的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。

> **说明** 当前仅支持在已创建的ASK Pro版集群中开启Secret落盘加密，暂不支持在新建的ASK Pro版集群中开启。

在已创建的ASK Pro版集群中开启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ASK Pro版集群名称。
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RAM用户，请确保该RAM用户对该集群有RBAC的管理员或运维人员权限。具体操作，请参见配置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ASK Pro版集群中关闭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ASK Pro版集群名称。
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RAM用户，请确保该RAM用户对该集群有RBAC的管理员或运维人员权限。具体操作，请参见配置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。

8.镜像

8.1. 配置ACR企业版免密

使用阿里云镜像服务ACR来拉取镜像时，可以配置免密来简化配置，加速镜像拉取。本文将为您介绍如何配置免密拉取ACR企业版镜像。

前提条件

请确保您已完成以下操作：

- 已开通弹性容器实例服务、容器镜像服务ACR服务，并完成了相关的RAM角色授权。
- 已创建ASK集群。具体操作，请参见创建Serverless Kubernetes集群。
- 已创建ACR企业版实例并完成镜像仓库相关配置。具体操作，请参见使用企业版实例推送拉取镜像。

背景信息

阿里云容器镜像服务ACR分为默认版和企业版。其中，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 ，以及符合OCI规范制品的生命周期管理，能够与容器服务Kubernetes版无缝集成，适用于业务大规模部署场景，帮助企业降低交付复杂度。更多信息，请参见容器镜像服务ACR简介。

目前容器镜像服务ACR的镜像使用有如下几种情况：

- 与ECI同一账号下的ACR镜像默认免密，无需上传密码。
- 非ACR镜像（Docker镜像）不支持免密，使用OpenAPI创建ECI实例时，您可以通过ImageRegistryCredential参数来上传密码。

配置ACR企业版免密访问

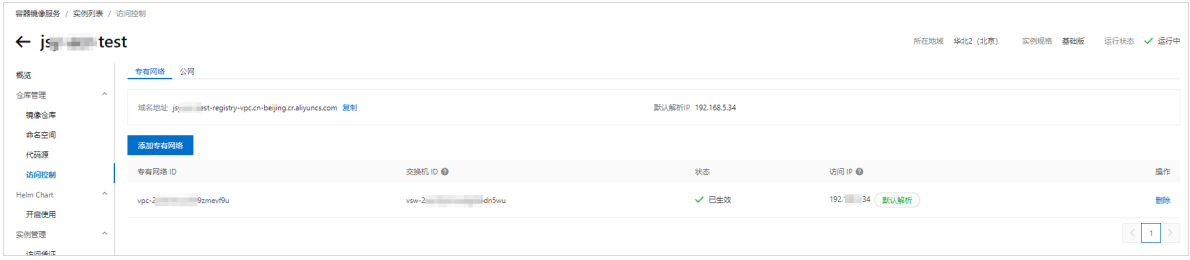
在容器镜像服务控制台打开要配置的ACR实例，配置网络访问控制如下：

- 公网访问

  开启公网访问入口后，可以直接通过公网域名地址来访问ACR企业版实例的镜像（可跨域）。具体操作，请参见配置公网的访问控制。

- 专有网络访问

  使用专有网络VPC访问，需要开启相关授权。具体操作，请参见配置专有网络的访问控制。



配置完成后，您可以记录实例ID、名称、域名等信息，以便后续配置使用。

使用ACR企业版免密拉取镜像（Kubernetes方式）

您可以通过添加Annotation来指定ACR实例，从对应的ACR实例中拉取镜像。

> **说明**
>
> Kubernetes方式仅支持指定单个ACR实例。如果您有多个ACR实例，且实例中存在多种不同的镜像，建议您将镜像直接归属到同一个ACR实例。如果您有配置多个ACR实例的需求，您可以使用OpenAPI方式。

示例如下：

1. 准备yaml文件。

   示例test_cri.yaml的内容如下：

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/acr-instance-id: cri-j36zhodptmyq****    #指定ACR企业版实例ID
 name: cri-test
spec:
 containers:
 - image: test****-registry.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0  #使用公网拉取镜像
  imagePullPolicy: Always
  name: nginx
 restartPolicy: Never
```

2. 创建Pod。

```
kubectl apply -f test_cri.yaml
```

使用ACR企业版免密拉取镜像（OpenAPI方式）

调用CreateContainerGroup接口创建ECI实例时，您可以设置AcrRegistryInfo相关参数来配置免密。相关参数说明如下表所示。更多信息，请参见CreateContainerGroup。

> **说明**
>
> 设置AcrRegistryInfo相关参数来配置免密时，必须设置AcrRegistryInfo.N.InstanceId。

名称	类型	示例值	描述
AcrRegistryInfo.N.RegionId	String	cn-beijing	ACR企业版实例所属地域。
AcrRegistryInfo.N.InstanceId	String	cri-nwj395hgf6f3****	ACR企业版实例ID。

名称	类型	示例值	描述
AcrRegistryInfo.N.Domain.N	RepeatList	test****-registry.cn-beijing.cr.aliyuncs.com	ACR企业版实例的域名。默认为相应实例的所有域名。支持指定个别域名，多个以半角逗号分隔。
AcrRegistryInfo.N.InstanceName	String	test****	ACR企业版实例的名称。

您可以使用以下几种方式传入AcrRegistryInfo相关参数进行配置：

- 示例一：指定地域、ACR实例ID、名称和域名。

```
'Container.1.Image': 'test****-registry.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0',
'Container.1.Name': 'c1',
'Container.2.Image': 'test****-registry-vpc.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0',
'Container.2.Name': 'c2',

#AcrRegistryInfo
'AcrRegistryInfo.1.RegionId':'cn-beijing',
'AcrRegistryInfo.1.InstanceId': 'cri-nwj395hg********',
'AcrRegistryInfo.1.Domain.1': 'test****-registry-vpc.cn-beijing.cr.aliyuncs.com',
'AcrRegistryInfo.1.Domain.2': 'test****-registry.cn-beijing.cr.aliyuncs.com'
```

- 示例二：指定ACR实例ID和名称

```
'Container.1.Image': 'test****-registry.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0',
'Container.1.Name': 'c1',
'Container.2.Image': 'test****-registry-vpc.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0',
'Container.2.Name': 'c2',

#AcrRegistryInfo
'AcrRegistryInfo.1.InstanceId': 'cri-nwj395hg********',
'AcrRegistryInfo.1.InstanceName': 'test****'
```

- 示例三：仅指定ACR实例ID

```
'Container.1.Image': 'test****-registry.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0',
'Container.1.Name': 'c1',
'Container.2.Image': 'test****-registry-vpc.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0',
'Container.2.Name': 'c2',

#AcrRegistryInfo
'AcrRegistryInfo.1.InstanceId': 'cri-nwj395hg********'
```

您也可以通过SDK方式进行配置。以下为Python示例：

```
#!/usr/bin/env python
#coding=utf-8

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.acs_exception.exceptions import ClientException
from aliyunsdkcore.acs_exception.exceptions import ServerException
from aliyunsdkeci.request.v20180808.CreateContainerGroupRequest import CreateContainerGroupReques
t

client = AcsClient('<accessKeyId>', '<accessSecret>', 'cn-beijing')

request = CreateContainerGroupRequest()
request.set_accept_format('json')

request.set_SecurityGroupId("sg-2zeh4cev9y7ulbr*****")
request.set_VSwitchId("vsw-2zejlv7xjnw61w6z*****")
request.set_ContainerGroupName("test-cri")
request.set_Containers([
 {
  "Image": "test****-registry.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0",
  "Name": "nginx"
 },
 {
  "Image": "test****-registry-vpc.cn-beijing.cr.aliyuncs.com/eci_test/nginx:1.0",
  "Name": "nginx2"
 }
])
request.set_AcrRegistryInfos([
 {
  "RegionId": "cn-beijing",
  "InstanceId": "cri-nwj395hgf6f*****",
  "Domains": [
   "test****-registry-vpc.cn-beijing.cr.aliyuncs.com",
   "test****-registry.cn-beijing.cr.aliyuncs.com"
  ]
 }
])

response = client.do_action_with_exception(request)
# python2:  print(response)
print(str(response, encoding='utf-8'))
```

8.2. 使用镜像缓存CRD加速创建Pod

阿里云以CRD的方式将ECI的镜像缓存功能提供给Kubernetes用户，以便Kubernetes用户也可以使用该功能来加速创建Pod。本文介绍如何使用镜像缓存CRD加速创建Pod。

背景信息

ECI实例在创建时，大部分时间消耗在镜像下载阶段。为加速实例创建速度，ECI提供镜像缓存功能。您可以预先将需要使用的镜像制作成云盘快照缓存，然后基于该快照来创建实例，避免或者减少镜像层的下载，从而提升实例创建速度。

经实测，以Docker Hub的flink镜像（约386.26 MB）创建Pod为例，正常创建ECI实例过程中，镜像准备阶段需要耗时50s，使用镜像缓存后，镜像准备阶段可以缩短至5s，可以极大节约实例创建耗时。

> **说明**
>
> 具体提升速度由Pod中使用的镜像个数、镜像大小和镜像仓库网络因素等决定。

为方便Kubernetes用户也可以使用到ECI的镜像缓存功能，阿里云以CRD的方式将镜像缓存功能提供给Kubernetes用户，对应的CRD为ImageCache。

> **注意**
>
> ImageCache CRD在kubernetes集群中为Cluster级别资源，被集群内所有Namespace共享。

准备工作

登录kubernetes集群，执行以下命令，验证您的kubernetes集群是否已经支持ImageCache。

```
kubectl get crd/imagecaches.eci.alibabacloud.com
```

- 如果返回Error信息，则表示您的kubernetes集群不支持ImageCache，返回示例如下：

  ```
  Error from server (NotFound): customresourcedefinitions.apiextensions.k8s.io "imagecaches.eci.alibabacloud.com" not found
  ```

  如果您的kubernetes集群不支持ImageCache，您需要升级Virtual Kubelet（VK）到相应支持ImageCache的版本。

  > **说明**
  >
  > 建议您升级VK到最新版本，以便更好地使用新功能。关于VK的版本信息，请参见Virtual Kubelet版本记录。

- 如果返回imagecaches.eci.alibabacloud.com的信息，则表示您的kubernetes集群已经支持ImageCache，返回示例如下：

  ```
  NAME                                CREATED AT
  imagecaches.eci.alibabacloud.com    2019-09-27T01:15:07Z
  ```

  如果您的kubernetes集群的ImageCache CRD不是最新的，建议您执行kubectl apply命令重新部署。

  ```
  kubectl apply -f imagecache-crd-sample.yaml
  ```

  imagecache-crd-sample.yaml的内容示例如下：

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 name: imagecaches.eci.alibabacloud.com
spec:
 group: eci.alibabacloud.com
 version: v1
 names:
  kind: ImageCache
  plural: imagecaches
  shortNames:
  - ic
  categories:
  - all
 scope: Cluster
 subresources:
  status: {}
 validation:
  openAPIV3Schema:
   required:
   - spec
   properties:
    spec:
     type: object
     required:
     - images
     properties:
      imagePullSecrets:
       type: array
       items:
        type: string
      images:
       minItems: 1
       type: array
       items:
        type: string
      imageCacheSize:
       type: integer
      retentionDays:
       type: integer
 additionalPrinterColumns:
 - name: Age
  type: date
  JSONPath: .metadata.creationTimestamp
 - name: CacheId
  type: string
  JSONPath: .status.imageCacheId
 - name: Phase
  type: string
  JSONPath: .status.phase
 - name: Progress
  type: string
  JSONPath: .status.progress
```

管理ImageCache

创建CustomResourceDefinition对象后，您可以操作ImageCache资源。更多信息，请参见kubernetes ImageCache API。

ImageCache的YAML配置文件如下：

```
apiVersion: eci.alibabacloud.com/v1
kind: ImageCache
metadata:
 name: imagecache-sample
 annotations:
  k8s.aliyun.com/imc-enable-reuse: "true" #开启镜像缓存复用
spec:
 images:
 - centos:latest
 - busybox:latest
 imagePullSecrets:
 - default:secret1
 - default:secret2
 - kube-system:secret3
 imageCacheSize:
  25
 retentionDays:
  7
```

> **说明**
>
> 开启镜像缓存复用后，在创建镜像缓存的过程中，系统将自动匹配已有镜像缓存，如果新的镜像缓存与已有镜像缓存存在重复的镜像层，新的镜像缓存将复用已有镜像缓存的镜像层，从而加速镜像缓存的创建。

相关参数说明如下：

名称	类型	是否必选	描述
spec.images	String[]	是	用于创建镜像缓存的容器镜像列表。
spec.imagePullSecrets	String[]	否	镜像仓库对应的Secret列表。 • 如果镜像列表中包含私有仓库的镜像，则需要为私有镜像仓库创建Secret，然后按照 `namespace:secretName` 的格式设置该参数。 • 如果镜像列表中的镜像均为公有镜像，则无需设置该参数。

名称	类型	是否必选	描述
spec.imageCacheSize	int	否	镜像缓存的大小，即用于创建镜像缓存快照的云盘大小。默认为20 GB。取值范围为20~32768 GB。
spec.retentionDays	int	否	镜像缓存保留时间，过期将会被清理。单位为天。默认永不过期。

创建ImageCache

1. 创建ImageCache。

   ```
   kubectl create -f imagecache-secrets-test.yaml
   ```

   imagecache-secrets-test.yaml的内容示例如下：

   ```
   apiVersion: eci.alibabacloud.com/v1
   kind: ImageCache
   metadata:
    name: imagecache-sample
    annotations:
     k8s.aliyun.com/imc-enable-reuse: "true" #开启镜像缓存复用
   spec:
    images:
    - centos:latest
    - busybox:latest
    imagePullSecrets:
    - default:secret1
    - default:secret2
    - kube-system:secret3
    imageCacheSize:
     25
    retentionDays:
     7
   ```

2. 查看ImageCache状态。

   ```
   kubectl get imagecache imagecache-sample-test
   ```

   返回结果示例如下：

   ```
   NAME                  AGE  CACHEID                 PHASE  PROGRESS
   imagecache-sample-test      20h  imc-2zeditzeoemfhqor****       Ready   100%
   ```

查询ImageCache

您可以根据需要查询集群下所有ImageCache列表或者查看某个ImageCache的详细信息：

- 查询集群下所有ImageCache列表

  ```
  kubectl get imagecache
  ```

- 查看某个ImageCache的详细信息

```
kubectl get imagecache/imagecache-sample-test -o yaml
```

删除ImageCache

如果想要删除某个ImageCache，可执行如下命令：

```
kubectl delete imagecache/imagecache-sample-test
```

使用ImageCache加速创建Pod

ImageCache资源是Cluster级别，因此在不同的namespace下创建Pod时均可以使用ImageCache来实现加速创建Pod。

使用ImageCache创建Pod包括明确指定和自动匹配两种方式，您可以在Pod级别的metadata中添加Annotation来配置，相关配置项如下：

- k8s.aliyun.com/eci-image-snapshot-id：明确指定使用哪个镜像缓存创建Pod。
- k8s.aliyun.com/eci-image-cache：根据匹配策略自动匹配最优的ImageCache创建Pod。

> **说明**
>
> 如果同时设置 `k8s.aliyun.com/eci-image-snapshot-id` 和 `k8s.aliyun.com/eci-image-cache` ，则明确指定方式的优先级高于自动匹配方式。

使用ImageCache创建Pod时，请注意以下事项：

- Pod中容器的镜像建议采用ImageCache中的镜像，以提高匹配度。
- Pod中容器的镜像拉取策略（ImagePullPolicy）建议设置为按需拉取（IfNotPresent），以避免镜像层重复下载。

明确指定

创建Pod时，您可以通过添加Annotation的方式，声明使用指定的ImageCache来加速创建Pod。

> **注意**
>
> 请确保指定的ImageCache的状态为Ready，其它状态的ImageCache会导致Pod创建失败。

- Deployment示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-image-snapshot-id: imc-2ze5tm5gehgtiiga**** #明确指定ImageCache
  spec:
   nodeName: virtual-kubelet
   containers:
   - name: nginx
    image: nginx:1.7.9
    imagePullPolicy: IfNotPresent
```

- Pod示例

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-image-snapshot-id: imc-2ze5tm5gehgtiiga**** #明确指定ImageCache
 name: nginx-imagecache-id
spec:
 containers:
 - image: nginx:1.7.9
  imagePullPolicy: IfNotPresent
  name: nginx
  resources:
   limits:
    cpu: 300m
    memory: 200Mi
   requests:
    cpu: 200m
    memory: 100Mi
 nodeName: virtual-kubelet
```

自动匹配

创建Pod时，您可以通过添加Annotation的方式，声明使用自动匹配的ImageCache来加速创建Pod。ECI将基于您已有的ImageCache列表，根据匹配策略进行匹配，选择最优的ImageCache来创建Pod。匹配策略的优先级从高到低依次为：镜像匹配度、匹配的镜像大小、创建时间。

> **说明**
>
> 如果没有匹配到合适的ImageCache，则将执行正常下载镜像的流程来创建Pod。

- Deployment示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
   annotations:
    k8s.aliyun.com/eci-image-cache: "true"  #开启自动匹配ImageCache
  spec:
   nodeName: virtual-kubelet
   containers:
   - name: nginx
    image: nginx:1.7.9
    imagePullPolicy: IfNotPresent
```

- Pod示例

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
  k8s.aliyun.com/eci-image-cache: "true"  #开启自动匹配ImageCache
 name: nginx-auto-match
spec:
 containers:
 - image: nginx:1.7.9
  imagePullPolicy: IfNotPresent
  name: nginx
  resources:
   limits:
    cpu: 300m
    memory: 200Mi
   requests:
    cpu: 200m
    memory: 100Mi
 nodeName: virtual-kubelet
```

9.应用

9.1. 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。本文介绍如何通过命令管理应用。

前提条件

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

通过命令创建应用

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

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

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

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

通过命令查看容器

运行如下命令列出所有default命名空间里正在运行的容器。

```
kubectl get pods
```

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

9.2. 使用镜像创建应用

本文为您介绍如何在控制台上使用镜像的方式创建应用。

前提条件

已创建Serverless Kubernetes集群。具体操作，请参见创建Serverless Kubernetes集群。

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**页面，设置应用的基本信息。

配置项	描述
应用名称	设置应用的名称。
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**等。
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）。

7. 单击**下一步**。进入**容器配置**页面。

步骤二：配置容器

在**容器配置**页签中，配置容器的镜像与资源、端口、环境变量、健康检查、生命周期、数据卷等。

> **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。

1. 在**基本配置**区域，完成容器的基本配置。

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**。 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。
	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。而选中该选项后，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：单击**设置镜像密钥**设置镜像的密钥。对于私有仓库访问时，需要设置密钥。具体操作，请参见使用镜像密钥创建应用。
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。

配置项	描述
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。更多信息，请参见Init Containers。

2. 在**端口设置**区域，单击**新增**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。
3. 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。更多信息，请参见Pod variable。
   - 类型：设置环境变量的类型，支持**自定义**、**配置项**等类型。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。
4. 在**健康检查**区域，根据需要开启**存活检查**和**就绪检查**。

   关于健康检查的更多信息，请参见Configure Liveness, Readiness and Startup Probes。
   - 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。

请求类型	配置说明

请求类型	配置说明
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。

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. 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。

   - **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. 在**数据卷**区域，增加本地存储或云存储声明PVC（Persistent Volume Claim）等。

   支持增加以下存储卷：

   - 本地存储
   - 云存储声明（PersistentVolumeClaim）
   - NAS
   - 云盘

7. 单击**下一步**，进行高级设置。

步骤三：完成高级配置

在**高级配置**页签中设置访问、伸缩和标签注解。

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**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。

配置项	描述
类型	选择服务类型，即服务访问的方式。本例中选择**虚拟集群IP**。 ■ **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）。 ■ **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。
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。

配置项	描述
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**。
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。
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。

  详细的路由配置信息，请参见路由配置说明。

  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，请参见路由配置说明。 ■ **域名**：输入Ingress域名。本例中使用测试域名 foo.bar.com 。 ■ **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。本例中为**nginx-svc**。 ■ **开启TLS**：配置安全的路由服务。更多信息，请参见Ingress高级用法。
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。
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. 在**伸缩配置**区域，勾选是否开启**容器组水平伸缩**，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弹性伸缩，即根据容器CPU和内存资源占用情况自动调整容器组数量。

   > **说明** 若要启用自动伸缩，您必须为容器设置所需资源，否则容器自动伸缩无法生效。

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。

3. 在**标签和注释**区域，单击**添加**设置容器组的标签和注释。
   - 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）。
4. 单击**创建**。

步骤四：查看应用

在**创建完成**页签中查看应用任务。

1. 在**创建完成**页签中单击**查看应用详情**。您可以看到新建的serverless-app-svc出现在无状态列表下。

2. 在集群管理页左侧导航栏中，选择**网络 > 服务**，可以看到新建的服务serverless-app-svc出现在服务列表下。

3. 在浏览器中访问外部端点，您可访问Nginx欢迎页，本示例为Nginx。

9.3. 创建服务

本文档为您介绍如何在Serverless Kubernetes集群中创建服务。

前提条件

已创建Serverless Kubernetes集群。具体操作，请参见创建Serverless Kubernetes集群。

背景信息

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

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

更多详细的原理，请参见Kubernetes service。

步骤一：创建Deployment

使用镜像创建一个Deployment，本例中创建的Deployment名称为serverless-app-deployment。具体操作请参见使用镜像创建应用。

步骤二：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. 在弹出的创建服务对话框中，进行配置。
   - **名称**：输入服务的名称，本例中为nginx-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）**。

     - 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。

       > **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对象，本例中是前面创建的serverless-app-deployment。若不进行关

联部署，则不会创建相关的Endpoints对象，您可自己进行绑定，参见services-without-selectors。

- **外部流量策略**：可选值为Local或Cluster。

  > **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**：添加服务端口和容器端口，容器端口需要与后端的pod中暴露的容器端口一致。
- **注解**：为该服务添加一个注解（annotation），配置负载均衡的参数，例如设置 service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth:20 表示将该服务的带宽峰值设置为20Mbit/s，从而控制服务的流量。更多参数请参见通过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**，nginx-svc服务出现在服务列表中。

9.4. 删除服务

您可以通过容器服务控制台快速对服务进行删除。本文介绍如何在容器服务控制台删除服务。

前提条件

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

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**网络 > 服务**。
5. 选择目标服务，单击右侧**操作**列下的**删除**。
6. 在弹出的**提示**对话框中，单击**确定**，确认删除，该服务在服务列表中消失。

9.5. 查看容器

您可以通过容器服务管理控制台的容器组页面查看阿里云Serverless Kubernetes集群的容器组。本文介绍在容器服务管理控制台如何查看ASK集群的容器。

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. 选择目标容器组，单击右侧的**详情**。

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

6. 进入容器组的详情页，您可查看该容器组的详情信息。

9.6. 查看服务

您在创建应用时，如果配置了外部服务，除了运行容器，Kubernetes Dashboard还会创建外部Service，用于预配负载均衡器，以便将流量引入到集群中的容器。本文介绍如何通过容器服务控制台界面查看服务详情。

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. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 您可以在**服务（Service）**页面，查看部署的服务。

   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。

10.配置项及密钥

10.1. 创建配置项

在容器服务管理控制台上，您可以通过配置项菜单创建配置项。本文介绍如何在容器服务控制台上创建配置项。

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**配置管理 > 配置项**。
5.
6. 在**配置项**页面，单击**创建**。
7. 在**创建**面板中，输入配置项名称，然后单击**+添加**，填写配置项名称和值。
   - **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（-）或者点号（.），名称不能为空。其他资源对象需要引用配置文件名来获取配置信息。
   - **配置项**：填写配置项名称和配置项的值。您也可以单击面板右上角**从文件导入**创建配置项。
8. 单击**确定**。

执行结果

您可以在配置项列表中看到创建好的配置项。

11.存储-Flexvolume

11.1. 存储插件说明

目前阿里云容器服务ASK集群支持两种存储插件Flexvolume和CSI。本文介绍两种存储插件特性及如何选择合适的存储插件。

Flexvolume和CSI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。ASK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。ASK默认提供云盘、NAS两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。ASK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。ASK默认提供云盘、NAS两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。

使用推荐

- 针对新建集群，推荐您使用CSI插件。ASK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。ASK会持续支持Flexvolume插件。

使用须知

- 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。

11.2. 存储Flexvolume概述

容器服务ASK支持自动绑定阿里云云盘、阿里云文件存储NAS（Network Attached Storage）。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	支持通过PV/PVC方式使用云盘静态存储卷	支持

阿里云存储	静态数据卷	动态数据卷
阿里云NAS	NAS静态存储卷支持以下两种使用方式： • 通过Flexvolume插件以通过PV/PVC方式使用NAS静态存储卷 • 通过Kubernetes的NFS驱动使用	不支持

11.3. 安装与升级Flexvolume插件

对于阿里云Kubernetes 1.16之前版本的集群，若创建集群时存储插件选择为Flexvolume，则控制台默认安装Flexvolume与Disk Controller组件。本文介绍如何安装与升级Flexvolume组件和Disk Controller组件。

前提条件

- 已创建ASK集群。具体操作，请参见ASK使用快速入门。
- 阿里云Kubernetes集群存储插件为Flexvolume。
- 已通过kubectl工具连接Kubernetes集群。具体操作，请参见通过kubectl连接Kubernetes集群。

使用限制

- 目前支持CentOS 7、Aliyun Linux 2操作系统。
- ASK集群不支持部署alicloud-nas-controller组件。

安装Flexvolume组件与Disk Controller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Flexvolume和CSI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ASK使用快速入门。

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Flexvolume和CSI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ASK使用快速入门。

验证安装

- 执行以下命令，查看Flexvolume组件是否成功部署。

  ```
  kubectl get pod -nkube-system | grep flexvolume
  ```

  预期输出：

  ```
  NAME    READY  STATUS   RESTARTS  AGE
  flexvolume  1/1   Running  0     14d
  ```

  从预期输出可得，Pod的状态为Running，表示Flexvolume组件安装成功。

- 执行以下命令，查看Disk Controller组件是否成功部署。

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

预期输出：

```
NAME                          READY  STATUS   RESTARTS  AGE
alicloud-disk-controller  1/1   Running  0     14d
```

从预期输出可得，Pod的状态为Running，表示Disk Controller组件安装成功。

升级Flexvolume组件与Disk Controller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。

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。

- Flexvolume组件升级出现以下问题时，请提交工单申请手动升级保障。
  - 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，请提交工单申请手动升级保障。

11.4. 云盘存储卷

11.4.1. 云盘存储卷概述

您可以在ASK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、注意事项以及计费说明。

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。

使用说明

ASK集群支持挂载云盘静态存储卷及云盘动态存储卷。

- 关于如何挂载云盘静态存储卷，请参见使用云盘静态存储卷。
- 关于如何挂载云盘动态存储卷，请参见通过命令行使用动态云盘卷或通过控制台使用动态云盘卷。

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，请参见云服务器ECS产品详情页。

更多信息，请参见计费。

11.4.2. 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

操作步骤

1. 创建云盘类型的PV。

   您可以使用YAML文件或者控制台界面创建云盘类型的PV。

   - 通过YAML文件创建PV。

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
```

11.4.3. 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。

11.4.4. 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

已创建ASK集群。具体操作，请参见创建Serverless Kubernetes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 > 存储卷**。
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 > 存储卷**。
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：创建应用并配置存储声明

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。

11.5. NAS存储卷

11.5.1. 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等。

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

- 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。
- 在ASK集群中，只支持挂载NAS静态存储卷，不支持挂载NAS动态存储卷。关于如何挂载NAS静态存储卷，请参见使用NAS静态存储卷。

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` 。

计费说明

关于NAS的计费说明，请参见NAS计费说明。

11.5.2. 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。
- 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插件。
- 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。

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数据卷。

   - 通过YAML文件创建PV。

     使用*nas-pv.yaml*文件创建PV。

     ```
     apiVersion: v1
     kind: PersistentVolume
     metadata:
      name: pv-nas
     spec:
      capacity:
       storage: 5Gi
      storageClassName: nas
      accessModes:
       - ReadWriteMany
      flexVolume:
       driver: "alicloud/nas"
       options:
        server: "0cd8b4a576-u****.cn-hangzhou.nas.aliyuncs.com"
        path: "/k8s"
        vers: "3"
        options: "nolock,tcp,noresvport"
     ```

   - 通过控制台界面创建云盘数据卷。

     a. 登录容器服务管理控制台。

     b. 在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。

参数	描述
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。
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
```

11.6. 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

- 您已创建一个Serverless Kubernetes集群，请参见ASK使用快速入门。
- 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 > 存储卷**。
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。

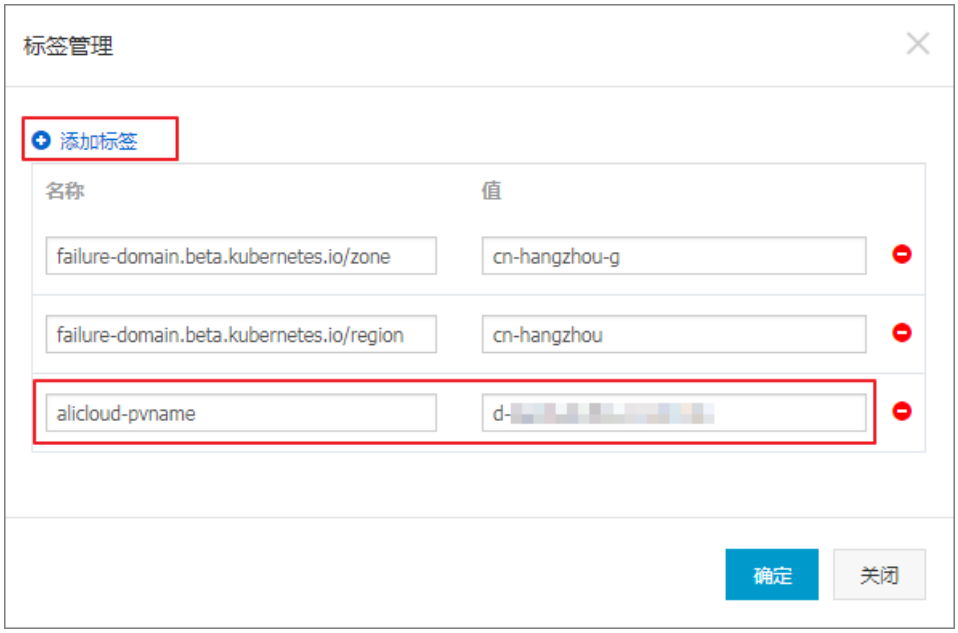
     > **说明** 仅分配模式选择**已有存储卷**，才需设置该参数。

   - **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。

11.7. 使用持久化存储卷声明

您可通过容器服务控制台，通过镜像或模板部署应用，从而使用持久化存储声明。本例中使用镜像来创建应用，若您想通过模板使用持久化存储卷声明，请参见云盘存储卷概述。

前提条件

- 您已创建一个Serverless Kubernetes集群，请参见ASK使用快速入门。
- 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。

11.8. 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.存储-CSI

12.1. 存储插件说明

目前阿里云容器服务ASK集群支持两种存储插件Flexvolume和CSI。本文介绍两种存储插件特性及如何选择合适的存储插件。

Flexvolume和CSI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。ASK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。ASK默认提供云盘、NAS两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。ASK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。ASK默认提供云盘、NAS两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。

> **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。ASK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。ASK会持续支持Flexvolume插件。

12.2. 存储CSI概述

容器服务ASK支持自动绑定阿里云云盘、阿里云文件存储NAS（Network Attached Storage）。本文介绍支持的存储服务和数据卷的情况。

目前CSI驱动支持静态存储卷和动态存储卷。每种数据卷的支持情况如下：

阿里云存储	静态数据卷	动态数据卷

阿里云存储	静态数据卷	动态数据卷
阿里云云盘	支持使用CSI驱动以PV/PVC方式挂载云盘静态存储卷。	支持
阿里云NAS	支持使用CSI驱动以PV/PVC方式挂载NAS静态存储卷。	不支持

版本支持

- 当Kubernetes版本为1.14及以上版本时，支持选择使用Flexvolume/CSI插件类型。
- 新创建集群选择为CSI驱动时，Kubelet参数 `enable-controller-attach-detach` 配置为 `true` 。

CSI驱动支持的地域同Kubernetes集群1.14所开放的地域。

12.3. 安装与升级CSI-Provisioner组件

CSI-Provisioner组件具备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。本文介绍在阿里云Serverless Kubernetes（ASK）中如何安装与升级CSI-Provisioner存储组件。

前提条件

- 已创建ASK集群。具体操作，请参见创建ASK集群。
- 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。

安装CSI-Provisioner

ASK集群需要手动安装CSI-Provisioner。如果您的集群中没有安装该组件，您可以手动安装。关于安装CSI-Provisioner的YAML示例，请参见alibaba-cloud-csi-driver。

验证安装

执行以下命令，验证CSI-Provisioner组件是否安装成功。

```
kubectl get pod -n kube-system | grep csi-provisioner
```

预期输出：

```
NAME      READY  STATUS   RESTARTS  AGE
csi-provisioner  1/1   Running  0     14d
```

从预期输出可得，Pod的状态为Running，表示CSI-Provisioner组件安装成功。

升级CSI-Provisioner

若需要升级CSI-Provisioner组件，请根据CSI-Provisioner组件的变更记录更新YAML文件中的镜像地址。关于CSI-Provisioner组件的版本信息，请参见csi-provisioner。

12.4. 云盘存储卷

12.4.1. 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。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

性能类别	ESSD云盘				SSD云盘	高效云盘	普通云盘
	PL3	PL2	PL1	PL0			
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。

使用限制

-
- 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

-
- 云盘的具体价格信息，请参见云服务器ECS产品详情页。

更多信息，请参见计费。

存储类（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"}}}'
```

12.4.2. 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**。

参数	描述
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。

参数	说明
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。
storage	声明应用使用量，不能大于存储卷的总量。

参数	说明
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。

12.4.3. 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ASK集群。具体操作，请参见创建Serverless Kubernetes集群。
- 已通过kubectl工具连接ASK集群。具体操作，请参见通过kubectl连接Kubernetes集群。

背景信息

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。
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*。

参数	说明
（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。
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。

12.4.4. 存储类（StorageClass）

本文为您介绍集群默认提供的StorageClass存储类型和适用场景，以及如何配置Default StorageClass存储类型。

StorageClass

ASK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：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。

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，ASK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
```

后续步骤

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"}}}'
```

12.5. NAS存储卷

12.5.1. NAS存储卷概述

您可以在ASK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

- 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。
- 在ASK集群中，只支持挂载NAS静态存储卷，不支持挂载NAS动态存储卷。关于如何挂载NAS静态存储卷，请参见使用NAS静态存储卷。

注意事项

- NAS为共享存储，可以同时为多个Pod提供共享存储服务，即一个PVC可以同时被多个Pod使用。
- 在没有卸载NAS文件系统前，务必不要删除NAS挂载点，否则会造成操作系统无响应。
- 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。

12.5.2. 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。
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。

参数	说明
driver	驱动类型。本例中取值为 nasplugin.csi.alibabacloud.com ，表示使用阿里云NAS CSI插件。
volumeHandle	配置PV的唯一标识符。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。

13.网络

13.1. Service管理

13.1.1. 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。

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。	

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。

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。

13.1.2. 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9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
```

- 创建HTTP类型的负载均衡

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

- 创建HTTPS类型的负载均衡

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

  > **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
```

- 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。

- 创建带健康检查的负载均衡

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

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

- 设置HTTP类型的健康检查

  设置HTTP类型的健康检查，以下所有的annotation必选。

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

- 为负载均衡设置调度算法

- 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
```

- 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=V
alue2"
 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
```

SLB常用注解

注解	类型	描述	默认值	支持的版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type	string	取值可以是*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	取值可以是*paybytraffic*或者*paybybandwidth*。	*paybytraffic*	v1.9.3及以上版本
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

注解	类型	描述	默认值	支持的版本
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-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-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

注解	类型	描述	默认值	支持的版本
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
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

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协议的负载均衡配置会话保持（insert cookie）
  - 仅支持HTTP及HTTPS协议的负载均衡实例。
  - 如果配置了多个HTTP或者HTTPS的监听端口，该会话保持默认应用到所有HTTP和HTTPS监听端口。
  - 配置insert cookie，以下四项annotation必选。

```
apiVersion: v1
kind: Service
metadata:
 annotations:
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session: "on"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session-type: "insert"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie-timeout: "1800"
  service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-port: "http:80"
 name: nginx
 namespace: default
spec:
 ports:
 - port: 80
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 type: LoadBalancer
```

- 为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
```

监听的常用注解

注解	类型	描述	默认值	支持的版本
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
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

注解	类型	描述	默认值	支持的版本
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

注解	类型	描述	默认值	支持的版本
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-flag	string	取值是*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
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

注解	类型	描述	默认值	支持的版本
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
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

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/zone=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：&quot;eni&quot;`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：&quot;eni&quot;`中的`eni`设置为`ecs`将ECS挂载到SLB后端。

后端服务器组的常用注解

注解	类型	描述	默认值	支持的版本
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

注解	类型	描述	默认值	支持的版本
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
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

13.1.3. 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。

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。

  更多SLB使用限制请参见使用限制。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：

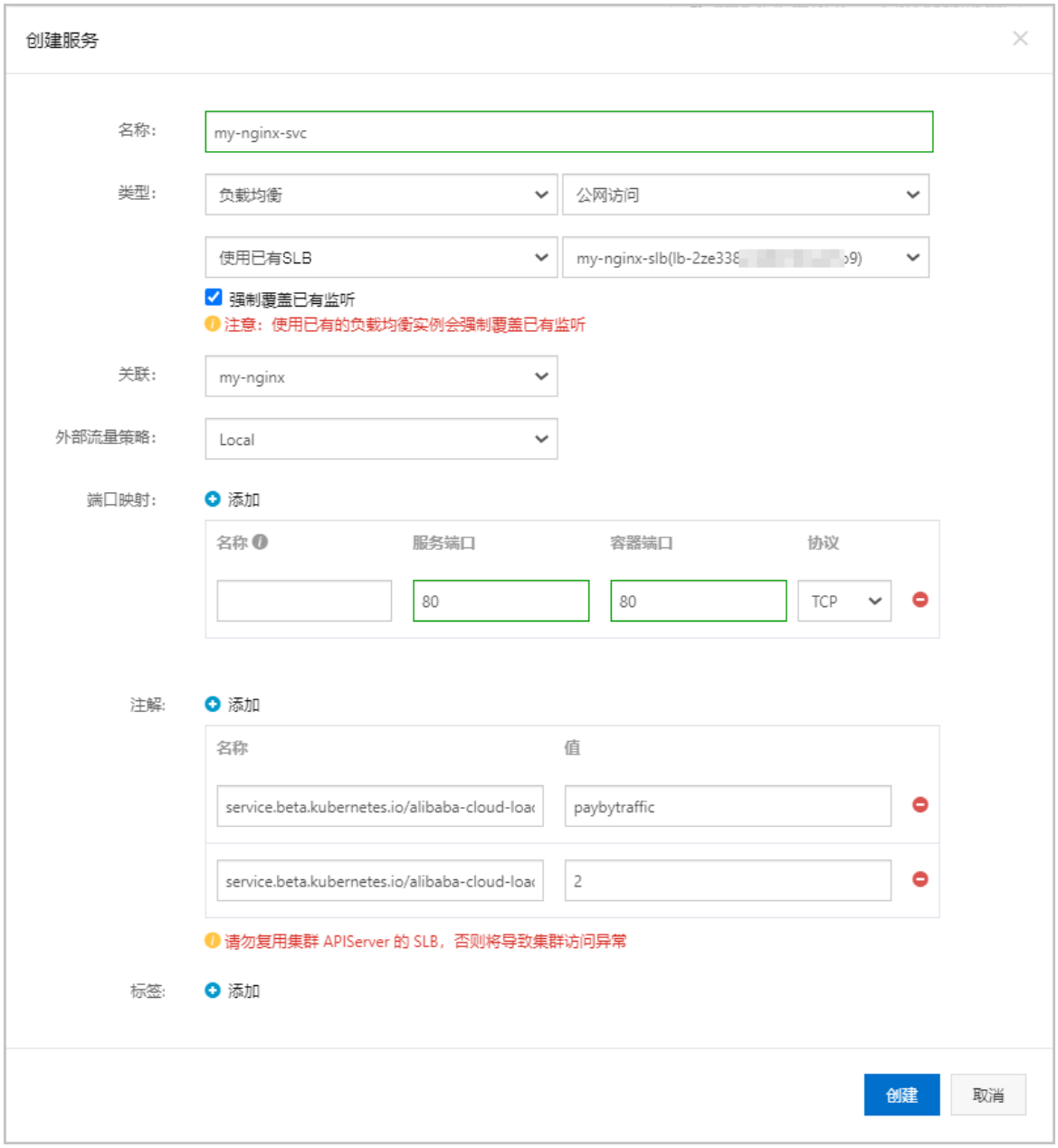
```
NAME    READY  UP-TO-DATE  AVAILABLE  AGE
my-nginx  3/3   3       3       50s
```

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

13.2. ALB Ingress管理

13.2.1. ALB Ingress概述

本文介绍Ingress基本概念、ALB Ingress Controller工作原理和ALB Ingress Controller使用说明。

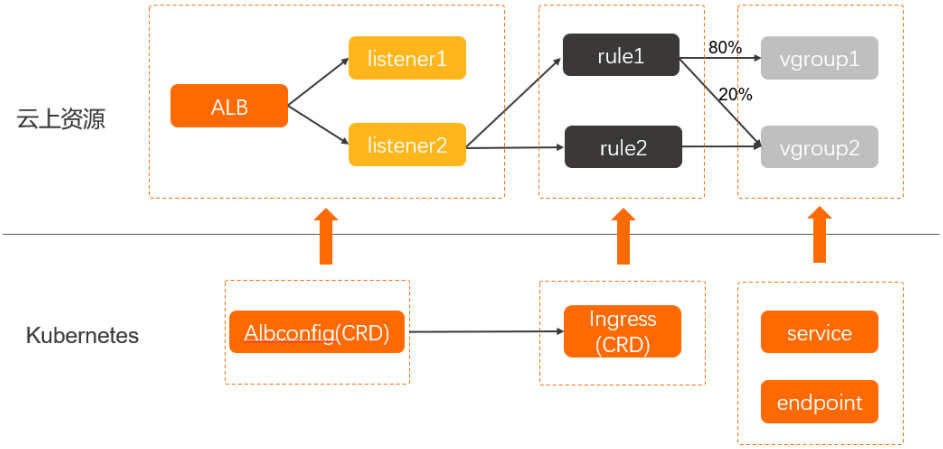
Ingress基本概念

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。

ALB Ingress Controller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。



ALB Ingress Controller使用说明

> **注意** 为Ingress服务的ALB是由Controller完全托管的，您不能自行在ALB控制台上进行配置，否则可能造成Ingress服务的异常。

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。

相关文档

- 通过ALB Ingress访问服务
- ALB Ingress Controller组件管理

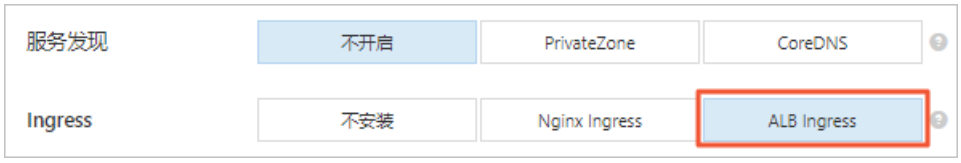
13.2.2. ALB Ingress Controller组件管理

阿里云Serverless Kubernetes集群，基于ALB七层转发规则提供了托管的ALB Ingress Controller。本文介绍如何在ASK集群安装和卸载ALB Ingress Controller。

安装ALB Ingress Controller

方式一：创建集群时安装ALB Ingress Controller

创建ASK集群时，在**Ingress**参数配置区域，选择安装**ALB Ingress**。具体操作，请参见ASK使用快速入门。



方式二：在组件管理页面安装ALB Ingress Controller

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**ALB Ingress Controller**组件区域单击**安装**。

卸载ALB Ingress Controller

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ALB Ingress Controller组件区域单击**卸载**。

13.2.3. 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）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。本文介绍如何使用ALB Ingress访问服务。

前提条件

- 您已创建一个ASK集群，集群的VPC需要配置NAT网关，从而可以访问外网，下载容器镜像。具体操作，请参见ASK使用快速入门。
- 您已通过kubectl连接到集群。具体操作，请参见通过kubectl连接Kubernetes集群。

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。

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
 clusterIP: None
---
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
 clusterIP: None
```

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
svc/coffee-svc  ClusterIP  <none>    <none>    80/TCP   1m
svc/tea-svc    ClusterIP  <none>    <none>    80/TCP   1m
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。

参数	说明
（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

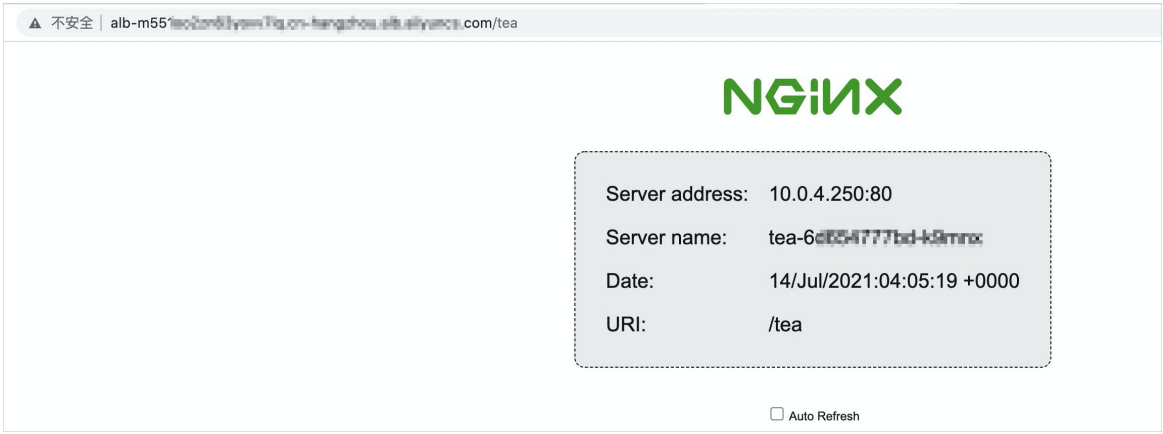
- 利用获取的ALB实例IP地址，通过以下两种方式访问 `coffee` 服务：
  - 通过浏览器访问 `coffee` 服务。

  - 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
```

13.2.4. ALB Ingress服务高级用法

在ASK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

- 您已创建一个ASK集群，集群的VPC需要配置NAT网关，从而可以访问外网，下载容器镜像。具体操作，请参见ASK使用快速入门。
- 已通过Kubectl工具连接集群。具体操作，请参见通过kubectl连接Kubernetes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: ClusterIP
```

```
---
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  #定义Ingress的域名。
    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 a
lbtop-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: ClusterIP
---
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:
 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: ClusterIP
---
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-k1akdsmts6njkvhasriop,vsw-k1amdv9ax94gr5iwamuwu"
    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

`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/alb config.order: "1"`
		`alb.ingress.kubernetes.io/vs witch-ids: "vsw-k1akdsmts6njkvhas****,vsw-k1amdv9ax94gr5iwa****"`
		`alb.ingress.kubernetes.io/ca nary: "true"`
		`alb.ingress.kubernetes.io/ca nary-by-header: "location"`
		`alb.ingress.kubernetes.io/ca nary-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"

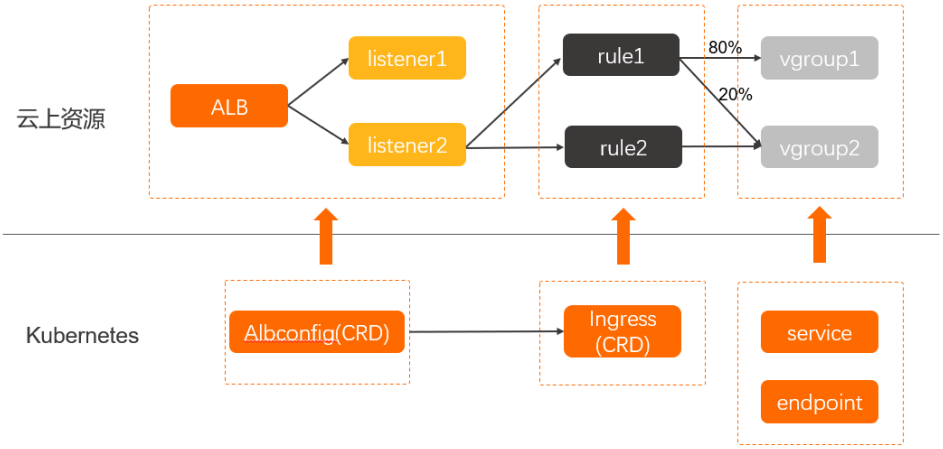
13.2.5. 配置Albconfig

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****
...
```

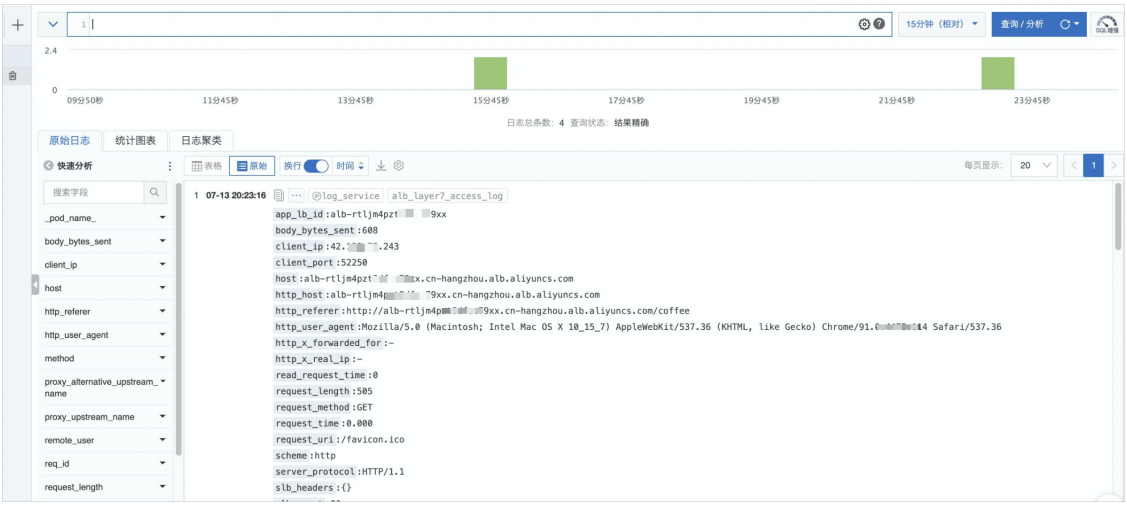
开启日志服务访问日志

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。

13.3. SLB Ingress管理

13.3.1. SLB Ingress概述

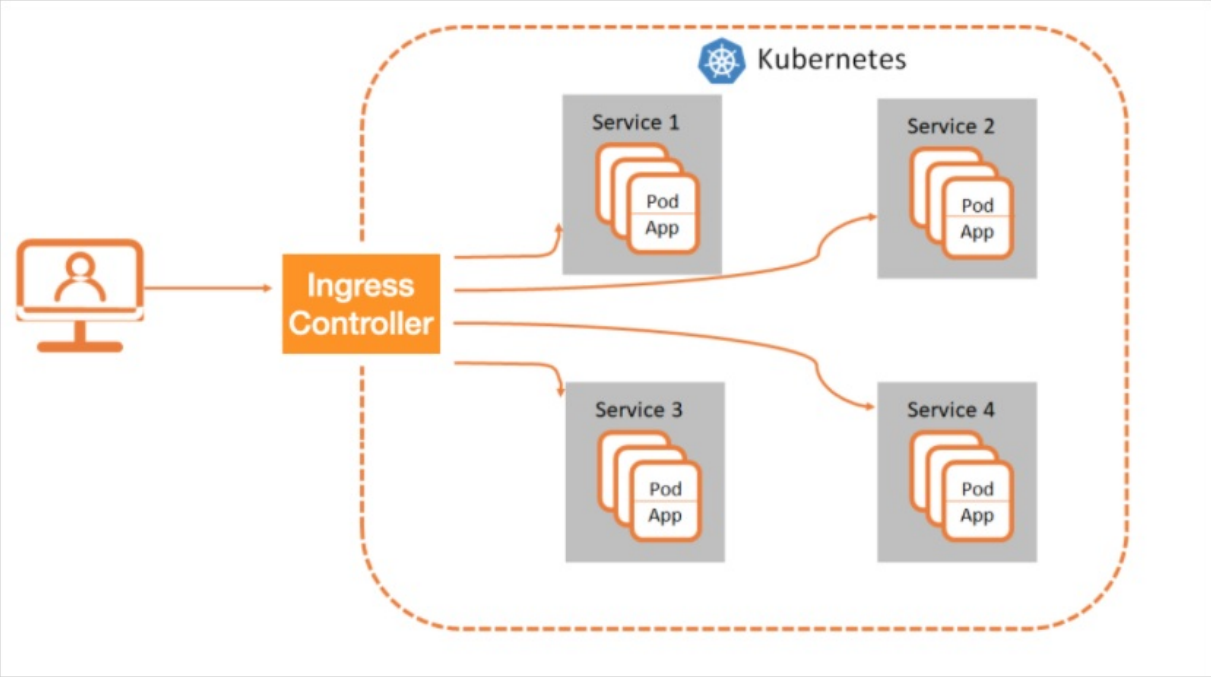
本文介绍Ingress基本概念、Ingress Controller工作原理和SLB Ingress Controller使用说明。

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 Controller工作原理

Ingress Controller通过API Server获取Ingress资源的变化，动态地生成Load Balancer所需的配置文件，然后依次生成新的路由转发规则。



SLB Ingress Controller使用说明

> **注意** 为Ingress服务的SLB是由Controller完全托管的，您不能自己在SLB控制台上面进行配置，否则可能造成Ingress服务的异常。

阿里云Serverless Kubernetes集群，基于SLB七层转发规则提供了托管的SLB Ingress Controller，您无需承担额外的IaaS成本，云产品级别的容灾能力给您提供稳定的Ingress服务。

- 在不指定SLB实例情况下，ASK会自动生成一个公网SLB实例。ASK不会自动生成私网SLB实例。
- 公网SLB和私网SLB都只能通过Annotation指定SLB ID。
  - 如果需要使用公网SLB提供服务，您可以使用已有的公网SLB实例或创建一个公网SLB实例，然后通过Annotation指定SLB ID。
  - 如果需要使用私网SLB提供服务，您可以使用已有的私网SLB实例或创建一个私网SLB实例，然后通过Annotation指定SLB ID。
- 当您指定使用已存在的SLB实例时，要求该SLB实例规格必须是性能保障型（支持ENI），同时确保80和443端口当前没有其他服务使用。
- SLB实例前端监听会把80端口设置成HTTP协议、把443端口设置成 HTTPS协议。
- SLB默认会使用第一个（按时间正序排序）Ingress指定的secret证书作为HTTPS监听证书，如果所有Ingress都没有指定secret，则使用fake的默认证书。

相关文档

- Nginx Ingress概述

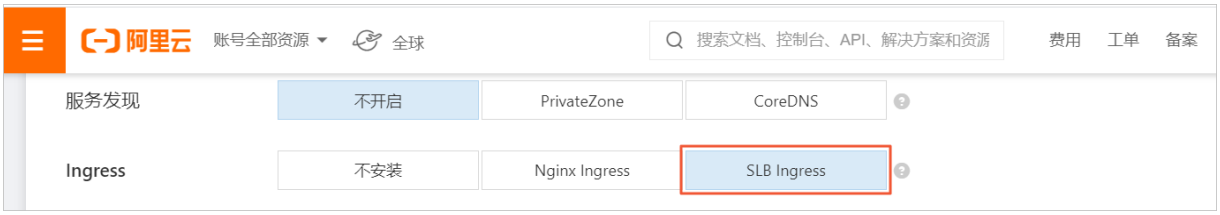
13.3.2. SLB Ingress Controller组件管理

阿里云Serverless Kubernetes集群，基于SLB七层转发规则提供了托管的SLB Ingress Controller。本文介绍如何在ASK集群安装和卸载SLB Ingress Controller。

安装SLB Ingress Controller

方式一：创建集群时安装SLB Ingress Controller

创建ASK集群时，在**Ingress**参数配置区域，选择安装**SLB Ingress**。具体操作，请参见ASK使用快速入门。



方式二：在组件管理页面安装SLB Ingress Controller

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SLB Ingress Controller组件区域单击**安装**。

卸载SLB Ingress Controller

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SLB Ingress Controller组件区域单击**卸载**。

删除SLB服务网关

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，从**命名空间**列表中，选择**kube-system**。
3. 在服务列表中单击**alb-ingress-lb**右侧操作列的**删除**。
4. 在**提示**对话框，单击**确定**。

13.3.3. 使用默认生成的SLB实例

如果您创建的Ingress没有通过Annotation指定SLB ID，ASK会自动生成一个公网SLB实例。本文介绍在该场景下如何使用该SLB提供Ingress的转发功能。

前提条件

- 您已创建一个ASK集群，集群的VPC需要配置NAT网关，从而可以访问外网，下载容器镜像。具体操作，请参见ASK使用快速入门。

- 您已通过kubectl连接到集群。具体操作，请参见通过kubectl连接Kubernetes集群。

操作步骤

步骤一：部署服务

1. 创建并拷贝以下内容到*cafe-service.yaml*文件中，并执行 `kubectl apply -f cafe-service.yaml` 命令，分别部署一个coffee服务和tea服务。

```
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
 clusterIP: None
---
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
 clusterIP: None
```

预期输出：

```
deployment "coffee" created
service "coffee-svc" created
deployment "tea" created
service "tea-svc" created
```

2. 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
svc/coffee-svc  ClusterIP  <none>    <none>    80/TCP   1m
svc/tea-svc    ClusterIP  <none>    <none>    80/TCP   1m
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，并执行 `kubectl apply -f cafe-ingress.yaml` 命令，通过Ingress配置coffee和tea服务对外暴露的域名和path路径。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: cafe-ingress
spec:
 rules:
 # 配置七层域名。
 - host: foo.bar.com
  http:
   paths:
   # 配置Context Path。
   - path: /tea
    backend:
     service:
      name: tea-svc
      port:
       number: 80
    pathType: ImplementationSpecific
   # 配置Context Path
   - path: /coffee
    backend:
     service:
      name: coffee-svc
      port:
       number: 80
    pathType: ImplementationSpecific
```

预期输出：

```
ingress "cafe-ingress" created
```

2. 执行以下命令获取SLB实例IP。

```
kubectl get ing
```

预期输出：

```
NAME      HOSTS     ADDRESS      PORTS   AGE
cafe-ingress  foo.bar.com  139.168.XX.XX   80     1m
```

步骤三：访问服务

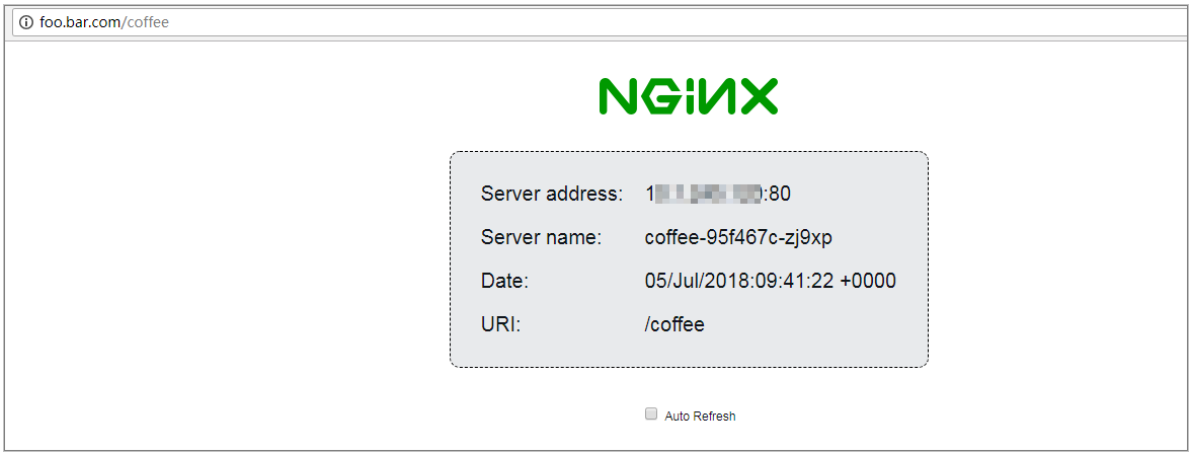
> **说明** 目前需要您自行将域名解析到SLB实例IP上。

本例在域名中添加一条DNS域名解析规则，用于测试服务访问。建议您在工作环境中对域名进行备案。

```
139.168.XX.XX   foo.bar.com
```

访问coffee服务。

- 通过浏览器访问coffee服务。

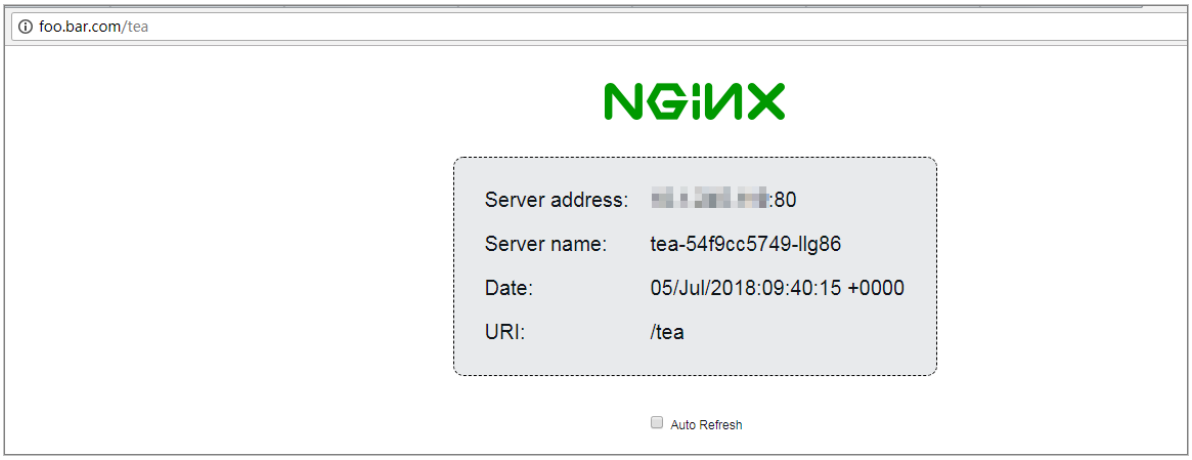


- 通过命令行方式访问coffee服务。

```
curl -H "Host: foo.bar.com" http://139.168.XX.XX/coffee
```

访问tea服务。

- 通过浏览器测试访问tea服务。



- 通过命令行方式测试访问tea服务。

```
curl -H "Host: foo.bar.com" http://139.168.XX.XX/tea
```

13.3.4. 使用指定的SLB实例

本文介绍在创建的Ingress中通过Annotation指定了SLB ID场景下，如何使用该SLB提供Ingress的转发功能。

前提条件

- 您已创建一个ASK集群，集群的VPC需要配置NAT网关，从而可以访问外网，下载容器镜像。具体操作，请参见ASK使用快速入门。
- 您已通过kubectl连接到集群。具体操作，请参见通过kubectl连接Kubernetes集群。
- 您已经创建一个和ASK集群在同一个VPC的性能保障型规格（支持ENI）SLB实例 。
  - 如果您在ASK集群同VPC下已经有一个SLB实例，您可以直接登录负载均衡管理控制台的**实例管理**页面获取SLB ID。
  - 如果您没有SLB ID，您需要在集群同VPC下创建一个性能保障型SLB实例（例如：*slb.s2.small*），该实例类型可以是私网或公网。具体操作，请参见创建实例。

- 本文操作示例中申请的是公网SLB实例。

操作步骤

> **说明** Ingress Controller会自动初始化SLB实例的80和443端口，请确保当前没有其他服务使用。

步骤一：部署服务

1. 创建并拷贝以下内容到*tomcat-service.yml*文件中，并执行 `kubectl apply -f tomcat-service.yml` 命令，部署一个tomcat测试应用。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tomcat
spec:
 replicas: 1
 selector:
  matchLabels:
   run: tomcat
 template:
  metadata:
   labels:
    run: tomcat
  spec:
   containers:
   - image: tomcat:7.0
    imagePullPolicy: Always
    name: tomcat
    ports:
    - containerPort: 8080
     protocol: TCP
   restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
 name: tomcat
spec:
 ports:
 - port: 8080
  protocol: TCP
  targetPort: 8080
 selector:
  run: tomcat
 clusterIP: None
```

预期输出：

```
deployment "tomcat" created
service "tomcat" created
```

2. 执行以下命令查看应用状态。

```
kubectl get svc,deploy tomcat
```

预期输出：

```
NAME     TYPE    CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
svc/tomcat  ClusterIP  <none>   <none>    8080/TCP  1m
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
deploy/tomcat  1    1    1      1      1m
```

步骤二：配置Ingress

1. 创建并拷贝如下内容到*tomcat-ingress.yml*文件中，并执行 `kubectl apply -f tomcat-ingress.yml` 命令，配置Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: tomcat-ingress
 annotations:
  # 配置使用指定的SLB实例（SLB ID）。
  service.beta.kubernetes.io/alicloud-loadbalancer-id: lb-xxxxxxxxxx       ##替换为你的SLB ID。
  service.beta.kubernetes.io/alicloud-loadbalancer-force-override-listeners: "true"
spec:
 rules:
 # 配置七层域名。
 - host: bar.foo.com
  http:
   paths:
   # 配置Context Path。
   - path: /
    backend:
     service:
      name: tomcat
      port:
       number: 8080
    pathType: ImplementationSpecific
```

预期输出：

```
ingress "tomcat-ingress" created
```

2. 执行以下命令获取SLB实例IP。

```
kubectl get ing tomcat-ingress
```

预期输出：

```
NAME       HOSTS     ADDRESS     PORTS   AGE
tomcat-ingress  bar.foo.com  47.168.XX.XX  80, 443  1m
```

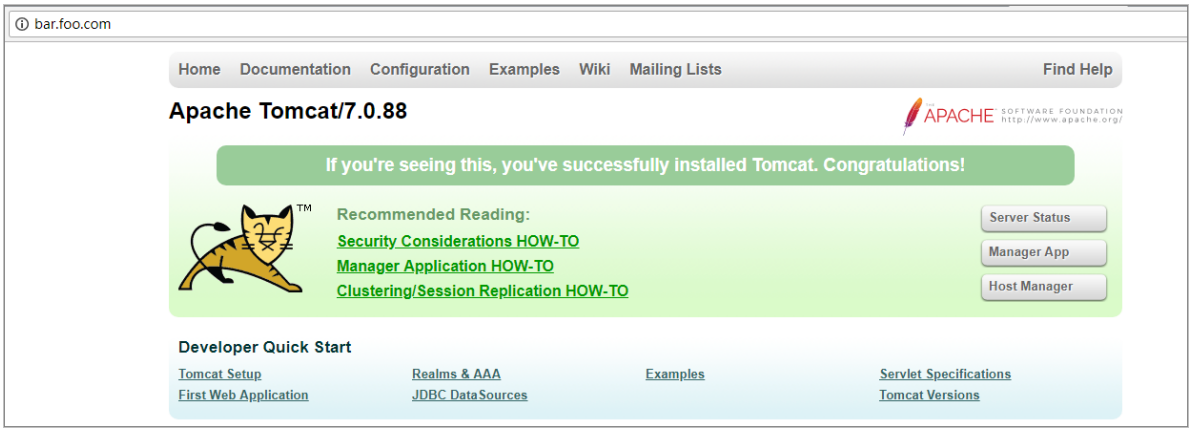
步骤三：访问服务

> **说明** 目前需要您自行将域名解析到SLB实例IP上。

本例在域名中添加一条DNS域名解析规则，用于测试服务访问。建议您在工作环境中对域名进行备案。

```
47.168.XX.XX  bar.foo.com
```

- 通过浏览器访问tomcat服务。



- 通过命令行方式访问tomcat服务。

```
curl -k -H "Host: bar.foo.com" https://47.168.XX.XX
```

13.3.5. 通过Secret配置TLS证书实现HTTPS访问

本示介绍Ingress如何通过Secret配置TLS证书实现HTTPS访问。

前提条件

- 您已创建一个ASK集群，集群的VPC需要配置NAT网关，从而可以访问外网，下载容器镜像。具体操作，请参见ASK使用快速入门。
- 您已通过kubectl连接到集群。具体操作，请参见通过kubectl连接Kubernetes集群。

操作步骤

步骤一：部署服务

1. 创建并拷贝以下内容到*cafe-service.yaml*文件中，并执行 `kubectl apply -f cafe-service.yaml` 命令，分别部署一个coffee服务和tea服务。

```
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
 clusterIP: None
---
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
 clusterIP: None
```

系统输出类似以下结果：

```
deployment "coffee" created
service "coffee-svc" created
deployment "tea" created
service "tea-svc" created
```

2. 执行以下命令查看服务状态。

```
kubectl get svc,deploy
```

系统输出类似以下结果：

```
NAME       TYPE    CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
svc/coffee-svc  ClusterIP  <none>    <none>    80/TCP   1m
svc/tea-svc    ClusterIP  <none>    <none>    80/TCP   1m
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TLS证书

您需要配置TLS证书实现HTTPS访问。

1. 执行以下命令，生成TLS证书。

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

系统输出类似以下结果：

```
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'tls.key'
-----
```

2. 执行以下命令根据生成的TLS证书文件创建集群的secret。

```
kubectl create secret tls cert-example --key tls.key --cert tls.crt
```

系统输出类似以下结果：

```
secret/cert-example created
```

3. 执行以下命令，查看新建TLS证书配置。

```
kubectl get secret cert-example
```

系统输出类似以下结果：

```
NAME      TYPE        DATA   AGE
cert-example  kubernetes.io/tls  2     12s
```

> 注意
>
> - Ingress Controller会自动依据第一个创建的Ingress的TLS证书来初始化SLB的HTTPS默认证书，您只能通过Ingress引用的secret修改证书配置，不可在SLB控制台自行修改。当前不支持配置多个证书，若您想要使用多个域名的多个证书您可以通过复用SLB的方式为每一个证书分别指定一个SLB实例。
> - 您手动在SLB控制台上面做的任何修改都有可能导致Ingress服务异常，请不要在SLB控制台做修改。

步骤三：配置Ingress

1. 创建并拷贝以下内容到*cafe-ingress.yaml*文件中，并执行 `kubectl apply -f cafe-ingress.yaml` 命令，通过Ingress配置coffee和tea服务对外暴露的域名和path路径。

```
# cat  cafe-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: cafe-ingress
spec:
 tls:
 - hosts:
  - foo.bar.com
  secretName: cert-example
 rules:
 # 配置七层域名
 - host: foo.bar.com
  http:
   paths:
   # 配置Context Path
   - path: /tea
    backend:
     service:
      name: tea-svc
      port:
       number: 80
    pathType: ImplementationSpecific
   # 配置Context Path
   - path: /coffee
    backend:
     service:
      name: coffee-svc
      port:
       number: 80
    pathType: ImplementationSpecific
```

   系统输出类似以下结果：

```
ingress "cafe-ingress" created
```

2. 执行以下命令获取SLB实例IP。

```
kubectl get ing
```

系统输出类似以下结果：

```
NAME      HOSTS     ADDRESS      PORTS   AGE
cafe-ingress  foo.bar.com  139.168.XX.XX   80     1m
```

步骤四：访问服务

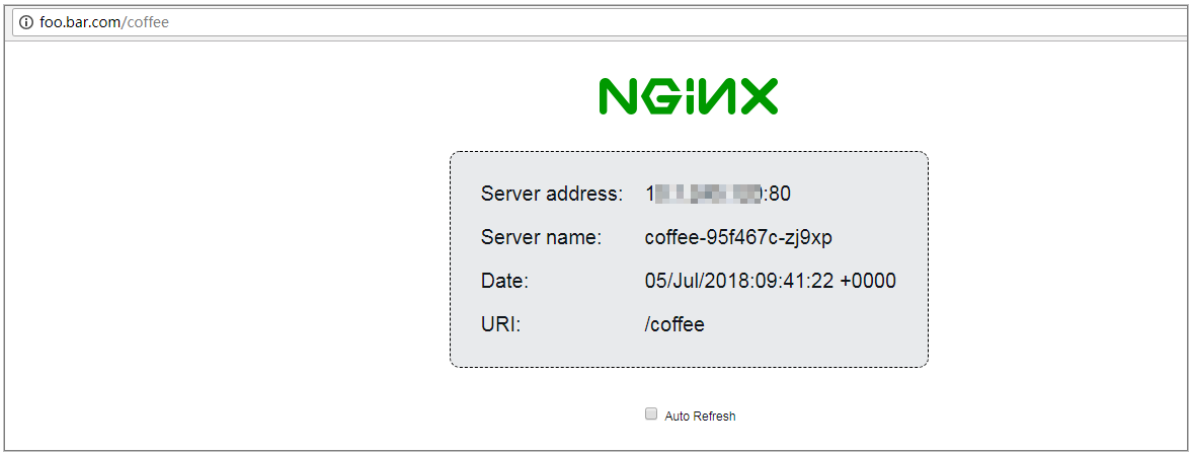
> **说明** 目前需要您自行将域名解析到SLB实例IP上。

本例在域名中添加一条DNS域名解析规则，用于测试服务访问。建议您在工作环境中对域名进行备案。

```
139.168.XX.XX   foo.bar.com
```

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 -H "Host: foo.bar.com" http://139.168.XX.XX/coffee
```

访问tea服务。

- 通过浏览器测试访问tea服务。

- 通过命令行方式测试访问tea服务。

```
curl -H "Host: foo.bar.com" https://139.168.XX.XX/tea -k
```

13.4. Nginx Ingress管理

13.4.1. Nginx Ingress概述

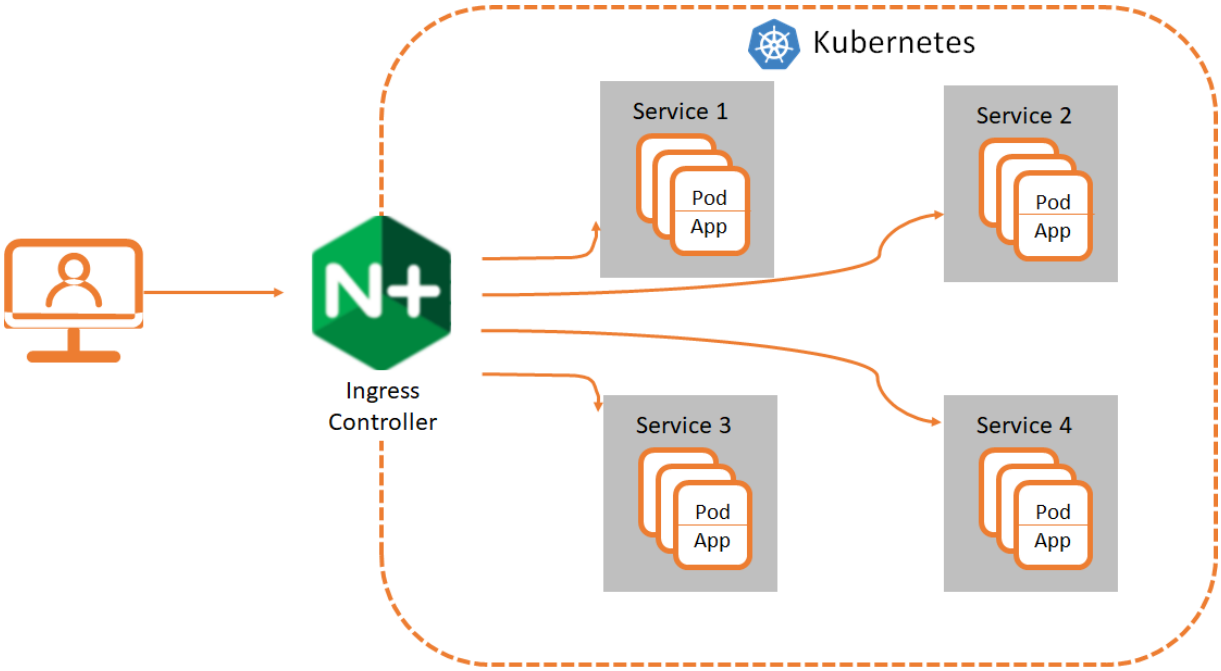
本文介绍Ingress基本概念、Ingress Controller工作原理和ASK的Ingress。

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 Controller工作原理

Ingress Controller通过API Server获取Ingress资源的变化，动态地生成Load Balancer所需的配置文件，然后依次生成新的路由转发规则。



ASK的Ingress

阿里云Serverless Kubernetes（ASK）官方基于社区版的Nginx Ingress Controller进行了优化，提供Nginx Ingress Controller功能。您可以在创建ASK集群时选择安装该组件，也可以创建集群后在集群的组件管理页面中手动安装。

相关文档

- 创建Ingress路由
- Ingress高级用法
- SLB Ingress概述

13.4.2. 安装Nginx Ingress Controller

阿里云Serverless Kubernetes（ASK）官方基于社区版的Nginx Ingress Controller进行了优化，提供Nginx Ingress Controller功能。本文介绍如何在ASK集群安装Nginx Ingress Controller。

操作步骤

方式一：创建集群时安装Nginx Ingress Controller

创建ASK集群时，在**Ingress**参数配置区域，选择安装**Nginx Ingress** 。具体操作，请参见ASK使用快速入门。

> **说明** 安装Nginx Ingress Controller需要满足以下两个条件：
> - 2个2核4 GiB的ECI实例：用于部署服务。
> - 1个SLB：SLB实例类型可以是公网或者私网，SLB实例规格您可以按需选择。

方式二：在组件管理页面安装Nginx Ingress Controller

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**网络**，在**Nginx Ingress Controller**组件区域单击**安装**。
6. 在**提示**对话框，单击**确定**。
   如果在**Nginx Ingress Controller**组件区域右上角显示**已安装**，说明您已安装该组件。

13.4.3. 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。

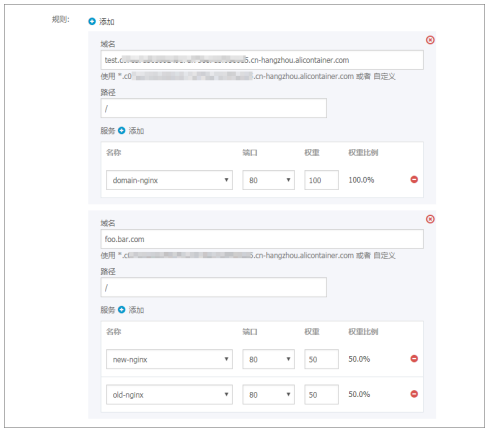
控制台操作指导

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：

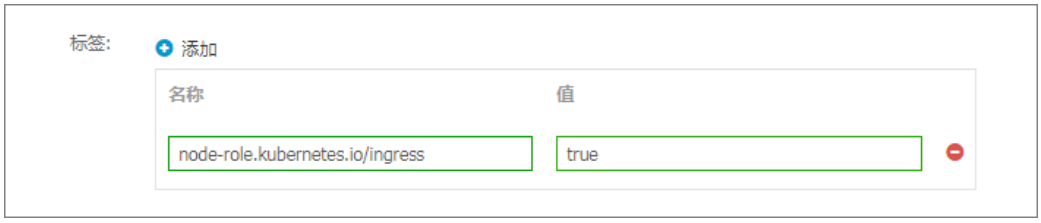
   - 选择**自定义注解**：输入注解名称和值，即Ingress的Annotation键值对，Ingress的注解请参见Annotations。
   - 选择**Ingress-Nginx**：根据名称选择或搜索要配置的注解。

     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>
```

13.4.4. 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 Authority'
```

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 -C3 "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自动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://raw.githubusercontent.com/AliyunContainerService/serverless-k8s-examples/master/cert-manager/ask-cert-manager.yaml
```

> **说明** 上述命令中的YAML示例仅适用于在ASK集群中部署cert-manager。关于如何在ACK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。

13.5. 网络管理最佳实践

13.5.1. Serverless集群基于云解析PrivateZone的服务发现2021年6月18日，与冬岛核实，下线ASK最佳实践节点下该页面内容，保留网络管理最佳实践下的页面，故将ASK最佳实践下的源头页面，挪动至网络管理最佳实践节点下，下线复用页面的内容。

阿里云Serverless Kubernetes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。
- 创建Serverless Kubernetes集群。
- 您已成功连接到Kubernetes集群，参见通过kubectl连接Kubernetes集群。

背景信息

云解析PrivateZone，是基于阿里云专有网络VPC（Virtual Private Cloud）环境的私有域名解析和管理服务。您能够在自定义的一个或多个专有网络中将私有域名映射到IP资源地址，同时在其他网络环境无法访问您的私有域名。

操作步骤

1. 部署Deployment和创建Service。

   样例模板如下所示，在YMAL文件中复制如下YAML代码，然后执行 `kubectl create -f nginx-service.yaml` 命令进行创建。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-headless-service
spec:
 ports:
 - port: 80
  protocol: TCP
 selector:
  app: nginx
 clusterIP: None
---
apiVersion: v1
```

```
kind: Service
metadata:
 name: nginx-clusterip-service
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
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
```

2. 执行以下命令，查看应用的运行状况。

```
kubectl get svc,pod,deployment
```

3. 登录云解析DNS控制台。
4. 在控制台左侧导航栏中，单击**PrivateZone**，选择**权威Zone**页签。
5. 选中目标Zone，单击目标Zone右侧**操作**列下的**解析设置**。

> 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中只对应一条解析Record，为Cluster 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。

更多信息，请参见serverless-k8s-examples。

13.5.2. 配置安全组

安全组是一种虚拟防火墙，具备状态检测和数据包过滤能力，用于在云端划分安全域。通过添加安全组规则，您可以控制安全组内ECI实例的入流量和出流量。

安全组概述

安全组定义

安全组是一个逻辑上的分组，由同一地域内具有相同安全保护需求并相互信任的实例组成。通过添加安全组规则，安全组可以允许或拒绝安全组内ECI实例对公网或者私网的访问，以及管理是否放行来自公网或私网的访问请求。

> 说明
>
> - 一个安全组可以管理同一个地域内的多台ECI实例。
> - 一台ECI实例必须且仅支持属于一个安全组。

安全组类型

安全组分为普通安全组和企业安全组，创建时默认添加的安全组规则如下：

- 入方向：放行80、443、22、3389及ICMP协议，可修改。
- 出方向：允许所有访问请求。

两种安全组主要的功能差异如下表所示。

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

功能	普通安全组	企业安全组
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
授权给其它安全组	支持组组授权	不支持组组授权

> **注意**
>
> 如果您对整体规模和运维效率有较高需求，建议您使用企业安全组。相比普通安全组，企业安全组大幅提升了组内支持容纳的实例数量，简化了规则配置方式。

安全组规则

安全组通过配置规则来控制出入流量。一条安全组规则由规则方向、授权策略、协议类型、端口范围、授权对象等属性确定。关于安全组规则，请注意以下事项：

- 每个安全组的入方向规则与出方向规则的总数不能超过200条。
- 添加规则时遵守最小授权原则。例如：
  - 选择开放具体的端口，如80/80，避免开放端口范围，如1/80。
  - 谨慎授权全网段访问源，即0.0.0.0/0。

更多信息，请参见安全组概述。

指定安全组

创建ECI实例时，必须要指定安全组，将ECI实例加入到安全组中。

> **注意**
>
> ECI实例不支持修改安全组。如果想要变更安全组，需要重新创建ECI实例。

Kubernetes方式

在Kubernetes场景中通过Virtual Kubelet（简称VK）使用ECI时，集群中所有ECI实例将默认加入到VK设置的安全组中。如果有特殊需求，您也可以为某个ECI实例指定其它安全组。

- **集群**

  您可以通过kubectl edit命令修改eci-profile配置文件，在data中修改ECI实例默认使用的安全组ID。

> **说明**
>
> VK版本为v2.0.0.90-15deb126e-aliyun及以上时，支持修改eci-profile实现配置热更新。如果您的VK版本低于该版本，建议您升级VK。

```
kubectl edit configmap eci-profile -n kube-system
```

修改data中的securityGroupId字段，示例如下：

```
data:
 enableClusterIp: "true"
 enableHybridMode: "false"
 enablePrivateZone: "false"
 resourceGroupId: ""
 securityGroupId: sg-2ze0b9o8pjjzts4h**** #指定安全组ID
 selectors: ""
 vSwitchIds: vsw-2zeet2ksvw7f14ryz****,vsw-2ze94pjtfuj9vaymf****
 vpcId: vpc-2zeghwzptn5zii0w7****
```

- **ECI实例**

  对于单个ECI实例，您可以在Pod metadata中添加Annotation来指定安全组。配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo
 labels:
  app: nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:

      k8s.aliyun.com/eci-security-group: "sg-bp1dktddjsg5nktv****"    #设置安全组

    labels:
      app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
```

OpenAPI方式

调用CreateContainerGroup接口创建ECI实例时，您可以通过SecurityGroupId参数来指定安全组。SecurityGroupId的参数说明如下表所示。更多信息，请参见CreateContainerGroup。

名称	类型	示例值	描述
SecurityGroupId	String	sg-uf66jeqopgqa9hdn****	指定安全组ID。

控制台方式

通过弹性容器实例售卖页创建ECI实例时，您可以指定一个安全组。

添加安全组规则

对于安全组内的ECI实例，您可以添加安全组规则来控制其出入流量。例如：

- 当您的ECI实例需要与所在安全组之外的网络进行通信时，您可以添加允许访问的安全组规则，实现网络互通。
- 当您在运行ECI实例的过程中，发现部分请求来源有恶意攻击行为时，您可以添加拒绝访问的安全组规则，实现网络隔离。

关于如何添加安全组规则，请参见添加安全组规则。

14.日志

14.1. 通过阿里云日志服务采集日志

本文将介绍如何在Serverless Kubernetes集群中将业务容器的标准输出和日志文件收集到阿里云日志服务。

前提条件

已创建Serverless Kubernetes集群。具体操作，请参见创建Serverless Kubernetes集群。

使用YAML模版来部署示例

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**工作负载 > 无状态**。
5. 在**无状态**页面，单击页面右上角的**使用YAML创建资源**。
6. 新建并部署以下YAML模板。

   YAML模板的语法同Kubernetes语法，采集配置通过ENV来暴露。以下是一个简单的Deployment示例。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
  app: alpine
 name: alpine
spec:
 replicas: 2
 selector:
  matchLabels:
   app: alpine
 template:
  metadata:
   labels:
    app: alpine
  spec:
   containers:
   - image: alpine
    imagePullPolicy: Always
    args:
    - ping
    - 127.0.0.1
    name: alpine
    env:
    #配置环境变量。
    #配置Project，如果使用K8s集群默认的Project可以不填。
    - name: aliyun_logs_test-stdout_project
     value: k8s-log-xxx
    - name: aliyun_logs_test-file_project
     value: k8s-log-xxx
    #配置机器组，如果使用K8s集群默认的Project下的默认机器组可以不填。
    - name: aliyun_logs_test-stdout_machinegroup
     value: k8s-group-app-alpine
    - name: aliyun_logs_test-file_machinegroup
     value: k8s-group-app-alpine
    #设置标准输出和错误输出的logstore test-stdout。
    - name: aliyun_logs_test-stdout
     value: stdout
    #将/log/*.log目录下的日志收集到logstore test-file。
    #除了标准输出和错误输出，其他的自定义收集目录必须挂载emptyDirVolume，否则无法收集。
    ######### 注：比如/log/目录就需要挂载emptyDirVolume，如下############
    - name: aliyun_logs_test-file
     value: /log/*.log
    ######### 日志保留时间，只对单个logstore生效 ############
    - name: aliyun_logs_test-stdout_ttl
     value: "7"
    ######### 日志分区数，只对单个logstore生效 ############
    - name: aliyun_logs_test-stdout_shard
     value: "2"
```

其中以下内容需要根据您的需求进行配置，一般按照顺序进行配置。

- 通过环境变量来创建您的采集配置，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 alivun_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。

- 如果您的采集配置中指定了非 stdout 的采集路径，需要将容器的标准输出采集到该路径下。

7. 当YAML编写完成后，单击**创建**，即可将相应的配置交由Serverless Kubernetes集群执行。

   部署完成后，您可以通过执行以下命令，查看Pod部署情况。

   ```
   kubectl get Pods
   ```

   预期输出：

   ```
   NAME          READY   STATUS  RESTARTS  AGE    IP       NODE
   alpine-76d978dbdd-gznk6  1/1    Running  0     21m    10.1.XX.XX  viking-c619c41329e624975a7bb505
   27180****
   alpine-76d978dbdd-vb9fv  1/1    Running  0     21m    10.1.XX.XX  viking-c619c41329e624975a7bb505
   27180****
   ```

查看日志

1. 安装成功后，进入日志服务控制台。
2. 在进入控制台后，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），单击，在下拉列表中选择**查询分析**。

   本例中，找到 test-stdout 这个Logstore，单击，在下拉列表中选择**查询分析**，可以看到收集到的ECI弹性容器实例的 stdout 日志。

14.2. Job类型任务如何采集日志

本文将为您介绍如何针对Job类型任务场景采集日志到日志系统。

前提条件

- 已创建了一个ASK集群。更多信息，请参见ASK使用快速入门。
- 已在该集群上部署了虚拟节点。更多信息，请参见通过部署ACK虚拟节点组件创建ECI Pod。
- 已创建NAS文件系统并添加挂载点。更多信息，请参见创建文件系统和管理挂载点。

> **说明** 如果您使用了阿里云日志服务，Job任务挂载volume收集日志通过配置环境变量，可以直接同步阿里云日志服务。详情请参见ECI中日志采集的自定义配置。

操作步骤

> **注意**
>
> 在ECS模式下Job任务可以通过DeamonSet方式采集标准输出，但是在ECI模式下不支持DeamonSet。当Job任务结束后，Pod会立即退出，此时日志可能还未被收集完成，针对这种情况我们可以采用如下方式解决：
>
> Job类任务挂载NAS盘，把输出的日志存储在NAS盘，再通过另一个同样挂载NAS盘的Pod来采集Job任务标准输出到日志系统中。

1. 通过kubectl客户端创建*job.yaml*

   一个计算π值Job任务：

```
apiVersion: batch/v1
kind: Job
metadata:
 name: pi
spec:
 template:
  spec:
   containers:
   - name: pi
    image: resouer/ubuntu-bc
    command: ["sh", "-c", "echo 'scale=1000; 4*a(1)' | bc -l > /eci/a.log 2>&1"] #运行输出结果重定向到指定文件
     volumeMounts:
     - name: log-volume
      mountPath: /eci
      readOnly: false
   restartPolicy: Never
   volumes:
   - name: log-volume
    nfs:
      path: /eci
      server: 04edd48c7c-****.cn-hangzhou.nas.aliyuncs.com
      readOnly: false
 backoffLimit: 4
```

2. 部署一个Job任务到虚拟节点。

   > **说明** 关于如何部署虚拟节点请参见通过部署ACK虚拟节点组件创建ECI Pod。

```
kubectl apply -f job.yaml -n fvt-eci
```

3. 查看Pod状态。

```
kubectl get pod -n fvt-eci
```

4. 通过kubectl客户端创建*log-collection.yaml*文件，并拷贝以下内容到该文件。然后执行指令创建一个Pod，挂载NAS盘用来采集Job任务输出日志。

```
apiVersion: v1
kind: Pod
metadata:
 name: log-collection
spec:
 containers:
 - image: nginx:latest
  name: log-collection
  command: ['/bin/sh', '-c', 'echo &(cat /eci/a.log)'] #查看Job日志文件
  volumeMounts:
  - mountPath: /eci
    name: log-volume
 restartPolicy: Never
 volumes:
 - name: log-volume
  nfs:
    server: 04edd48c7c-****.cn-hangzhou.nas.aliyuncs.com
    path: /eci
    readOnly: false
```

14.3. ECI中日志采集的自定义配置

用户自定义设置

根据业务需要，您可能需要将ECI的日志收集到自定义项目下的自定义日志库里。对于不同的应用和服务，您可能还需要将ECI实例加入不同的机器组。对于项目、日志库和机器组等自定义配置需求，您可以通过两种办法实现：

- **通过日志服务控制台（API）手动设置**

用户可以自行登录日志服务控制台，创建自定义项目，创建自定义日志库，以及自定的机器组，为日志库创建自定义config并应用到选择的机器组。这样日志内容就可以导向新的日志库了。

如果觉得通过日志服务控制台配置太繁琐，您可以通过ECI代创建和配置。

- **通过ECI自定义**

ECI除了具备为用户生成所有默认设置外，还支持为用户生成自定义的配置。比如项目名、日志库名、配置名、机器组名、以及日志收集目录等。具体的参数通过ECI的容器的环境变量传入，格式如下：

Logstore名和配置名

首先是配置名

```
-name: aliyun_logs_{配置名}
-value: {日志采集路径}
```

> **注意**
>
> 如需采集标准输出，请将日志采集路径设置为stdout。

默认情况下，logstore的名字和配置名同名，如果需要设置配置所输出的logstore的名字，可以采用如下的方式自定义：

```
-name: aliyun_logs_{配置名}_logstore
-value: {logstore 名称}
```

日志库名约束

- 日志库名称仅支持小写字母、数字、连字符（-）和下划线（_）。
- 必须以小写字母和数字开头和结尾。
- 名称长度为3-63个字符。

> **注意**
>
> 校验不通过的，会直接忽略，使用ECI默认的。

项目名

设置日志收集所属的project，方式如下。

```
-name: aliyun_logs_{配置名}_project
-value: {project 名称}
```

默认情况下，对于ECI的API用户，每个地域会有一个默认的project，对于k8s的用户，默认project为每个集群一个，命名方式为“k8s-log-{k8s集群id}”

项目名约束

- 项目名称仅支持小写字母、数字和连字符（-）。
- 必须以小写字母和数字开头和结尾。
- 名称长度为3-63个字符。

> **注意**
>
> 校验不通过的，会直接忽略，使用ECI默认的。

Logstore设置分区数

什么是分区（Shard），请参见分区。

设置方法：

```
-name: aliyun_logs_{配置名}_shard
-value: {shard数值}
```

默认值为2，可选范围是[1,10]。

Logstore设置日志保留时间

设置方法：

```
-name: aliyun_logs_{配置名}_ttl
-value: {ttl数值}
```

默认值为90，可选范围是[1,3650]。

机器组名称

非必填参数，

默认情况，

对于ECI API 用户，ECI实例会加入到ECI帮用户创建的默认机器组，一个region对应一个。

对于 kubernetes 用户，ECI实例会加入集群默认的机器组，命名格式“k8s-group-{k8s集群id}”。

自定义设置的格式如下：

```
-name: aliyun_logs_{配置名}_machinegroup
-value: {机器组名称}
```

机器组名约束

- 机器组名称仅支持字母、数字、连字符（-）和下划线（_）。
- 必须以小写字母和数字开头和结尾。
- 名称长度为3-63个字符。

> **注意**
>
> 校验不通过的，会直接忽略，使用ECI默认的。

用户Volume日志收集

对于标准输出和错误输出，只需要通过环境变量进行设置就可以收集；如需要收集任意的自定的目录下的日志文件，需要依赖Volume才可以进行收集。

Volume的标准日志收集目录为Volume挂载的目录下的子目录，具体取决于用户自己的设定。

比如：

用户有个EmptyDirVolume，挂载到了容器的/pod/data/目录下，那么Volume的日志收集可以指定是/pod/data/下的任意子目录下的任意文件。通过这种方式，用户可以灵活的调整挂载目录并配合自己的业务，实现自定义的日志收集目录。

创建EmptyDirVolume

```
'Volume.1.Name': 'default-volume',
'Volume.1.Type': 'EmptyDirVolume',
```

将Volume挂载至容器目录

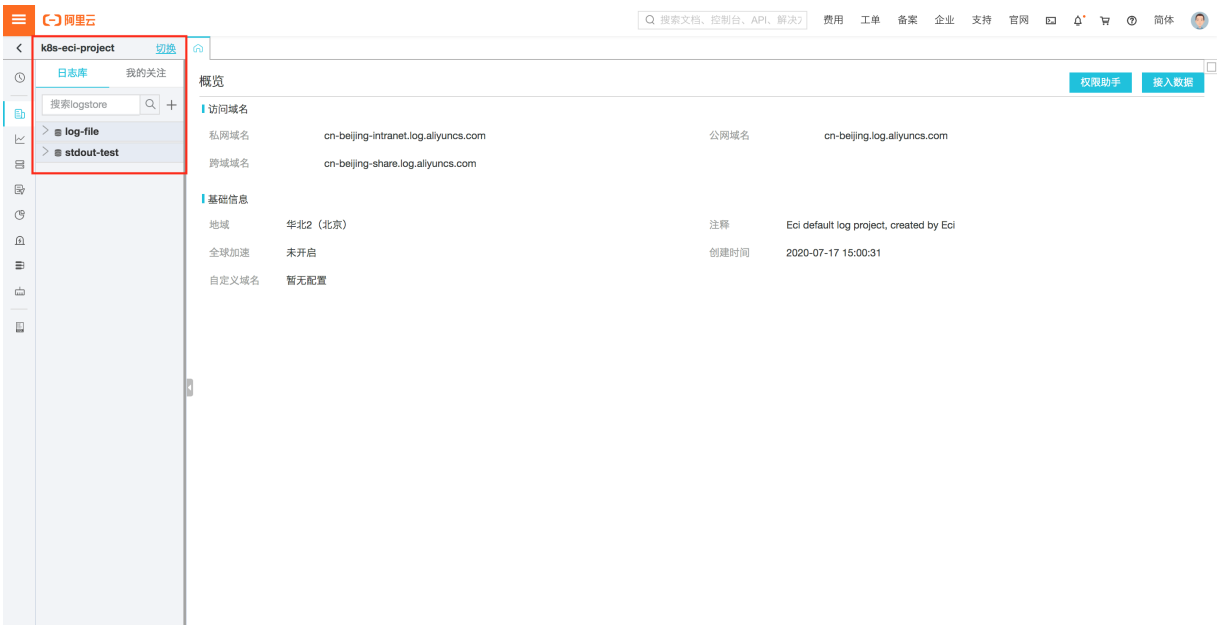
```
'Container.1.VolumeMount.1.Name': 'default-volume',
'Container.1.VolumeMount.1.MountPath': '/pod/data/',
'Container.1.VolumeMount.1.ReadOnly': False,
```

配置日志仓库

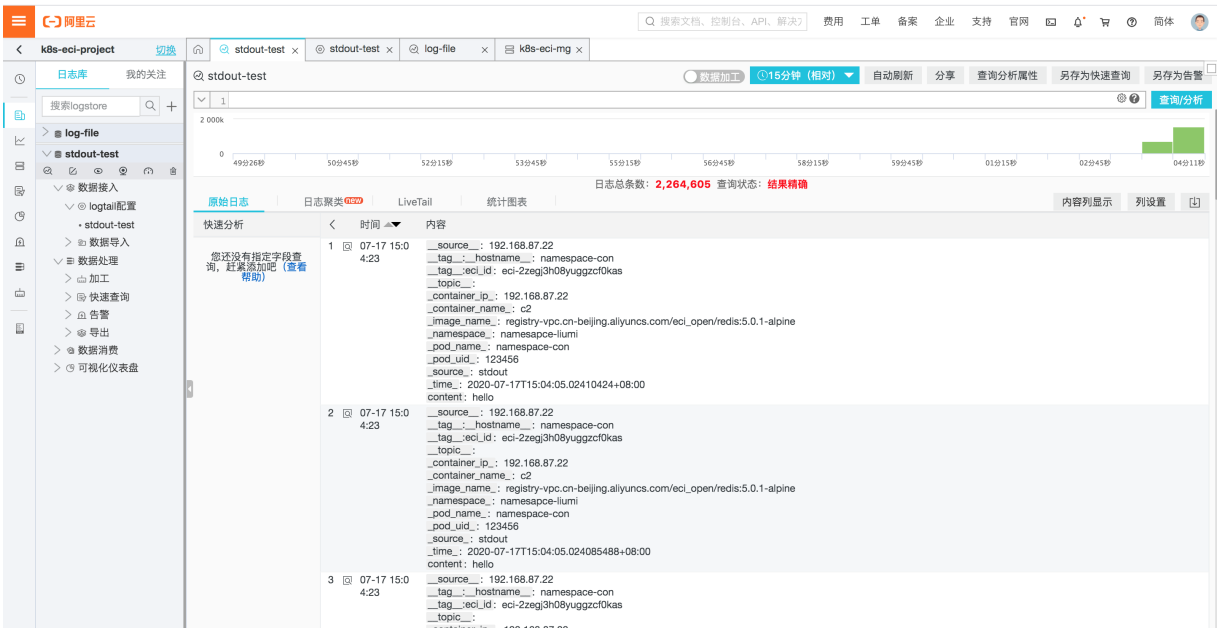
'aliyun_logs_stdout-test' 为ECI的容器的标准输出的收集目录，'aliyun_logs_log-file' 为Volume的日志收集目录，模糊匹配/pod/data/目录下的任意日志文件。

```
'Container.1.EnvironmentVar.1.Key': 'aliyun_logs_log-file',
'Container.1.EnvironmentVar.1.Value': '/pod/data/*.log',
'Container.1.EnvironmentVar.2.Key': 'aliyun_logs_stdout-test',
'Container.1.EnvironmentVar.2.Value': 'stdout',
'Container.1.EnvironmentVar.3.Key': 'aliyun_logs_log-file_project',
'Container.1.EnvironmentVar.3.Value': 'k8s-eci-project',
'Container.1.EnvironmentVar.4.Key': 'aliyun_logs_stdout-test_project',
'Container.1.EnvironmentVar.4.Value': 'k8s-eci-project',
'Container.1.EnvironmentVar.5.Key': 'aliyun_logs_log-file_machinegroup',
'Container.1.EnvironmentVar.5.Value': 'k8s-eci-mg',
'Container.1.EnvironmentVar.6.Key': 'aliyun_logs_stdout-test_machinegroup',
'Container.1.EnvironmentVar.6.Value': 'k8s-eci-mg',
```

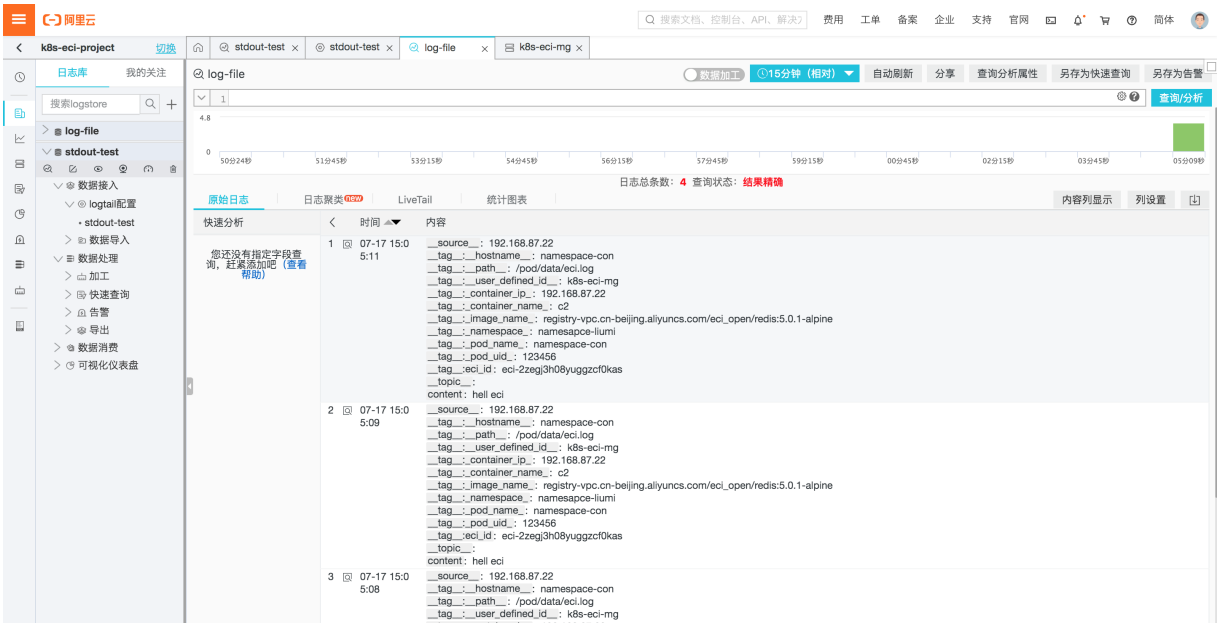
效果



标准输出日志：



日志文件收集：



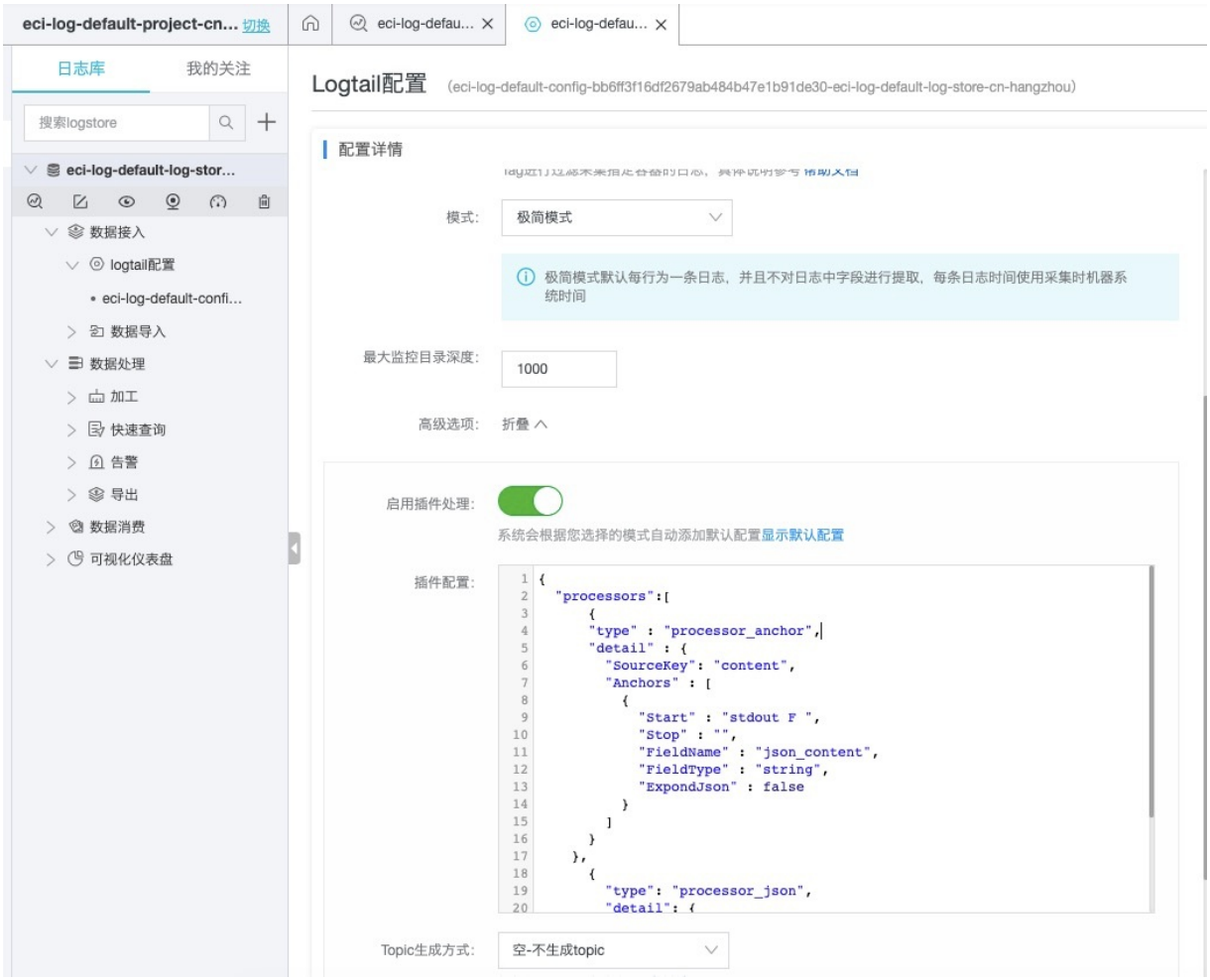
14.4. 用户日志JSON解析

ECI收集的用户标准输出、错误日志是原生K8S落盘的日志，K8S会在每行日志前增加时间戳、来源等信息，这就破坏了用户原生的日志格式，比如用户的标准输出原生记录的是JSON格式，K8S添加前缀后JSON解析就会失败，如下：

```
2020-04-02T15:40:05.440500764+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:07.442412564+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:09.442774495+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:11.443799303+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:13.445099622+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:15.445934358+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:17.447064707+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:19.448112987+08:00 stdout F {"key1":"val1","key2":"val2"}
2020-04-02T15:40:21.449393263+08:00 stdout F {"key1":"val1","key2":"val2"}
```

本文采用SLS Processor能力解决用户日志JSON解析的问题。

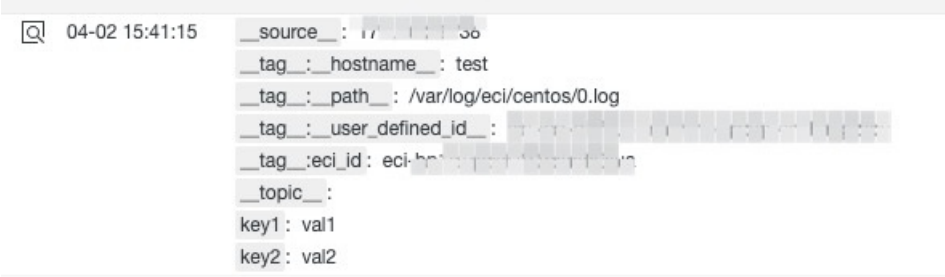
ECI的用户日志会收集到用账户下的日志仓库内，找到相应的logstore后修改配置，采用极简模式并启用插件处理能力，如下图：



插件配置内容如下： 参见sls-json-processor。

```
{
  "processors": [
    {
      "type": "processor_anchor",
      "detail": {
        "SourceKey": "content",
        "Anchors": [
          {
            "Start": "stdout F ",
            "Stop": "",
            "FieldName": "json_content",
            "FieldType": "string",
            "ExpondJson": false
          }
        ]
      }
    },
    {
      "type": "processor_json",
      "detail": {
        "SourceKey": "json_content",
        "KeepSource": false,
        "ExpandConnector": ""
      }
    }
  ]
}
```

保存配置后隔几秒，就可以查看到正常解析的日志内容，如下所示：



可见，SLS已经正确解析了JSON日志。

15.监控

15.1. 将应用实时监控服务ARMS接入ASK集群

阿里云Serverless Kubernetes（ASK）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接入至标准的Serverless集群。

前提条件

创建Serverless Kubernetes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

15.2. 阿里云Prometheus监控

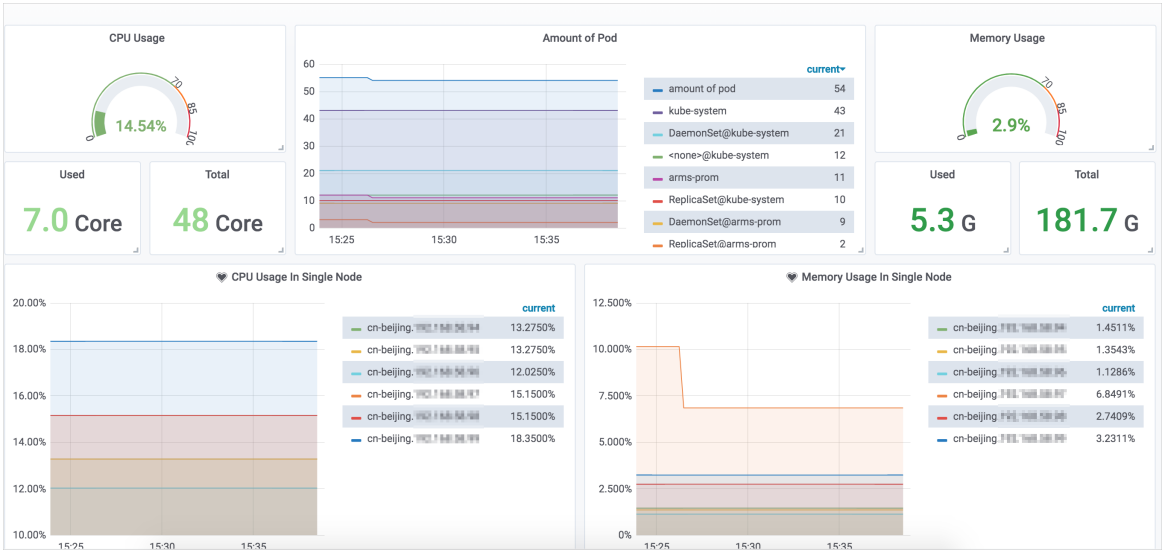
您可以通过阿里云Prometheus监控查看Serverless Kubernetes（ASK）集群预先配置的监控大盘和监控性能指标。本文为您介绍如何在ASK中接入阿里云的Prometheus监控。

背景信息

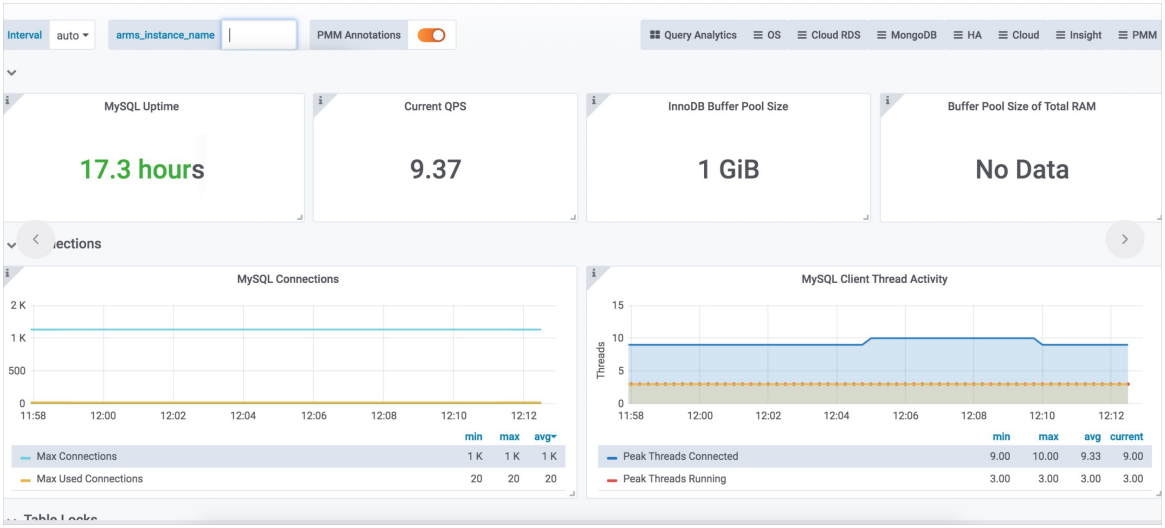
阿里云Prometheus监控全面对接开源Prometheus生态，支持类型丰富的组件监控，提供多种开箱即用的预置监控大盘，且提供全面托管的Prometheus服务。借助阿里云Prometheus监控，您无需自行搭建Prometheus监控系统，因而无需关心底层数据存储、数据展示、系统运维等问题。

整体而言，与开源Prometheus监控相比，阿里云Prometheus监控的优势体现为：

- 更轻量、更稳定、更准确的重试机制
  -
- 数据量无上限
  -
- 完全兼容开源生态
  -
- 节省成本
  - 阿里云Prometheus监控支持默认K8s监控。在您安装默认K8s监控后，阿里云Prometheus监控会自动为您创建默认的Exporter、采集规则、Grafana大盘以及ARMS告警。您的时间成本可由原来使用开源Prometheus监控K8s的3天左右降低至10分钟左右。



- 阿里云Prometheus监控支持开源组件监控。您还可以输入阿里云的RDS、Redis组件的账号和密码，阿里云Prometheus监控即可为您默认生成这些组件的Exporter，并为您创建默认的组件大盘。您的时间成本可由原来使用开源Prometheus监控开源组件的7天左右降低至3分钟左右。



- 阿里云Prometheus监控支持一键安装、一键卸载，以及通过健康检查功能调试Prometheus监控。您的时间成本可由原来使用开源Prometheus监控的1天左右降低至3分钟左右。

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

在控制台左侧导航栏选择**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**概览**页签下单击右上角的**prometheus监控**，跳转到prometheus控制台。在**监控列表**页面找到目标集群并单击已安装的插件，进入大盘查看监控数据。

16.Knative

16.1. 概述

社区Knative作为一款云原生、跨平台的Serverless编排引擎，您可以直接把社区Knative部署在ASK集群上，但这种方式需要付出额外的成本。ASK集成了Knative，您只需要拥有一个ASK集群并开通Knative功能，就可以基于Knative API使用云的能力。并且无需为Knative Controller付出任何成本。

ASK Knative优势

社区Knative	ASK Knative
默认使用Istio作为Gateway，安装Istio的Controller需要额外支出部分IaaS成本。	无需为Knative Controller付出任何成本。
Knative自身的Controller部署，需要额外支出部分IaaS成本。	
在ASK集群中创建一个Pod是有明显的冷启动时间，Knative的缩容到零机制可以很好的节省成本，但是第一批请求过来时有可能导致短暂的超时失败。	ASK Knative在没有流量的时候不把应用实例数缩容到零个，而是保留实例。通过低成本的保留实例来平衡成本和冷启动时长。保留实例的详细介绍请参见保留实例。

Knative资源托管

ASK作为无服务器的Kubernetes集群，无需单独购买节点即可直接部署容器应用，是一种理想的Kubernetes使用方式。以下为Knative资源托管优势：

- ASK上提供了Knative的托管服务，您可以通过Knative管理应用。
- Knative可以在需要的时候自动从ASK中申请IaaS资源，此处IaaS资源在ASK中指Pod。

Knative Serving Controller和阿里云容器服务进行了融合。您只需要拥有一个ASK集群并开通Knative功能就可以基于Knative API使用云的能力，并且无需为Knative Controller付出任何成本。

Knative Gateway

社区Knative默认支持Istio、Gloo、Contour、Kourier和Ambassador等多种Gateway实现方案。在这些实现方案中Istio使用频率最高。因为Istio除了可以充当Gateway的角色还能做为ServiceMesh服务使用。Serverless服务需要有Gateway实例常驻运行，而为了保证高可用至少要有两个实例互为备份。其次这些Gateway的Controller也需要常驻运行，这些常驻实例的IaaS费用和运维都是业务需要支付的成本。

为了极致的Serverless体验，通过阿里云SLB实现了Knative Gateway。Knative Gateway具备所有需要的功能并且属于云产品级别的支撑。不需要常驻资源，不仅节省了您的IaaS成本还省去了很多运维负担。

保留实例

社区Knative默认在没有流量时可以把应用实例缩容到零，但是缩容到零之后，从零到一的冷启动问题很难解决。冷启动除了要解决IaaS资源的分配、Kubernetes的调度、拉镜像等问题以外，还涉及到应用的启动时长。而应用镜像的大小以及应用启动时长与具体的开发者或者业务有很强的关联。

ASK的Knative和社区Knative不同点在于默认在没有流量的时候不把应用实例数缩容到零个，而是保留一个实例。以下为保留实例的策略：

- 在业务波谷时使用突发性能实例替换标准的计算型实例，当第一个请求来临时再无缝切换到标准的计算型实例，从而降低流量低谷的成本。

- 流量低谷时获得的CPU积分可以在业务高峰到来时使用，节约了使用成本。

保留实例的详细介绍请参见保留实例。

Knative安装部署

Serverless Kubernetes （ASK）集群支持部署Knative。

- 如果您的ASK集群版本≥1.16，可以直接通过控制台部署Knative，详细介绍请参见开启Knative。
- 如果您的ASK集群版本<1.16，请先升级ASK集群。

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ECI容器实例、负载均衡实例、NAT网关等会按照相应资源的价格计费。更多信息，请参见容器实例计费概述。

16.2. 社区Knative简介

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。

16.3. Knative版本发布说明

16.3.1. 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` 。

16.4. Knative组件管理

16.4.1. 开启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。本文介绍如何在ASK开启Knative。

背景信息

Knative当前支持在Kubernetes专有版集群、Kubernetes托管版集群以及Serverless Kubernetes （ASK）集群部署。目前在ASK集群中部署的Knative暂只支持Knative Serving。

在创建ASK集群时部署Knative

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**ASK集群**页签，以下为重点关注参数，详细参数配置请参见创建Serverless Kubernetes集群。

参数	配置
Kubernetes版本	选择1.15及以上版本。
NAT网关	设置是否为专有网络创建NAT网关并配置SNAT规则。仅当**专有网络**选择为**自动创建**时，需要设置该选项。 ○ 若您选择自动创建VPC，可选择是否自动配置SNAT网关。 ○ 若选择不自动配置SNAT，您可自行配置NAT网关实现VPC安全访问公网环境，并且手动配置SNAT，否则VPC内实例将不能正常访问公网。
公网访问	设置是否开放**使用EIP暴露API Server**。 ○ 如果选择开放，将创建一个EIP，同时会暴露Master节点的6443端口（对应API Server），您可以在外网通过kubeconfig连接或操作集群。 ○ 若选择不开放，不会创建EIP，您只能在VPC内部用kubeconfig连接和操作集群。
Knative	选中开启**Knative**，将在创建ASK集群时部署Knative。 

5. 在页面右侧，单击**创建集群**，在弹出的**当前配置确认**页面，单击**确定**，启动部署。

在ASK集群管理中部署Knative

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Knative的**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组件用于管理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。

结果验证

Knative部署完成后，在**Knative组件管理**页面可以看到Knative组件显示**已安装**。

16.4.2. 卸载Knative

本文介绍如何卸载Knative。

前提条件

开启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Knative的**组件管理**页签单击右上角的**一键卸载**。
6. 在**卸载Knative**页面，选中**我已知晓并确认卸载Knative**，单击**确认**。

结果验证

卸载完成后，可以在**Knative组件管理**页面看到Knative卸载成功的信息。

16.5. Knative服务管理

16.5.1. 快速部署Serverless应用

本文以Hello World示例为您介绍如何通过Knative快速部署一个Serverless应用。

前提条件

- 创建Serverless Kubernetes集群。
- 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**服务管理**页签右上角，单击**创建服务**。
6. 设置**集群**、**命名空间**、**服务名称**，选择所要使用的镜像和镜像版本等配置信息。

参数	描述
服务名称	自定义该服务的名称。本例为*helloworld-go*。
镜像名称	您可以单击**选择镜像**，在弹出的对话框中选择所需的镜像并单击**确定**。您还可以填写私有registry。填写的格式为*domainname/namespace/imagename:tag*。本例中为*registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go*。
镜像版本	您可以单击**选择镜像版本**。若不指定，默认为latest。本例中为*73fbdd56*。
环境变量	支持通过键值对的形式配置环境变量。本例中，`TARGET=Knative`。

界面其他参数详细信息请参见参数说明。

7. 单击**创建**。

   创建完成后，您可以在**服务管理**页签的列表中，看到新创建的服务。

服务访问

服务部署完成后，通过绑定Host域名与访问网关，然后可以直接访问服务URL。

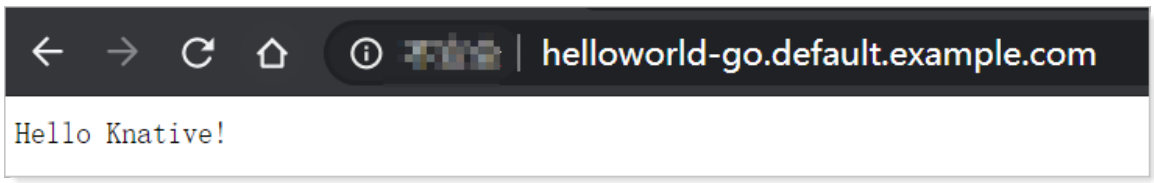
1. 在Kubernetes菜单下，单击左侧导航栏的**Knative > 组件管理**，进入**组件管理**页面。可以看到访问网关。
2. 将访问网关地址与需要访问的域名进行Host绑定，在Hosts文件中添加绑定信息，具体格式如下。

   网关+ 域名

   样例如下：

   ```
   47.95.XX.XX helloworld-go.default.example.com
   ```

3. 完成Host绑定后，可通过域名*http://helloworld-go.default.example.com*直接对服务进行访问。



16.5.2. Knative Gateway

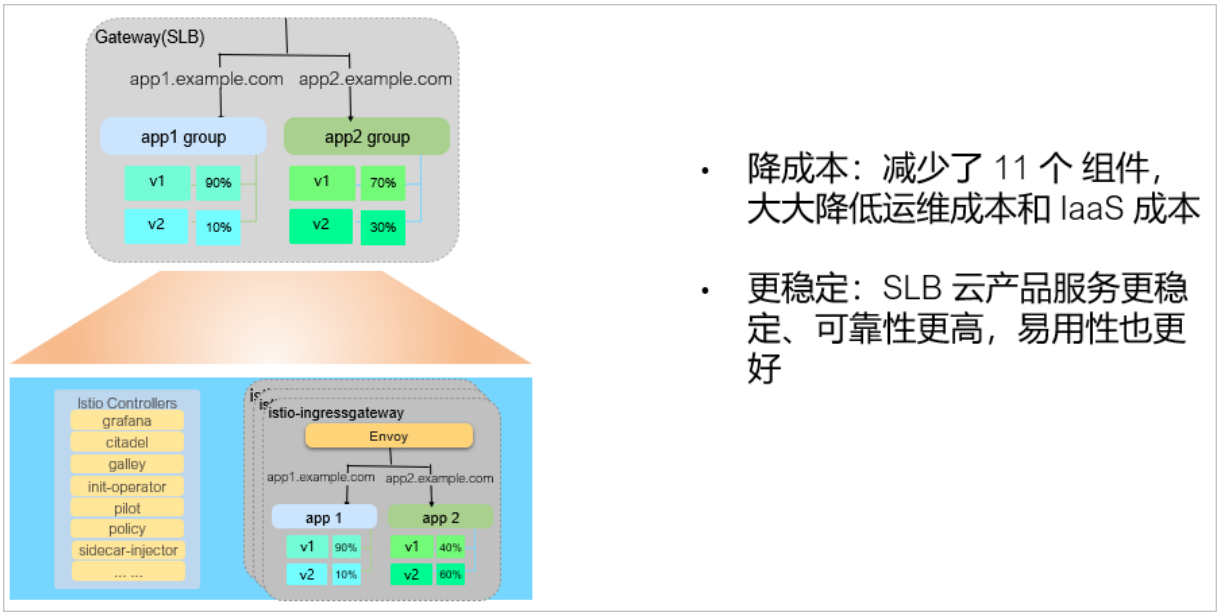
Kative Gateway具备多种Gateway实现方案，并且属于云产品级别的支撑，不需要常驻资源。不仅可以帮助您节省IaaS成本还省去了很多运维负担。本文介绍Knative Gateway的优势和使用方法。

Knative Gateway优势

社区Knative默认支持Istio、Gloo、Contour、Kourier和Ambassador等多种Gateway实现方案。在这些实现方案中Istio使用频率最高。因为Istio除了可以充当Gateway的角色还能作为ServiceMesh服务使用。Serverless服务首先需要有Gateway实例常驻运行，而且为了保证高可用至少要有两个实例互为备份。其次这些Gateway的K8s Controller也需要常驻运行，这些常驻实例的IaaS费用和运维都是业务需要支付的成本。

ASK Knative使用SLB作为Gateway，提供内网和外网两种类型的网关。除了HTTP以外，Gateway还提供HTTPS功能。Knative默认给Gateway生成一个自签名的HTTPS证书，没有域名限制，可用于测试。在使用Knative部署线上服务时，您需要创建HTTPS证书，并将Knative Gateway Service的证书ID修改为创建的证书ID。更多信息，请参见配置HTTPS证书。

为了极致的Serverless体验，阿里云通过SLB实现了Knative Gateway。



使用Knative Gateway

1. 执行以下命令，获取SLB的IP地址。

   ```
   kubectl -n knative-serving get svc
   ```

   预期输出：

   ```
   NAME              TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
   ingress-gateway       LoadBalancer   172.21.XX.XX   8.131.XX.XX     80:32701/TCP,443:30561/TCP   2d20h
   ingress-local-gateway   LoadBalancer   172.21.XX.XX   192.168.XX.XX   80:32537/TCP                 2d20h
   ```

   > 说明
   > - ingress-gateway表示外网网关，适用于将服务暴露到外部提供访问。
   > - ingress-local-gateway表示内网网关，适用于VPC内部服务访问。

2. 访问服务。
   - 外网服务访问
     - 通过HTTP访问服务。

       执行以下命令：

       ```
       curl -H "Host: helloworld-go.default.example.com" http://8.131.XX.XX
       ```

       预期输出：

       ```
       Hello Knative!
       ```

- 通过HTTPS访问服务。

  执行以下命令：

  ```
  curl -H "Host: helloworld-go.default.example.com" https://8.131.XX.XX -k
  ```

  预期输出：

  ```
  Hello Knative!
  ```

- VPC内服务访问

  > 说明 需要先开启PrivateZone。

  a. 执行以下命令，编辑*eci-profile*文件。

  ```
  kubectl -n kube-system edit configmap eci-profile
  ```

  b. 修改 `enablePrivateZone` 参数值为 `true` ，保存并退出*eci-profile*文件，开启PrivateZone。

  ```
  apiVersion: v1
  data:
   ...
   enablePrivateZone: "true"
   ...
  kind: ConfigMap
  metadata:
   name: eci-profile
   namespace: kube-system
  ```

  c. 在服务内部直接通过 `服务名.命名空间.svc.cluster.local` 调用服务。

  以调用Default命名空间下helloworld-go服务为例：

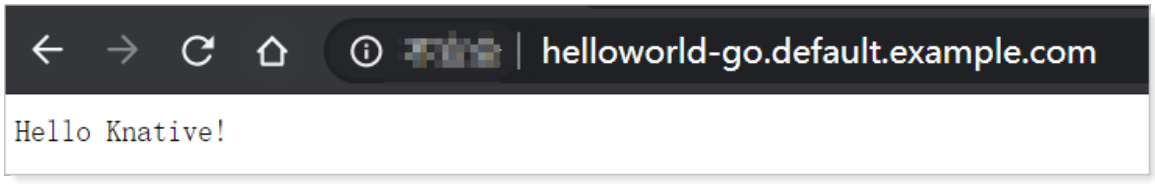
  ```
  http://helloworld-go.default.svc.cluster.local
  ```

3. 将SLB的IP地址与Knative域名进行Host绑定，在Hosts文件中添加绑定信息。以下为绑定样例。

   > 说明 Knative默认的路由根域名是example.com，您可以自定义域名。更多信息，请参见在Knative使用自定义域名。

   ```
   106.15.2**.** helloworld-go.default.example.com
   ```

   完成Hosts绑定后，如果您可以通过域名访问服务，表示Knative Gateway可以正常使用。



16.5.3. 添加自定义路由

如果Knative Service需要配置多个不同的域名，您可以通过 knative.aliyun.com/serving-ingress Annotation直接指定自定义的域名和路径，并且通过自定义域名和路径访问Knative Service。本文介绍如何添加自定义的域名和路径，并通过自定义域名和路径访问Knative Service。

背景信息

Knative的配置中有一个默认的主域名，每一个Knative Service都基于主域名、命名空间和Service名称生成一个唯一域名。域名生成规则为 {ksvc-name}.{namespace}.{knative-default-domain} 。Knaive默认的主域名是 example.com ，例如，一个名为Coffee的knative Service部署在Default命名空间中，默认的唯一域名是 coffee.default.example.com 。您可以修改默认的 example.com 域名，详细介绍请参见在Knative使用自定义域名。

如果Knative Service需要配置多个不同的域名，您可以通过 knative.aliyun.com/serving-ingress Annotation直接指定自定义的域名和路径。例如， knative.aliyun.com/serving-ingress: cafe.mydomain.com/coffee 表示指定cafe.mydomain.com域名和*/coffee*路径。

> **说明** 配置一个自定义域名和路径并不会替换默认的域名，而是在默认域名的基础之上增加一个新的域名。

操作步骤

1. 创建并拷贝以下内容到*ingress-domain.yaml*。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: coffee-mydomain
 annotations:
  knative.aliyun.com/serving-ingress: cafe.mydomain.com/coffee
spec:
 template:
   spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go:160e4dc8
```

2. 创建Knative Service。

```
kubectl apply -f ingress-domain.yaml
```

3. 查看Knative Service信息。

   可以看到coffee-mydomain.default.example.com是当前Knative Service的默认域名。

```
kubectl get ksvc
```

   预期输出：

```
NAME        URL                                        LATESTCREATED      LATESTREADY        READY  REASON
coffee-mydomain  http://coffee-mydomain.default.example.com  coffee-mydomain-sbwhz  coffee-mydomain-sbwhz  True
```

4. 通过默认域名访问Knative Service。

```
curl -H "Host: coffee-mydomain.default.example.com" http://106.15.2**.**
```

   预期输出：

```
Hello World!
```

5. 通过自定义域名和路径访问Knative Service。

   ```
   curl -H "Host: cafe.mydomain.com" http://106.15.2**.**/coffee
   ```

   预期输出：

   ```
   Hello World!
   ```

16.5.4. 配置HTTPS证书

ASK Knative使用SLB作为Gateway。除了HTTP以外，Gateway还提供HTTPS功能。Knative默认给Gateway生成一个自签名的HTTPS证书，没有域名限制，可用于测试。在使用Knative部署线上服务时，您需要创建HTTPS证书，并将Knative Gateway Service的证书ID修改为创建的证书ID。本文介绍如何查看、创建和使用HTTPS证书。

Gateway的详细介绍请参见Knative Gateway。

查看默认证书

1. 登录传统型负载均衡CLB控制台。
2. 在左侧导航栏，选择**传统型负载均衡 CLB（原SLB） > 证书管理**。
3. 在**证书管理**页面找到knative-default-gateway-cert，knative-default-gateway-cert就是Knative的默认证书。默认证书由Knative自动生成，可用于测试。

创建证书

您可以通过使用阿里云签发的证书或上传非阿里云签发的证书的方式创建证书，详细介绍请参见选择阿里云签发证书和上传非阿里云签发证书。

使用创建的证书

1. 登录传统型负载均衡CLB控制台。
2. 在左侧导航栏，选择**传统型负载均衡 CLB（原SLB） > 证书管理**。
3. 在**证书管理**页面鼠标悬停至目标证书ID，单击目标证书**证书名称/ID**列的，复制证书ID。
4. 将Knative Gateway Service的 `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id` 的值修改为步骤中复制的证书ID。通过这个Annotation指定证书ID后，您就可以使用创建的证书提供服务。

```
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
spec:
 ports:
 - port: 443
  protocol: TCP
  targetPort: 80
 selector:
  run: nginx
 type: LoadBalancer
```

管理多域名证书

阿里云SLB支持配置多个证书，每一个证书可以服务一组域名，您可以配置多组证书和域名。

> **说明** 更多域名管理操作请参见添加扩展域名。

1. 登录传统型负载均衡CLB控制台。
2.
3. 在**实例管理**页面，单击目标实例ID。
4. 在**监听**页签找到已创建的HTTPS监听，选择操作列下的 > **扩展域名管理**。

> **说明** 在**监听**页签监听的**前端协议/端口**列显示HTTPS，则表示该监听是HTTPS监听。

5. 在**扩展域名管理**页面，单击**添加扩展域名**，输入扩展域名，选择服务器证书。

   域名只能由字母、数字、连字符（-）和点（.）组成，首位必须是字母或数字。合法域名检测，请参见阿里云域名检测工具。

6. 单击**确定**。

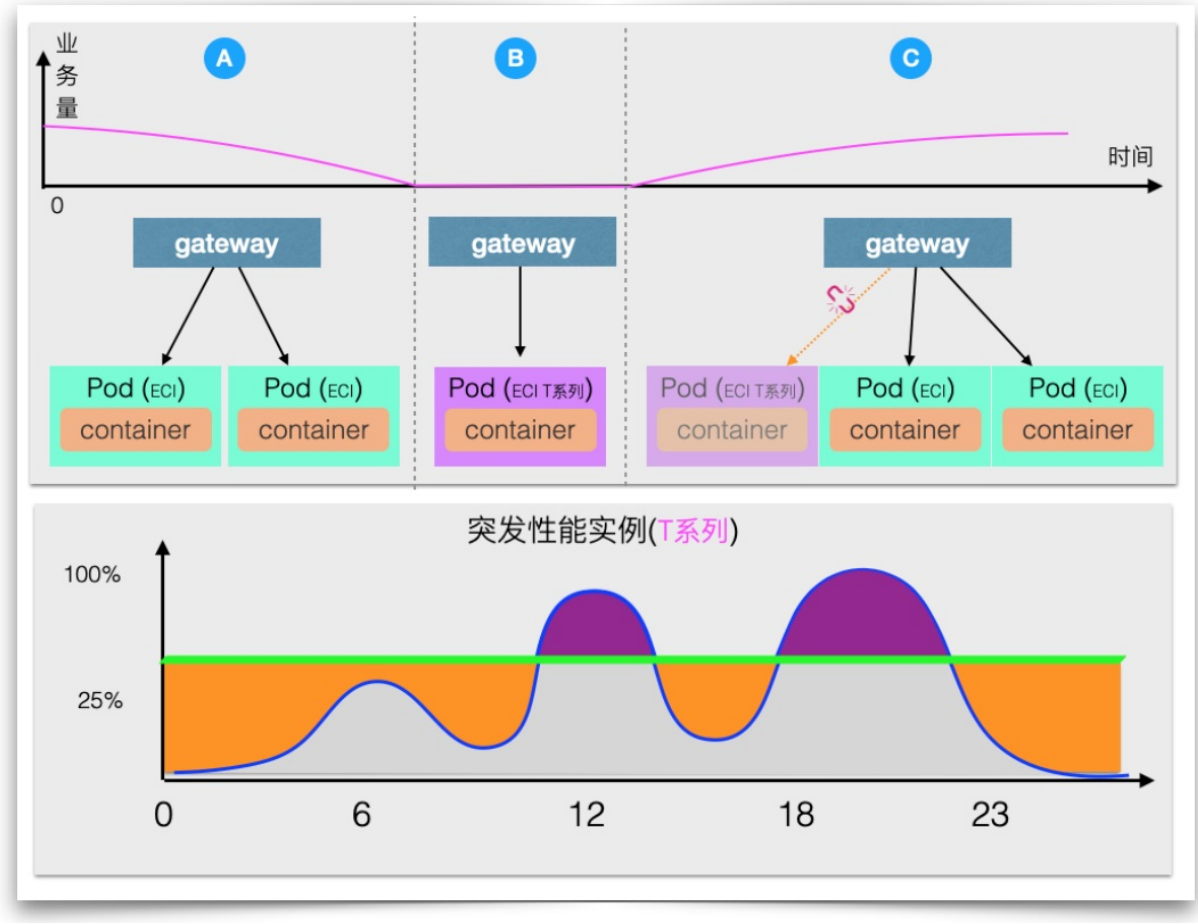
16.5.5. 保留实例

保留实例是ASK Knative独有的功能。如果想通过Knative降低成本而不引入冷启动，可以通过低成本的保留实例来平衡成本和冷启动时长。

保留实例优势

社区的Knative默认在没有流量的时候可以把应用实例数缩容到零个，这样做的目的是降低常驻实例的运行成本。缩容到零的方式虽然降低了常驻成本，但从零到一会有一个明显的冷启动过程，这个冷启动过程会有明显的延时。分配一个新的实例首先需要分配IaaS资源，这就涉及到Kubernetes的调度。除了调度还有拉取应用镜像、启动应用等环节。而应用镜像大小以及应用启动时长与具体的开发者或者业务有很强的关联，这在通用的平台层面几乎无法控制。

如果想通过Knative降低成本而不引入冷启动，可以通过低成本的保留实例来平衡成本和冷启动时长。ASK的Knative和社区Knative不同点在于默认在没有流量的时候不把应用实例数缩容到零个，而是保留一个实例。保留的实例可以和默认规格设置不一样的配置。例如，使用比默认规格价格更低的规格，从而平衡成本和效率。



保留实例策略

ASK Knative的策略是在业务波谷时使用保留实例替换默认的计算型实例，当第一个请求来临时使用保留实例提供服务，同时也会触发默认规格实例的扩容。当默认规格实例扩容完成以后所有新请求就会都转发到默认规格上，同时保留实例则不会接受新的请求，并且等保留实例所有接收到的请求处理完成以后就会被下线。通过这种无缝替换的方式实现了成本和效率的平衡，即降低了常驻实例的成本又不会有显著的冷启动时长。

保留实例规格

阿里云的ECI有一种规格族叫做突发性能实例。突发性能实例是一种通过CPU积分来保证计算性能的实例规格，适用于平时CPU使用率低，但偶尔有突发高CPU使用率的场景。突发性能实例在创建后可以持续获得CPU积分，在性能无法满足负载要求时，通过消耗更多CPU积分来无缝提高计算性能，不会影响部署在实例上的环境和应用。较之其他实例规格，突发性能实例的CPU使用更加灵活且成本较低。通过CPU积分，您可以从整体业务角度分配计算资源，将业务平峰期的计算能力转移到高峰期使用，以节约使用成本。

突发性能的低成本、CPU积分等特性适合作为保留实例。ecs.t6-c1m1.large，ecs.t5-lc1m2.small，ecs.s6-c1m2.small，ecs.t6-c1m2.large，ecs.n1.small为ASK Knative默认的保留实例的规格配置，是满足1核2G要求的价格最低的前5个规格。您可以到云产品价格页面获取所有ECI的规格和价格信息。

> **说明** ECI最多只能同时配置5个规格。

您可以通过 `knative.aliyun.com/reserve-instance-eci-use-specs` Annotation手动配置保留实例的规格。有两种配置规格的方法：

- 指定ECI规格列表，以下以指定ecs.t6-c1m1.large，ecs.t5-lc1m2.small规格为例。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: hello-spec-1
spec:
 template:
  metadata:
   annotations:
    knative.aliyun.com/reserve-instance-eci-use-specs: "ecs.t6-c1m1.large,ecs.t5-lc1m2.small"
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go:160e4dc8
```

- 指定CPU和内存的规格，Knative会自动根据ECS的价格从小到大搜索满足条件的规格，然后使用价格最低的5个规格创建实例。以下以指定1核2G为例，`1-2Gi` 表示指定1核2G。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: hello-spec-2
spec:
 template:
  metadata:
   annotations:
    knative.aliyun.com/reserve-instance-eci-use-specs: "1-2Gi"
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go:160e4dc8
```

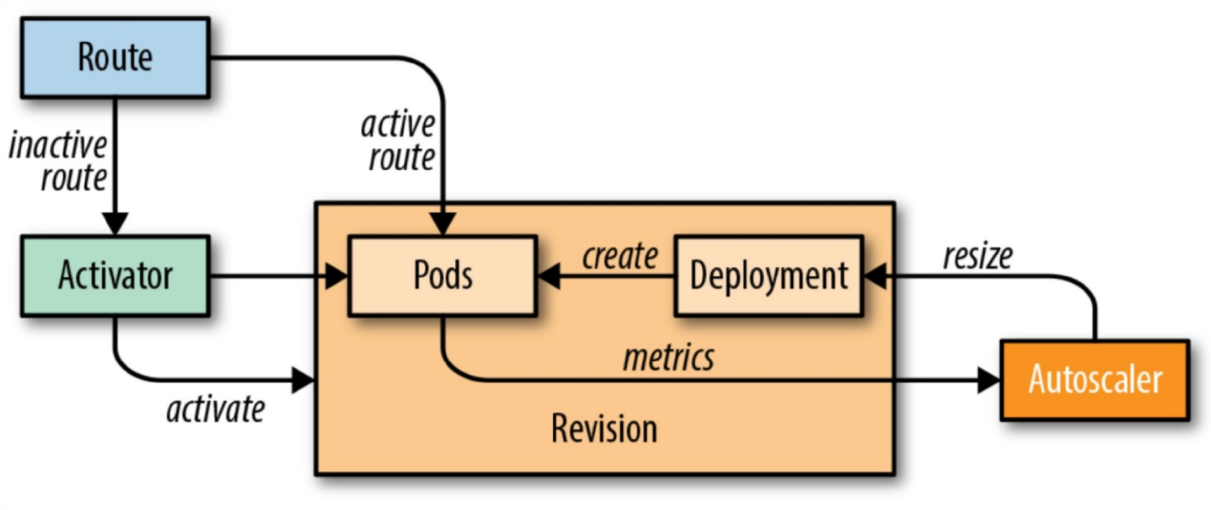
这两种方式Knative都是使用一个ECI规格列表去申请ECI实例的，多个规格之间是有顺序的，当前面的规格库存不足时，后面的规格将自动申请实例。

16.5.6. 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
    ```

  - containerConcurrency

    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. 开启Knative。
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，获取访问网关。

   ```
   kubectl -n knative-serving get svc
   ```

   预期输出：

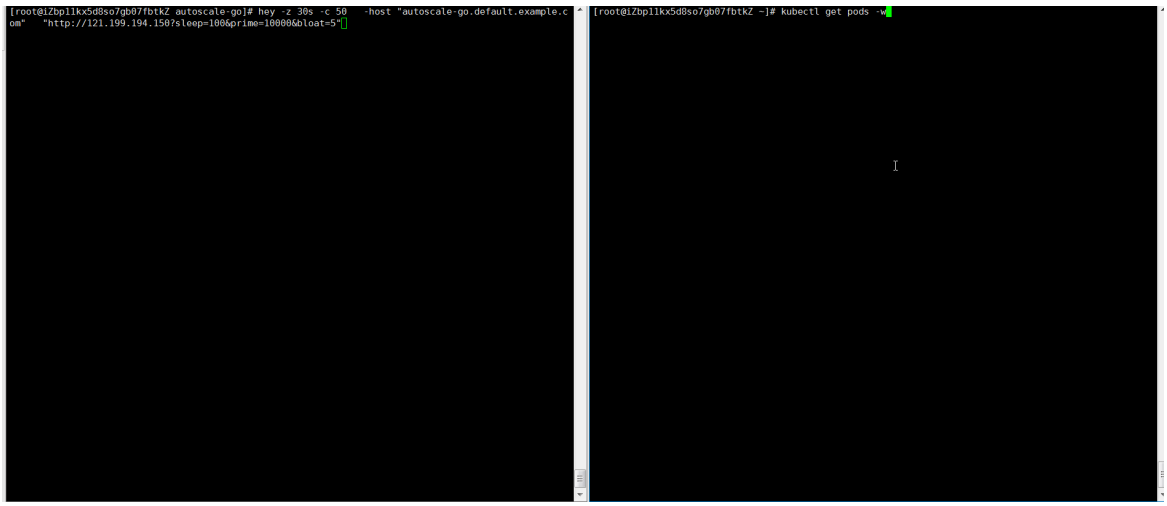
   ```
   NAME        TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)           AGE
   ingress-gateway  LoadBalancer  172.19.1*.***  121.199.19*.***  80:32185/TCP,443:31137/TCP  69d
   ```

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. 开启Knative。
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。

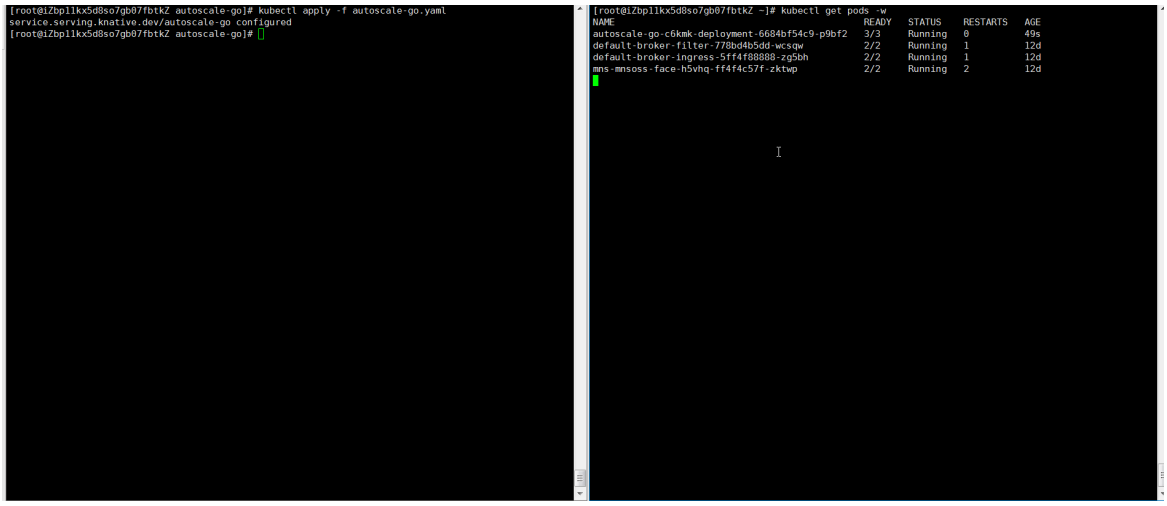
```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
 name: autoscale-go
 namespace: default
spec:
 template:
  metadata:
   labels:
    app: autoscale-go
   annotations:
    autoscaling.knative.dev/target: "10"
    autoscaling.knative.dev/minScale: "1"
    autoscaling.knative.dev/maxScale: "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。

16.5.7. 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

开启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'
```

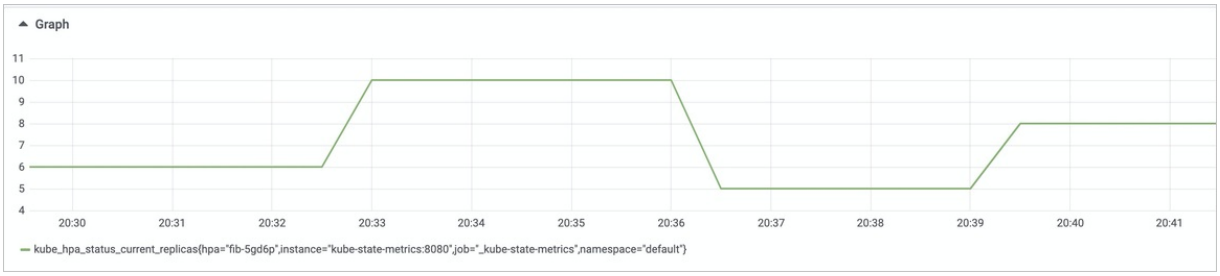
- 通过 `autoscaling.knative.dev/class: "hpa.autoscaling.knative.dev"` 指定HPA弹性插件。
- 通过 `autoscaling.knative.dev/metric` 设置HPA CPU指标。
- 通过 `autoscaling.knative.dev/target` 设置HPA CPU指标的阈值。
- 通过 `autoscaling.knative.dev/minScale: "1"` 设置弹性策略实例数的最小值。
- 通过 `autoscaling.knative.dev/maxScale: "10"` 设置弹性策略实例数的最大值。

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。



16.5.8. 在Knative上设置定时弹性

ASK支持定时弹性功能，可以通过定时弹性提前规划资源数量，满足周期性的设置弹性资源的诉求。除此之外，Knative定时弹性融合了HPA的弹性能力，对于突发高负载的场景下，依然能够保证弹性扩容足够的资源提供服务。本文介绍如何在Knative上设置定时弹性。

前提条件

开启Knative

操作步骤

1. 设置定时策略。

   通过ConfigMap配置定时弹性策略，每个Knative Service对应一个定时弹性策略，以下为配置示例。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: cron-autoscaler
 namespace: default
data:
 jobs: |
  - name: "workday"
    schedule: "* * 1-5"
    timeseries:
    - timeSlice: 01:30:30
      replicas: 1
    - timeSlice: 05:30:30
      replicas: 2
    - timeSlice: 07:30:30
      replicas: 5
    - timeSlice: 10:24:30
      replicas: 8
    - timeSlice: 11:47:30
      replicas: 10
    - timeSlice: 13:17:30
      replicas: 6
    - timeSlice: 16:50:30
      replicas: 9
    - timeSlice: 20:17:30
      replicas: 5
    - timeSlice: 23:30:30
      replicas: 1
  - name: "holiday"
    schedule: "* * 0,6"
    timeseries:
    - timeSlice: 08:24:30
      replicas: 4
    - timeSlice: 11:47:30
      replicas: 3
    - timeSlice: 13:17:30
      replicas: 2
```

   - `schedule` 字段 `* * 1-5` ：第一个参数表示一个月份中的第几日，第二个参数表示月份，第三个表示一个星期中的第几天。1-5表示周一到周五，0,6表示周六、周日。
   - `timeseries` 字段格式（24时制）： `小时:分钟:秒` 。

> **说明**
> - 如果更新该策略，引用该策略的所有Knative Service都会更新。
> - 如果删除该策略，现有引用该策略的Knative Service保持当前的状态，不会缩容到0。

2. 使用定时弹性策略。

   配置完弹性策略之后，您可以通过Knative Service指定使用该定时弹性策略。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: helloworld-go-ppa
 annotations:
  alicloud.autoscaling.service.knative.dev/timeseries: "cron-autoscaler"
spec:
 template:
  metadata:
   labels:
    app: helloworld-go-ppa
   annotations:
    autoscaling.knative.dev/class: "ppa.autoscaling.knative.dev"
    alicloud.autoscaling.knative.dev/metric-target: "cpu:50"
  spec:
   containers:
    - image: registry.cn-hangzhou.aliyuncs.com/knative-samples/helloworld-go:160e4dc8
      resources:
       requests:
        cpu: '200m'
```

   - 通过 `alicloud.autoscaling.service.knative.dev/timeseries` 指定弹性策略名称。
   - 通过 `autoscaling.knative.dev/class: "ppa.autoscaling.knative.dev"` 指定定时弹性插件。
   - 多指标支持。社区Knative仅支持同时设置一个指标，通过 `alicloud.autoscaling.knative.dev/metric-target` 可以设置多个HPA弹性指标。例如同时支持CPU和内存指标，则设置为 `cpu:50,memory:50` 。
   - 通过 `autoscaling.knative.dev/minScale: "1"` 设置弹性策略最小值。
   - 通过 `autoscaling.knative.dev/maxScale: "10"` 设置弹性策略最大值。

   > **说明** 以下为弹性策略最小值（ `minScale` ）、最大值（ `maxScale` ）和定时弹性值之间的关系：
   > - HPA最小值=Min（minScale, 定时弹性值）
   > - HPA最大值=Max（maxScale, 定时弹性值）

结果验证

开启定时弹性后，实例变化趋势如下图所示。

16.5.9. 在Knative中使用函数部署服务

将Knative与函数计算FC（Function Compute）结合，使用函数部署服务，支持根据代码、OSS文件和镜像部署服务。本文介绍如何在Knative中使用函数部署服务。

前提条件

- ASK使用快速入门
- 开启Knative

使用代码部署服务

1. 创建*coffee.yaml*。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: coffee
 annotations:
  workload.serving.knative.aliyun.com/class: "fc"
spec:
 template:
  metadata:
   annotations:
    fc.revision.serving.knative.aliyun.com/code-space: "inline"
  spec:
   containers:
    - image: nodejs8
     command:
     - |
      var getRawBody = require('raw-body')
      var version = 'coffee-default'
      module.exports.handler = function (request, response, context) {
        var respBody = new Buffer('Hello ' + version + '\n')
        response.setStatusCode(200)
        response.setHeader('content-type', 'application/json')
        response.send(respBody)
      };
```

`image` 和 `command` 下的内容代表使用Node.js代码部署服务，不同的服务代码内容不同，请根据实际情况替换。

2. 部署服务。

```
kubectl apply -f coffee.yaml
```

基于函数计算OSS文件部署服务

使用函数计算在对象存储OSS上下载函数代码，部署服务。

1. 创建*coffee-oss.yaml*。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: coffee
 annotations:
  workload.serving.knative.aliyun.com/class: "fc"
spec:
 template:
  metadata:
   annotations:
    fc.revision.serving.knative.aliyun.com/code-space: "oss"
  spec:
   containers:
    - image: nodejs8
      command:
      - fc-zip
      - knative-demo/node-demo.zip
```

`image` 和 `command` 下的内容表示使用函数计算OSS文件部署服务。以下为 `image` 和 `command` 参数介绍：

   - `image`：输入 `runtime` 环境，例如 `nodejs8` 。
   - `command`：`command` 数组有两个元素，依次是Bucket Name和Object Name。

2. 部署服务。

```
kubectl apply -f coffee-oss.yaml
```

基于函数计算镜像部署服务

为了简化开发者体验、提升开发和交付效率，函数计算提供了Custom Container Runtime。通过HTTP协议和函数计算系统交互，开发者将容器镜像作为函数的交付物。关于函数计算镜像部署的详细介绍，请参见简介。

使用限制

配置项	使用限制
镜像大小	• 若您的函数执行内存<1 GB，则解压前镜像大小不能超过256 MB。 • 若您的函数执行内存≥1 GB，则解压前镜像大小不能超过1024 MB。
镜像仓库	目前仅支持阿里云容器镜像服务中的默认实例镜像，未来会支持其他镜像仓库。关于默认实例的详细介绍，请参见什么是容器镜像服务ACR。
镜像访问	目前仅支持同账号同地域下私有镜像仓库读取，未来会支持读取公共镜像。

配置项	使用限制
容器内文件读写权限	容器run-as-user UID在10000到10999中随机分配。默认容器可在*/tmp*目录中具有写入权限，其他目录读写权限由镜像文件系统控制。如果运行的UID没有读写某个文件的权限，则需要在Dockerfile中修改。
容器可写层存储空间限制	排除只读镜像层，容器产生的数据最大不能超过512 MB。
权限策略	创建目标服务时需要绑定权限策略AliyunContainerRegistryReadOnlyAccess或者AliyunContainerRegistryFullAccess。您需要创建RAM角色，并绑定权限策略AliyunContainerRegistryReadOnlyAccess或者AliyunContainerRegistryFullAccess。然后在RAM角色管理页**基本信息**区域获取ARN。具体操作，请参见为RAM角色授权和RAM角色概览。

1. 创建*coffee-image.yaml*。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: coffee
 annotations:
  workload.serving.knative.aliyun.com/class: "fc"
spec:
 template:
  metadata:
   annotations:
    fc.revision.serving.knative.aliyun.com/code-space: "image"
    fc.revision.serving.knative.aliyun.com/role-arm: "acs:ram::1041208914252405:role/fc-yuanyi-test"
  spec:
   containers:
    - image: registry.cn-shenzhen.aliyuncs.com/aliknative/nodejs-express:v3.0
```

   `annotations`和`image`下的内容代表基于函数镜像部署服务，以下为参数解释：
   - fc.revision.serving.knative.aliyun.com/code-space: image：指定镜像类型。
   - fc.revision.serving.knative.aliyun.com/role-arm：输入ARN绑定权限策略AliyunContainerRegistryReadOnlyAccess或者AliyunContainerRegistryFullAccess。
   - image：输入镜像名称。
2. 部署服务。

```
kubectl apply -f coffee-image.yaml
```

结果验证

验证部署服务是否成功，以下以使用代码部署的服务为例。

1. 查看服务部署状态。

```
kubectl get ksvc
```

   预期输出：

```
NAME   URL                                                                    LATESTCREATED  LATESTREADY   READY
REASON
coffee  https://198639303048****.cn-beijing.fc.aliyuncs.com/2016-08-15/proxy/kn_default_coffee.http-
prd/kn_default_coffee/  coffee-5bqdr   coffee-5bqdr   True
```

2. 使用 `curl` 命令访问服务。

```
curl https://198639303048****.cn-beijing.fc.aliyuncs.com/2016-08-15/proxy/kn_default_coffee.http-prd
/kn_default_coffee/
```

预期输出：

```
Hello coffee-default
```

3. 在函数计算控制台查看服务和函数。
   i. 登录函数计算控制台。
   ii. 在顶部菜单栏，选择地域。
   iii. 在左侧导航栏中，单击**服务及函数**。在**服务及函数**页面可以看到kn_default_coffee服务。
   iv. 单击kn_default_coffee，在**函数列表**页签下可以看到kn_default_coffee函数。

16.5.10. 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*。

参数	说明
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：

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。

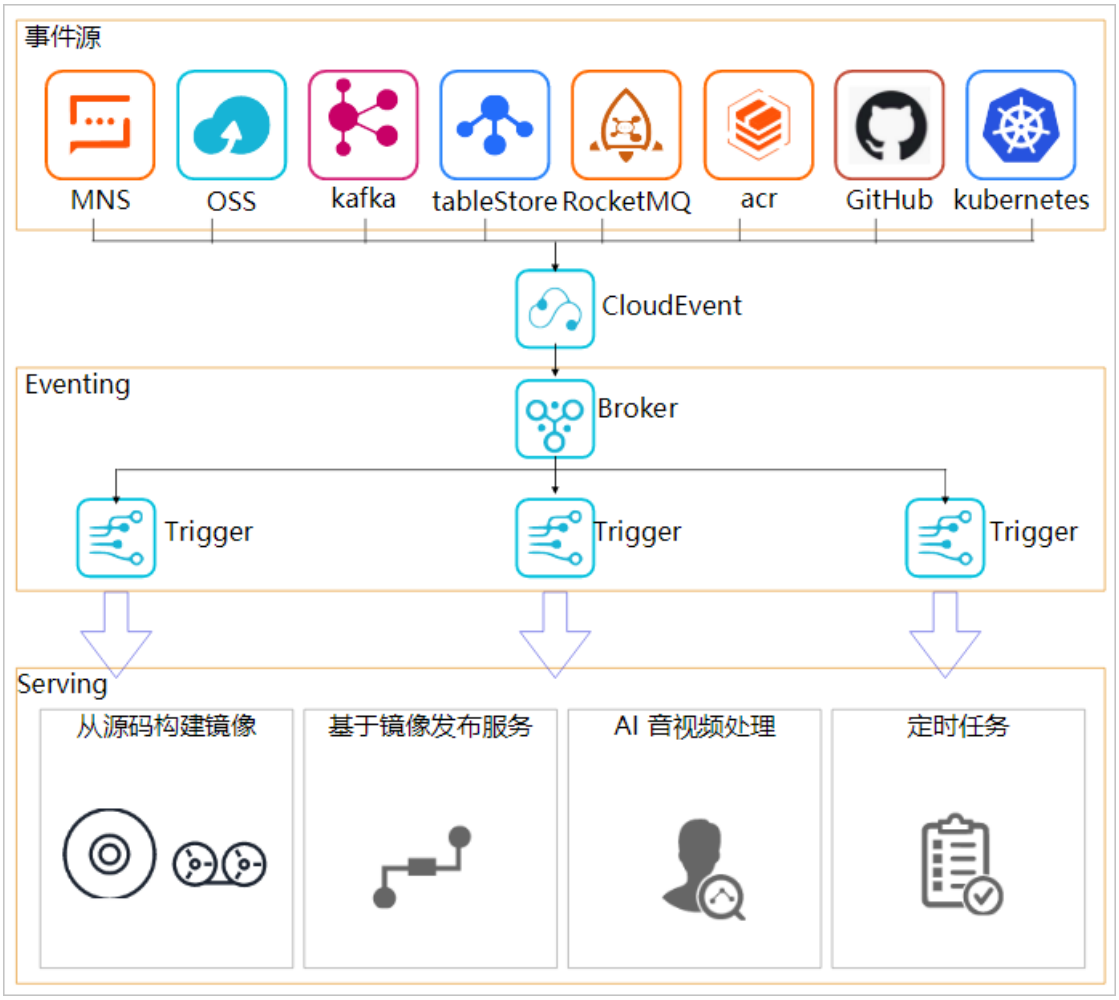
16.6. Knative事件驱动

16.6.1. 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，请见部署Eventing。

相关文档

- 部署Eventing

16.6.2. 部署Eventing

ASK Knative中提供了事件驱动框架Eventing。Eventing组件针对Serverless事件驱动模式做了一套完整的设计，提供了事件的接入、触发等一整套事件管理的能力。本文介绍如何在ASK Knative中部署Eventing。

前提条件

- ASK使用快速入门
- 开启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**核心组件**区域单击Eventing**操作**列的**部署**。
6. 在**部署Eventing**对话框中提示**当前集群未开启PrivateZone，请先开启PrivateZone**，请按以下步骤操作，开启PrivateZone。

   > 说明　在部署Eventing对话框中未提示**当前集群未开启PrivateZone，请先开启PrivateZone**，可跳过此步骤，直接执行步骤。

   i. 执行以下命令，编辑*eci-profile*文件。

   ```
   kubectl -n kube-system edit configmap eci-profile
   ```

   ii. 修改 `enablePrivateZone` 参数值为 `true` ，保存并退出*eci-profile*文件。

   ```
   apiVersion: v1
   data:
    ...
    enablePrivateZone: "true"
    ...
   kind: ConfigMap
   metadata:
    name: eci-profile
    namespace: kube-system
   ```

7. 在**部署Eventing**对话框中单击**确定**。

执行结果

部署Eventing完成后，在**组件管理**页签，可以看到Eventing组件状态为**已部署**。

16.7. Knative最佳实践

16.7.1. 在Knative上观测服务的QPS、RT和Pod扩缩容趋势

在服务运行过程中，您可以在Knative上观测当前服务运行的状况，包括每秒查询数QPS（Query per Second）、平均响应时RT（Response Time）以及Pod扩缩容趋势。本文主要介绍如何配置和观测这些信息。

前提条件

阿里云Serverless Kubernetes（ASK）集群已开通日志服务和Knative功能。更多信息，请参见开启Knative。

创建并配置Logstore

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标ASK集群对应的日志服务Project名称。
3. 在**日志存储**页面的**日志库**页签，单击**+**图标。
4. 在**创建Logstore**面板中配置好相关参数后，单击**确定**。更多关于**创建Logstore**参数的相关描述，请参见管理Logstore。
5. 在**创建成功**对话框中，单击**数据接入导向**。
6. 在**数据接入**区域，单击**Docker标准输出**。

7. 在**创建机器组**步骤，单击**使用现有机器组**。
8. 在**机器组配置**步骤，选择已有机器组后，单击**下一步**。
9. 在**数据源设置**步骤，使用以下模板样例设置日志服务的采集日志规则后，单击**下一步**。

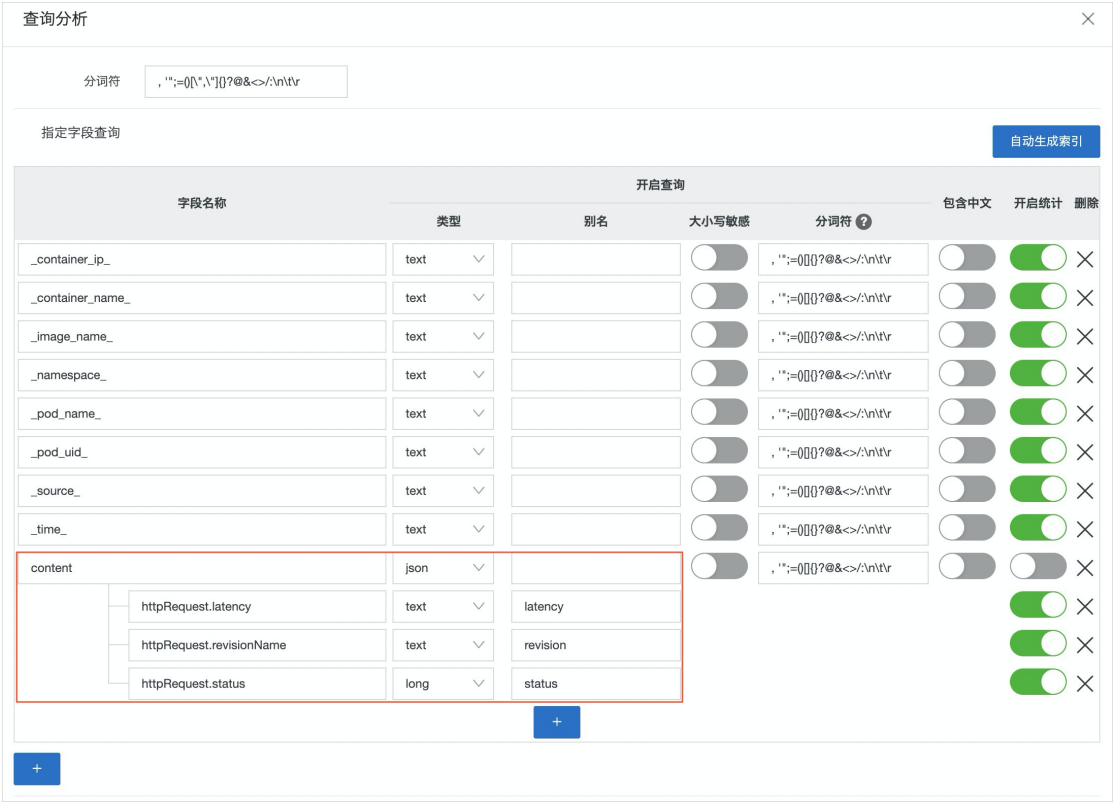
```
{
  "inputs": [
    {
      "detail": {
        "IncludeEnv": {
          "QUEUE_SERVING_PORT": "8012"
        },
        "ExcludeLabel": {}
      },
      "type": "service_docker_stdout"
    }
  ]
}
```

10. 在**查询分析配置**步骤，保留选择默认勾选项，单击**下一步**。
11. 在**结束**步骤页签的**查询日志**区域，单击**立即查询**。
    i. 单击**查询分析**观察日志输出。

    ii. 从**查询分析属性**列表中选择**属性**。

iii. 在**查询分析**面板单击**自动生成索引**，在弹出的**自动生成索引属性**对话框中，单击**追加**，然后在**content**字段中添加以下参数。

参数名称	类型	别名
httpRequest.latency	text	latency
httpRequest.status	long	status
httpRequest.revisionName	text	revision

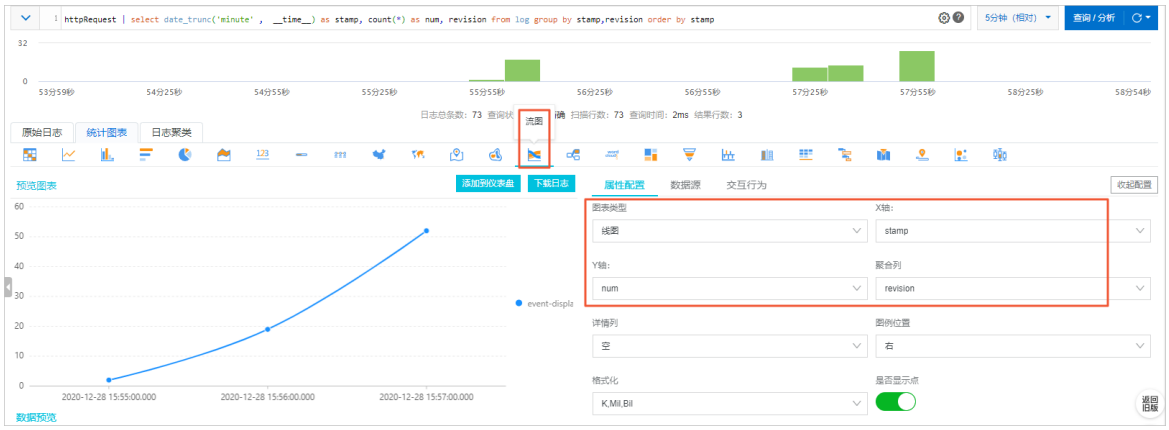


配置QPS统计

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标ASK集群对应的日志服务Project名称。
3. 在**日志存储**页面的**日志库**的页签，单击目标Logstore。
4. 在**查询/分析**文本框中输入以下SQL查询命令。


```
httpRequest | select date_trunc('minute', __time__) as stamp, count(*) as num, revision from log grou
p by stamp,revision order by stamp
```

5. 单击**流图**，进行**属性配置**之后，单击**添加到仪表盘**。

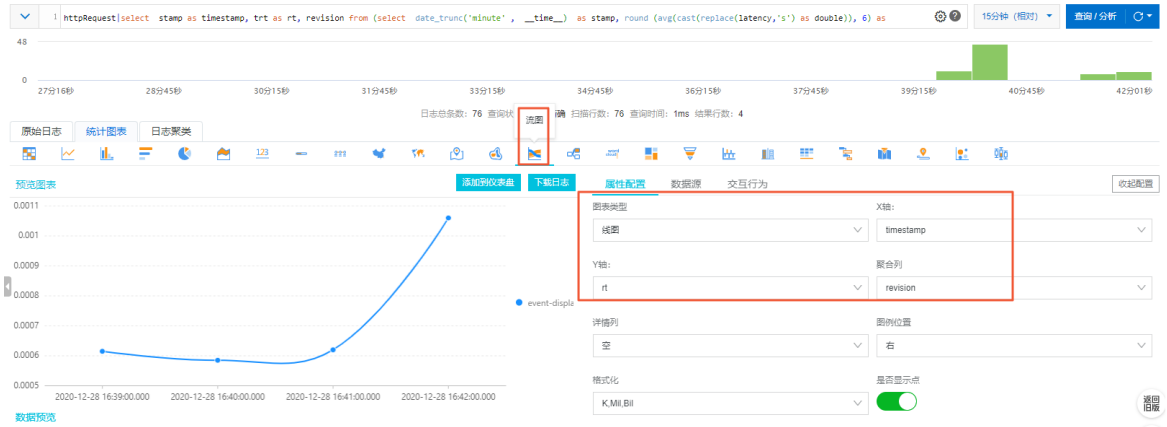


配置RT统计

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标ASK集群对应的日志服务Project名称。
3. 在**日志存储**页面的**日志库**的页签，单击目标Logstore。
4. 在**查询/分析**文本框中输入以下SQL查询命令。

```
httpRequest|select  stamp as timestamp, trt as rt, revision from (select  date_trunc('minute' , __time__ ) as stamp, round (avg(cast(replace(latency,'s') as double)), 6) as trt, revision  from log  group by  stamp, revision
    order by stamp
    limit 100000)
```

5. 单击**流图**，进行**属性配置**之后，单击**添加到仪表盘**。



配置Pod扩缩容趋势统计

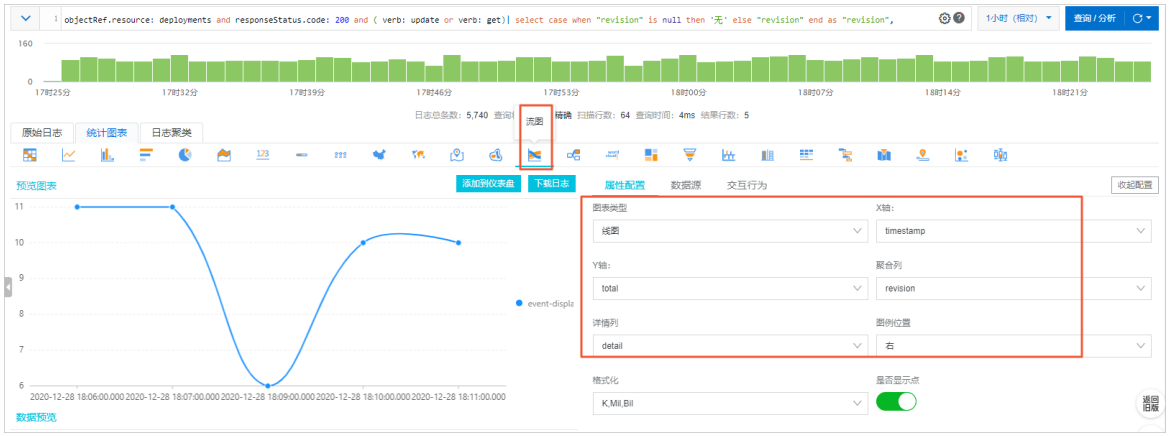
1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标ASK集群对应的日志服务Project名称。
3. 在**日志存储**页面的**日志库**的页签，单击audit名称开头的Logstore。
4. 从**查询分析属性**列表中选择**属性**。
5. 在**查询分析**面板的**指定字段查询**区域的**responseObject**字段添加以下参数。

参数名称	参数类型	别名
metadata.labels.serving.knative.dev/revision	text	revision
metadata.labels.serving.knative.dev/service	text	service
status.readyReplicas	long	readyReplicas
status.replicas	long	replicas

6. 在**查询/分析**文本框中输入以下SQL查询命令。

```
objectRef.resource: deployments and responseStatus.code: 200 and ( verb: update or verb: get)| select case when "revision" is null then '无' else "revision" end as "revision", "t" as timestamp, total,concat('total:', cast(total AS varchar), ', ready: ', cast(ready AS varchar), ', notReady',cast((total-ready) AS varchar)) as detail from (select date_trunc('minute' , __time__) as t, max(replicas) as total, max(readyReplicas) as ready, revision from log where service='event-display-common' group by t, revision order by t)
```

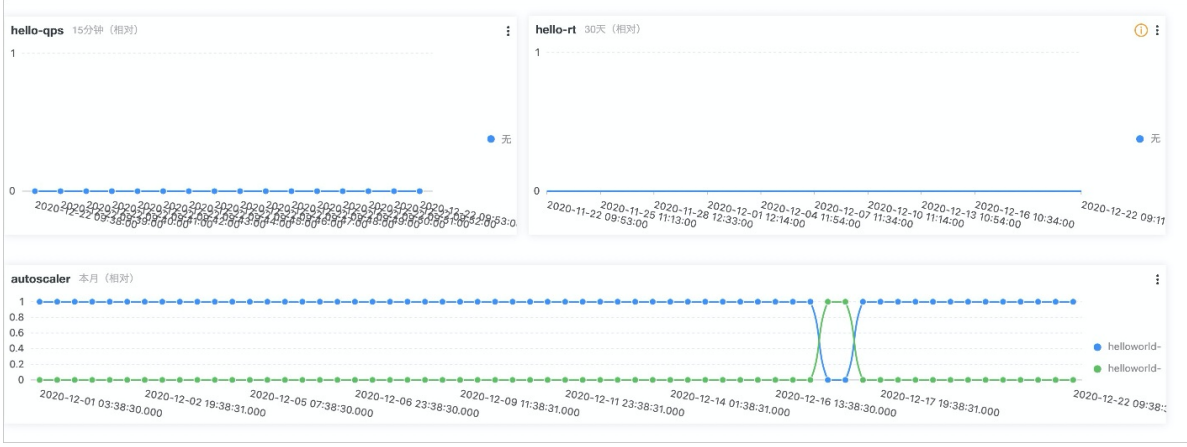
7. 单击**流图**，进行**属性配置**之后，单击**添加到仪表盘**。



观测服务运行状况

为了更好地观测服务运行状况的图表数据，您可将多个图表放在一个仪表盘进行展示，具体有以下两种方式：

- 添加仪表盘时，选择相同名称的仪表盘。
- 创建一个全局的仪表盘，然后将目标图表导入即可。具体操作，请参见创建仪表盘和添加统计图表到仪表盘。



16.7.2. 在Knative上观测服务的CPU和Memory使用情况

在服务运行过程中，您可以在Knative上观测当前服务运行的状况，包括CPU和Memory使用情况。本文主要介绍如何观测这些信息。

前提条件

- 阿里云Serverless Kubernetes（ASK）集群已开通Knative功能。具体操作，请参见开启Knative。
- ASK集群已开通阿里云Prometheus监控功能。具体操作，请参见阿里云Prometheus监控。

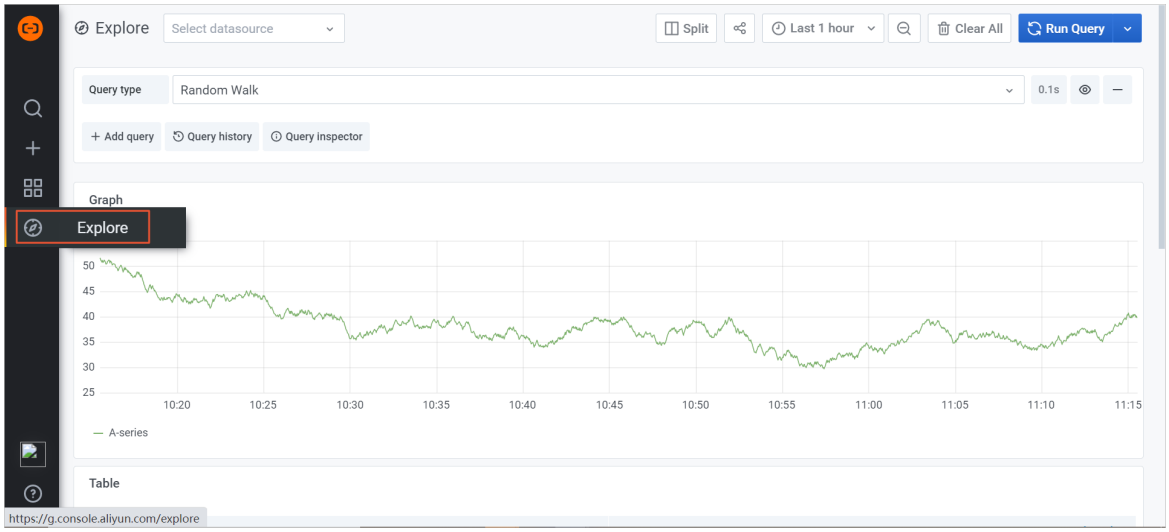
操作步骤

1. 登录ARMS控制台。
2. 在控制台左侧导航栏中，单击**Prometheus监控**，然后在该页面单击目标ASK集群。

   > **说明** 如果您没有安装过Prometheus，需先在**Prometheus监控**页面单击**安装**。

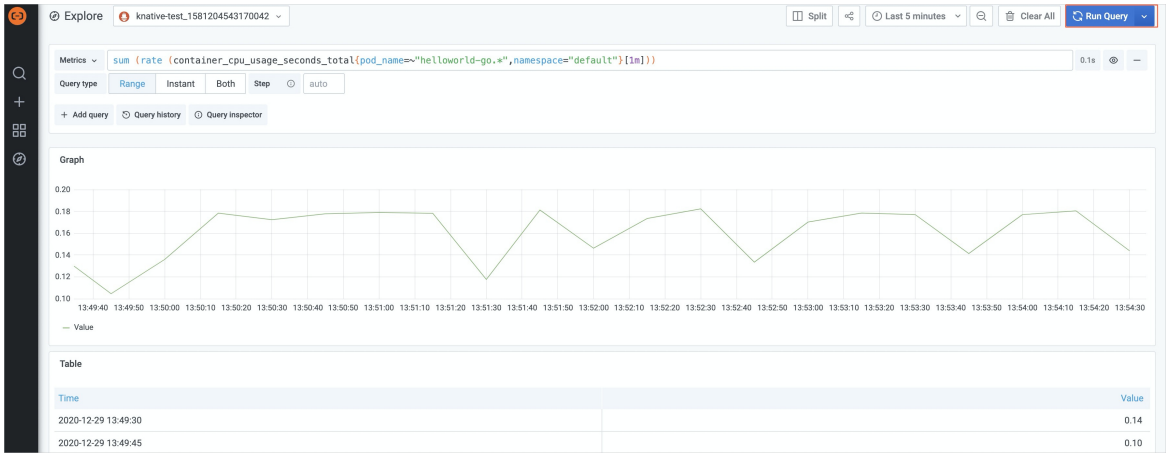
3. 在**大盘列表**页面的**名称**列单击**Prometheus**。
4. 单击左侧**Explore**，在**Select datasource**下拉列表选择目标集群。



5. 查询Knative服务的CPU使用情况。

   在**Metrics**文本框中输入以下PromQL查询命令，以**helloworld-go**服务为例，单击**Run Query**查询该服务的CPU使用情况：

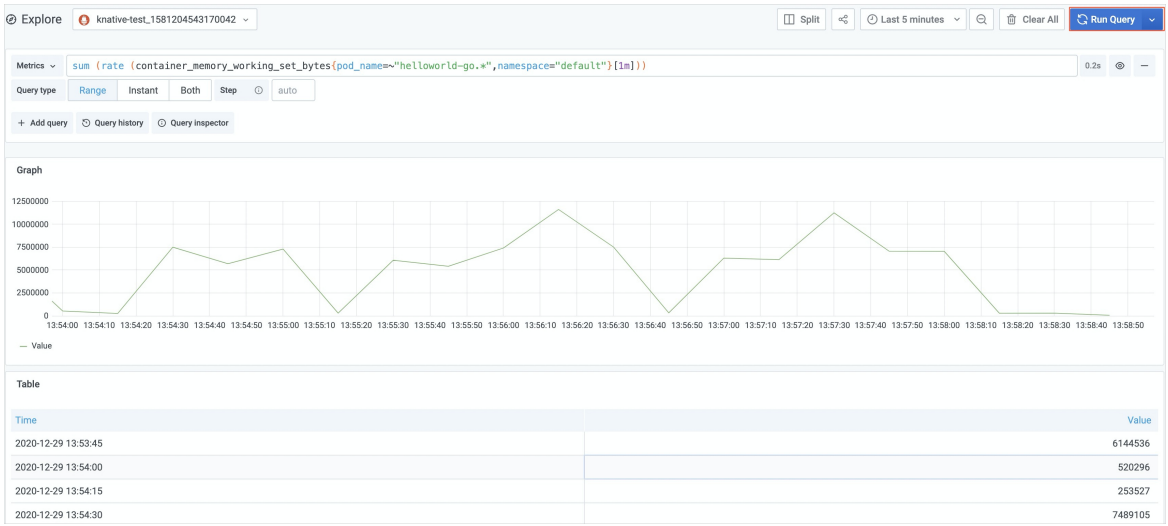
   ```
   sum (rate (container_cpu_usage_seconds_total{pod_name=~"helloworld-go.*",namespace="default"}[1m]))
   ```



6. 查询Knative服务的Memory使用情况。

   在**Metrics**文本框中输入以下PromQL查询命令，以**helloworld-go**服务为例，单击**Run Query**查询该服务的Memory使用情况：

   ```
   sum (rate (container_memory_working_set_bytes{pod_name=~"helloworld-go.*",namespace="default"}[1m]))
   ```


Explore
knative-test_1581204543170042
Split
Last 5 minutes
Clear All
Run Query
Metrics
sum (rate (container_memory_working_set_bytes{pod_name=~"helloworld-go.*",namespace="default"}[1m]))
0.2s
Query type
Range
Instant
Both
Step
auto
Add query
Query history
Query inspector
Graph
12500000
10000000
7500000
5000000
2500000
0
13:54:00 13:54:10 13:54:20 13:54:30 13:54:40 13:54:50 13:55:00 13:55:10 13:55:20 13:55:30 13:55:40 13:55:50 13:56:00 13:56:10 13:56:20 13:56:30 13:56:40 13:56:50 13:57:00 13:57:10 13:57:20 13:57:30 13:57:40 13:57:50 13:58:00 13:58:10 13:58:20 13:58:30 13:58:40 13:58:50
Value
Table
Time
Value
2020-12-29 13:53:45
6144536
2020-12-29 13:54:00
520296
2020-12-29 13:54:15
253527
2020-12-29 13:54:30
7489105

17.Serverless Kubernetes集群最佳实践

17.1. 通过ASK运行Job任务

在ASK集群中，您可以按需按量创建Pod。当Pod结束后停止收费，无需为Job任务预留计算资源，从而摆脱集群计算力不足和扩容的烦扰，同时结合抢占式实例可以降低Job任务的计算成本。本文主要为您介绍如何通过ASK按需创建Job任务。

前提条件

- 创建Serverless Kubernetes集群
- 通过kubectl连接Kubernetes集群

操作步骤

1. 通过kubectl客户端创建*job.yaml*文件，并拷贝以下内容到该文件。

```
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 apply -f job.yaml
```

3. 执行以下命令，查看Pod的运行状态。

   查看Pod的状态。

   ```
   kubectl get pod
   ```

   预期输出：

   ```
   NAME    READY  STATUS    RESTARTS  AGE
   pi-4f7w5  0/1    Completed  0      80s
   ```

   查看Pod的具体运行状态。

```
kubectl describe pod
```

预期输出：

```
Name:         pi-4f7w5
Namespace:    default
Priority:     0
PriorityClassName: <none>
Node:         virtual-kubelet-cn-hongkong-b/10.10.66.169
...
Events:
 Type   Reason           Age   From      Message
 ----   ------           ----  ----      -------
 Normal SuccessfulMountVolume 114s kubelet, eci MountVolume.SetUp succeeded for volume "default-token-8k4jz"
 Normal Pulling          113s kubelet, eci pulling image "perl"
 Normal Pulled           64s  kubelet, eci Successfully pulled image "perl"
 Normal Created          64s  kubelet, eci Created container
 Normal Started          64s  kubelet, eci Started container
```

4. （可选）通过给Pod加上抢占式实例的Annotation，使用抢占式实例。

   关于抢占式实例Annotation的用法，请参见使用抢占式实例。

```
apiVersion: batch/v1
kind: Job
metadata:
 name: pi
spec:
 template:
  metadata:
   annotations:
    k8s.aliyun.com/eci-spot-strategy: SpotAsPriceGo
  spec:
   containers:
   - name: pi
    image: perl
    command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
    resources:
     requests:
      cpu: 16
      memory: 32Gi
   restartPolicy: Never
 backoffLimit: 4
```

17.2. 通过ASK创建Spark计算任务

在ASK集群中，您可以按需按量创建Pod。当Pod结束后停止收费，无需为Spark计算任务预留计算资源，从而摆脱集群计算力不足和扩容的烦扰，同时结合抢占式实例可以降低任务的计算成本。本文主要为您介绍如何通过ASK按需创建Spark计算任务。

前提条件

- 已创建ASK集群。具体操作，请参见创建Serverless Kubernetes集群。

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

操作步骤

1. 部署ack-spark-operator Chart，可以通过以下两种方式：
   - 在容器服务管理控制台的导航栏中选择**市场 > 应用目录**，通过选择ack-spark-operator来进行部署。
   - 通过**helm**命令行手动安装。

     > **说明** 要求Helm的版本不低于V3。

     ```
     #创建Service account。
     kubectl create serviceaccount spark
     #绑定权限。
     kubectl create clusterrolebinding spark-role --clusterrole=edit --serviceaccount=default:spark --nam
     espace=default
     #安装Operator。
     helm repo add incubator http://storage.googleapis.com/kubernetes-charts-incubator
     helm install incubator/sparkoperator --namespace default  --set operatorImageName=registry.cn-ha
     ngzhou.aliyuncs.com/acs/spark-operator  --set operatorVersion=ack-2.4.5-latest  --generate-name
     ```

     部署后可以执行以下命令确认spark-operator已经启动成功。

     ```
     kubectl -n spark-operator get pod
     ```

     预期输出：

     ```
     NAME                              READY  STATUS    RESTARTS  AGE
     ack-spark-operator-7698586d7b-pvwln  1/1    Running   0      5m9s
     ack-spark-operator-init-26tvh      0/1    Completed  0      5m9s
     ```

2. 创建*spark-pi.yaml*文件并拷贝以下内容到该文件。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
 name: spark-pi
 namespace: default
spec:
 arguments:
 - "1000"
 sparkConf:
  "spark.scheduler.maxRegisteredResourcesWaitingTime": "3000s"
  "spark.kubernetes.allocation.batch.size": "1"
  "spark.rpc.askTimeout": "36000s"
  "spark.network.timeout": "36000s"
  "spark.rpc.lookupTimeout": "36000s"
  "spark.core.connection.ack.wait.timeout": "36000s"
  "spark.executor.heartbeatInterval": "10000s"
 type: Scala
 mode: cluster
 image: "registry.cn-shenzhen.aliyuncs.com/ringtail/spark-pi:0.4"
 imagePullPolicy: Always
 mainClass: org.apache.spark.examples.SparkPi
 mainApplicationFile: "local:///opt/spark/examples/jars/spark-examples_2.11-2.4.5.jar"
 sparkVersion: "2.4.5"
 restartPolicy:
  type: Never
 args:
 driver:
  cores: 4
  coreLimit: "4"
  annotations:
   k8s.aliyun.com/eci-image-cache: "true"
  memory: "6g"
  memoryOverhead: "2g"
  labels:
   version: 2.4.5
  serviceAccount: spark
 executor:
  annotations:
   k8s.aliyun.com/eci-image-cache: "true"
  cores: 2
  instances: 1
  memory: "3g"
  memoryOverhead: "1g"
  labels:
   version: 2.4.5
```

3. 部署Spark计算任务。
   i. 执行以下命令，部署Spark计算任务。

      ```
      kubectl apply -f spark-pi.yaml
      ```

      预期输出：

```
sparkapplication.sparkoperator.k8s.io/spark-pi created
```

ii. 执行以下命令，查看Spark计算任务的部署状态。

```
kubectl get pod
```

预期输出：

```
NAME        READY  STATUS   RESTARTS  AGE
spark-pi-driver  1/1    Running  0      2m12s
```

从预期输出可得，Pod的运行状态为Running，表示正在部署Spark计算任务。

iii. 执行以下命令，再次查看Spark计算任务的部署状态。

```
kubectl get pod
```

预期输出：

```
NAME        READY  STATUS    RESTARTS  AGE
spark-pi-driver  0/1    Completed  0      2m54s
```

从预期输出可得，Pod的运行状态为Completed，表示Spark计算任务已部署完成。

4. 执行以下命令，查看Spark任务的计算结果。

```
kubectl logs spark-pi-driver|grep Pi
```

预期输出：

```
20/04/30 07:27:51 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 11.031 s
20/04/30 07:27:51 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 11.137920 s
Pi is roughly 3.1414371514143715
```

5. （可选）通过给Pod加上抢占式实例的Annotation，使用抢占式实例。

关于抢占式实例Annotation的用法，请参见使用抢占式实例。

17.3. 如何给Pod挂载弹性公网IP

本文主要为您介绍如何在Serverless Kubernetes或虚拟节点中给Pod挂载EIP。

背景信息

阿里云Serverless Kubernetes服务、虚拟节点推出Pod挂载弹性公网IP功能，此功能使某些Serverless容器应用的部署和服务访问变得更加简单和便利。

- 无需创建VPC NAT网关即可让单个Pod访问公网。
- 无需创建Service也可让单个Pod暴露公网服务。
- 可以更加灵活而且动态的绑定Pod和EIP。

前提条件

- 创建一个Serverless Kubernetes集群或在Kubernetes集群创建一个部署虚拟节点Chart。
- 请确保该集群的安全组已开放相关端口（本示例中需要开放80端口）。

> 说明
> - 需要升级ack-virtual-node组件到v1.0.0.7-aliyun及之后版本。
> - 挂载弹性公网IP仅支持创建容器组时设置，更新容器组时添加或修改相关设置无效。

操作步骤

您可以通过以下两种方法给Pod挂载弹性公网IP。

方法一：自动分配弹性公网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**，选择示例模板或自定义，然后单击**创建**。

   您可以使用如下YAML示例模板创建Pod。本例中，通过指定*k8s.aliyun.com/eci-with-eip*为*true*，Serverless Kubernetes服务和虚拟节点会自动为此Pod分配一个EIP，并且绑定到Pod上。

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 annotations:
  k8s.aliyun.com/eci-with-eip: "true"
 # k8s.aliyun.com/eip-bandwidth: '5' #注意：指定带宽不需要带单位
spec:
 containers:
 - image: nginx:alpine
  imagePullPolicy: Always
  name: nginx
  ports:
  - containerPort: 80
   name: http
   protocol: TCP
 restartPolicy: OnFailure
```

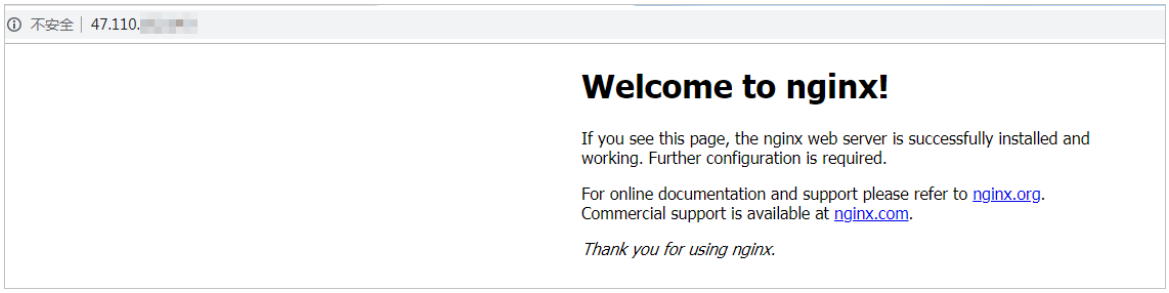
   > 说明
   > - 您可以通过Annotation *k8s.aliyun.com/eip-bandwidth*指定EIP的带宽，默认值为5，单位为Mbps。
   > - 您也可以通过Annotation *k8s.aliyun.com/eip-common-bandwith-package-id*让EIP绑定共享带宽。
   > - 如果您创建的是Deployment，那么Deployment中的每一个Pod都将会被挂载不同的EIP，请谨慎使用此操作。

6. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**，查看容器组的状态。
7. 在目标容器组右侧单击**编辑**，弹出**编辑YAML文件**。

> **说明** YAML文件中 `k8s.aliyun.com/allocated-eipAddress: 47.110.XX.XX` 的IP地址即为EIP的公网访问地址。

8. 在浏览器中输入*http://ip地址*，您可访问nginx欢迎页。

   此处的*http://ip地址*为YAML文件中 `k8s.aliyun.com/allocated-eipAddress: 47.110.XX.XX` 的IP地址。



> **说明** 此方式中EIP为动态分配，其生命周期与Pod相同，当删除Pod时，动态分配的EIP也会一并删除。

9. （可选）如果需要为ECI指定线路，需设置 `k8s.aliyun.com/eip-isp` 。

   ISP字段表示线路类型，默认为 `BGP` ，更多信息，请参见AllocateEipAddressPro。

   YAML样例模板如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 annotations:
  k8s.aliyun.com/eci-with-eip: "true"
  k8s.aliyun.com/eip-isp: "BGP"
spec:
 containers:
 - image: nginx:alpine
  name: nginx
  ports:
  - containerPort: 80
   name: http
   protocol: TCP
 restartPolicy: OnFailure
```

方法二：指定弹性公网IP实例ID

1. 登录VPC管理控制台，购买弹性公网IP。请参见申请EIP。

> **说明** 申请的EIP和集群必须在同一地域。

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**工作负载 > 无状态**。

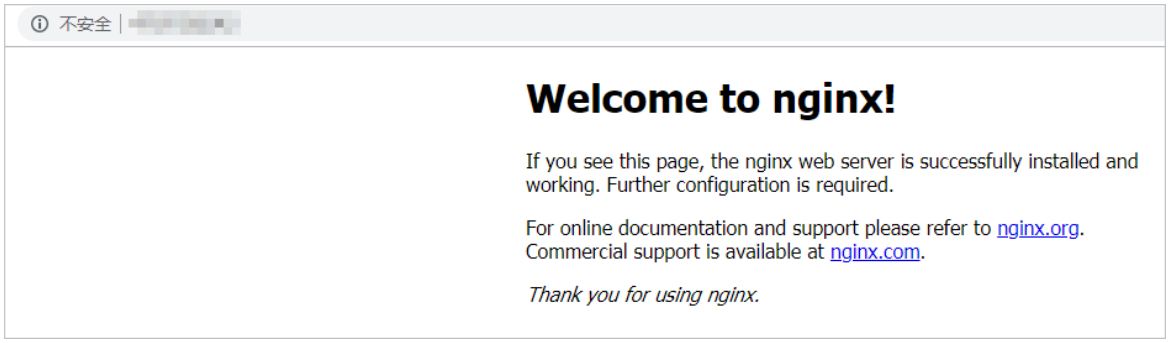
6. **无状态**页面，然后单击**使用YAML创建**，选择示例模板或自定义，然后单击**创建**。

   您可以使用如下YAML示例模板创建Pod。本例中，通过指定Pod的Annonation *k8s.aliyun.com/eci-eip-instanceid*为EIP实例ID。

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 annotations:
  "k8s.aliyun.com/eci-eip-instanceid": "<youreipInstanceId>"
spec:
 containers:
 - image: nginx:alpine
  imagePullPolicy: Always
  name: nginx
  ports:
  - containerPort: 80
   name: http
   protocol: TCP
 restartPolicy: OnFailure
```

   > **说明**
   > - `<youreipInstanceId>` 需要替换成步骤1中获取的EIP实例ID。
   > - 如果同时设置了自动分配弹性公网EIP和指定弹性公网IP实例ID，则手动指定的EIP无效。

7. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**，查看容器组的状态。
8. 在浏览器中输入*http://ip地址*，您可访问nginx欢迎页。



> **说明** 此处的*http://ip地址*为步骤1中申请的EIP的IP地址。

17.4. 在ASK上快速搭建Jenkins环境及执行流水线构建

本文主要演示如何在阿里云Serverless Kubernetes服务（ASK）上快速搭建Jenkins持续集成环境，并基于提供的应用示例快速完成应用源码编译、镜像构建和推送以及应用部署的流水线。

前提条件

- 已创建ASK集群。具体操作，请参见创建Serverless Kubernetes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。

部署Jenkins

1. 执行以下命令下载部署文件。

```
git clone https://github.com/AliyunContainerService/jenkins-on-serverless.git
cd jenkins-on-serverless
```

2. 完成 jenkins_home 持久化配置。

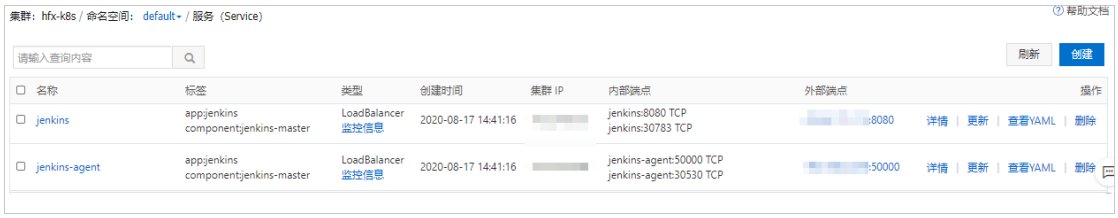
   Serverless Kubernetes目前不支持云盘，如需持久化 jenkins_home ，您可以挂载 nfs volume ，修改*serverless-k8s-jenkins-deploy.yaml*文件，取消以下字段注释并配置您的NFS信息：

```
#volumeMounts:
    # - mountPath: /var/jenkins_home
    #   name: jenkins-home
  .....
  #volumes:
  # - name: jenkins-home
  #   nfs:
  #     path: /
  #     server:
```

3. 执行以下命令部署Jenkins。

```
kubectl apply -f serverless-k8s-jenkins-deploy.yaml
```

4. 登录Jenkins。
   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. 单击Jenkins服务的外部端点登录Jenkins。



   vi. 在Jenkins登录页面，输入用户名和密码。默认用户名和密码均为**admin**。

> 注意　为了保证Jenkins系统安全，请登录后修改密码。

5. 配置Jenkins的**新手入门**。
   i. 在**新手入门**配置页面，单击**安装推荐的插件**。
   ii. 插件安装完成后，在**新手入门**的**实例配置**页面，单击**保存并完成**。
   iii. 实例配置保存完成后，单击**开始使用Jenkins**。
6. 获取Token的Secret。

i. 执行以下命令查看Token。

```
kubectl get secret
```

预期输出：

```
NAME                  TYPE                                  DATA   AGE
ack-jenkins-sa-token-q****   kubernetes.io/service-account-token   3      28m
default-token-b****          kubernetes.io/service-account-token   3      27h
```

ii. 执行以下命令获取**ack-jenkins-sa-token-q****的Secret。

```
kubectl get secret ack-jenkins-sa-token-q**** -o jsonpath={.data.token} |base64 -d
```

预期输出：

```
sdgdrh****
```

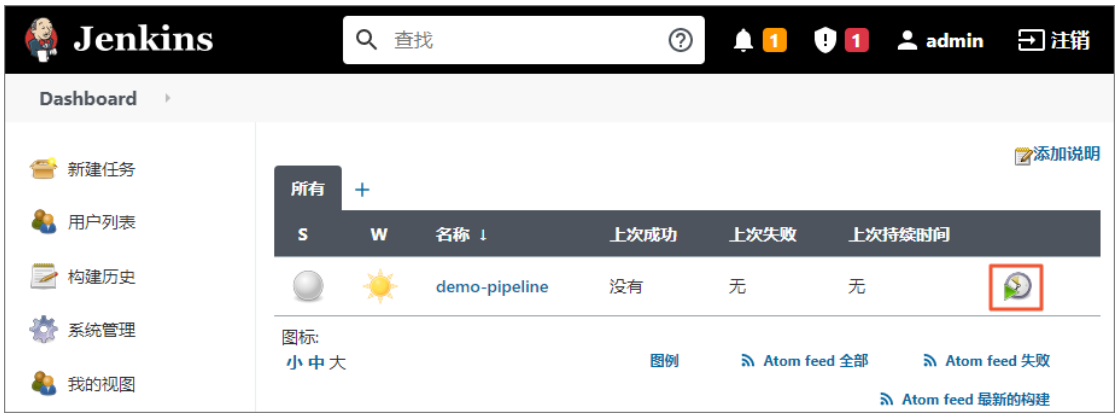
7. 创建Secret Text类型凭证。
   i. 在Jenkins系统左侧导航栏，选择**系统管理**。
   ii. 在**管理Jenkins**页面，**系统配置**下单击**节点管理**。
   iii. 在**节点列表**页面左侧导航栏，选择**Configure Clouds**。
   iv. 在**配置集群**页面，单击**Kubernetes Cloud details...**。
   v. 在**凭据**字段，选择**添加 > Jenkins**。
   vi. 在**Jenkins 凭据提供者: Jenkins**对话框，添加Secret Text类型的凭证。

   凭证的参数说明如下所示：

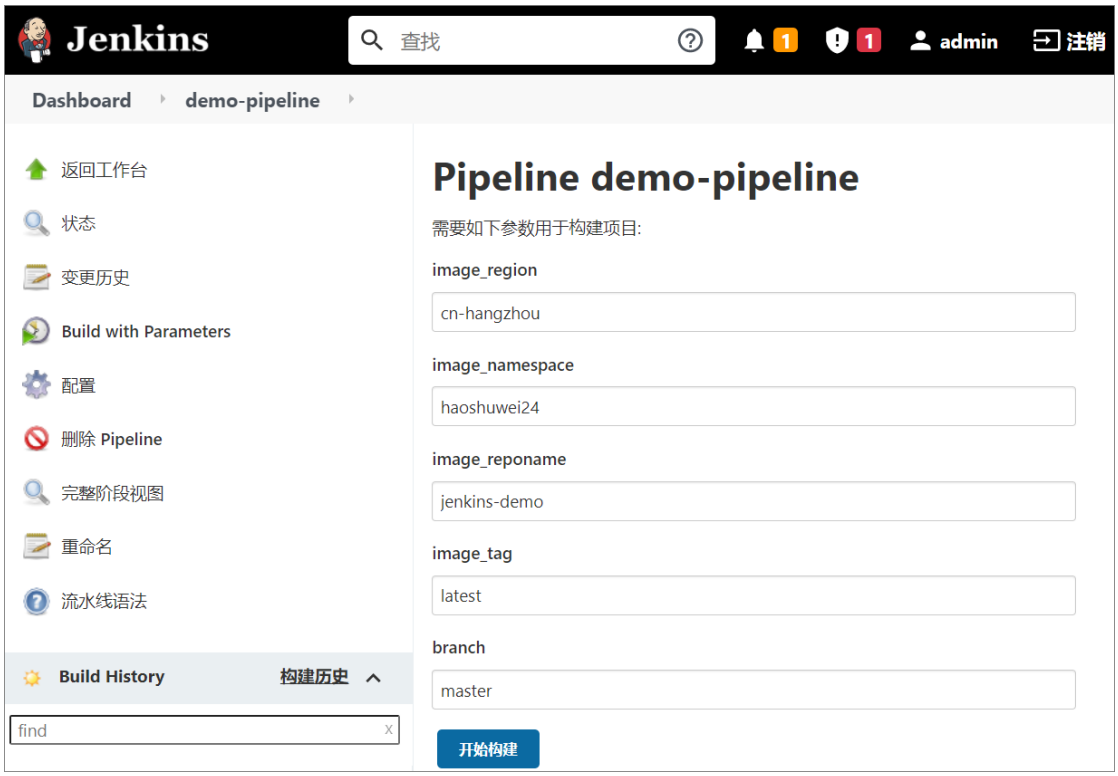
参数	说明
Domain	表示凭据的域。默认为**全局凭据（unrestricted）**
类型	表示凭据的类型。本示例选择**Secret Text**。
范围	表示凭据的范围，可选择全局或系统。本示例选择**全局（Jenkins，nodes，items，all child items，etc）**。
Secret	表示凭据的Secret。本示例输入上个步骤获取的Secret。
ID	表示凭据的名称。本示例为**ask-jenkins-token**。
描述	表示凭据的补充说明。

   vii. 单击**添加**。
8. 在**配置集群**页面配置相关参数。更多信息，请参见Kubernetes Cloud的配置说明。
   i. 配置**Kubernetes地址**，**凭据**选择为**ask-jenkins-token**，单击**连接测试**验证连接是否正常。
   ii. 配置**Jenkins地址**和**Jenkins通道**。
   iii. 单击**Save**。
9. 构建**demo-pipeline**并访问应用服务。

i. 在Jenkins首页，单击**demo-pipeline**的图标。



ii. 根据您的镜像仓库信息修改构建参数。本示例中源码仓库分支为 `master` ，镜像为 `registry.cn-beijing.aliyuncs.com/ack-cicd/ack-jenkins-demo:latest` 。

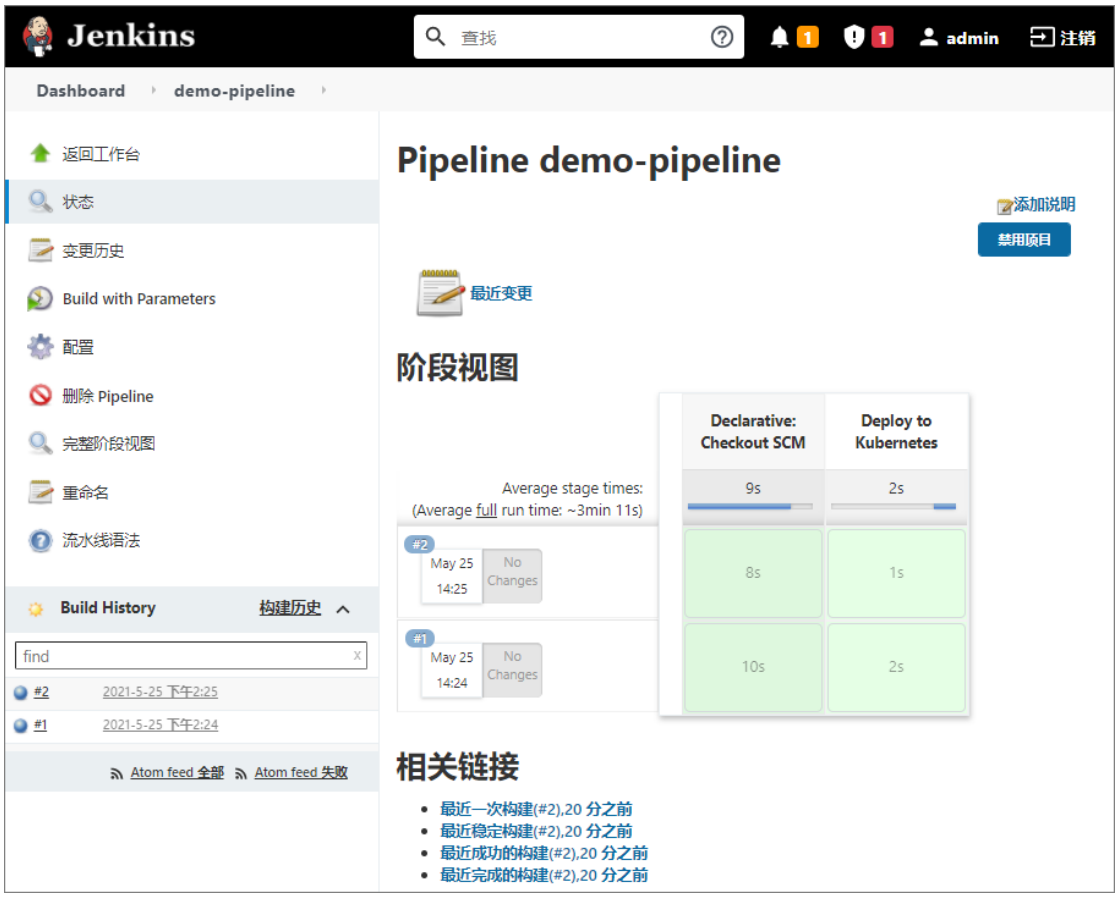


iii. 单击**开始构建**。

测试Kubernetes集群动态分配的Jenkins Slave Pod与Jenkins Master是否连接正常。

执行构建后，Jenkins从Kubernetes集群动态创建一个Slave Pod运行本次构建任务。关于示例应用代码，请参见jenkins-demo-GitHub或jenkins-demo-haoshuwei。

iv. 查看**状态**，若构建成功则表示Jenkins on Kubernetes运行正常。



后续步骤

- 关于如何为Slave Pod配置Maven缓存，请参见为Slave Pod配置Maven缓存。
- 关于如何使用kaniko构建和推送容器镜像，请参见使用kaniko构建和推送容器镜像。

17.5. 搭建TensorFlow应用

本文介绍如何使用阿里云Serverless Kubernetes（ASK）和弹性容器实例（ECI），快速完成基于GPU的TensorFlow训练任务。

背景信息

人工智能与机器学习已经被广泛应用到各个领域，近些年来各种各样的训练模型被提出，更多的训练任务运行到云上。然而上云之后，想要轻松、持久地运行训练任务，仍有一些痛点，例如：

- 环境搭建麻烦：您需要购买GPU实例并安装GPU驱动，即使您已经把训练任务容器化，还需要安装GPU runtime hook。
- 使用缺乏弹性：运行完任务后，为了节约成本，您可能需要释放闲置资源，但在下次启动任务时您需要重新创建实例并配置环境；或者在计算节点资源不够的情况下，扩容需要您再次创建实例并配置环境。

针对上述痛点，推荐您使用ASK+ECI的方案来运行训练任务，可以帮助您轻松地创建和启动训练任务。该方案还具备以下优势：

- 按需付费，免运维。
- 一次配置，无限次复用。

- 镜像缓存功能加速Pod创建，训练任务启动快速。
- 数据与训练模型解耦，数据可以持久化存储。

准备工作

1. 准备好训练模型的容器镜像和训练数据。

   本文以Github的一个TensorFlow训练任务为例，相关示例镜像（eci/tensorflow）您可以从阿里云容器镜像仓库中获取。更多信息，请参见TensorFlow训练任务。

2. 创建ASK集群。

   在容器服务管理控制台上创建ASK集群。更多信息，请参见创建Serverless Kubernetes集群。

   > 说明
   >
   > 如果您需要从公网拉取镜像，或者训练任务需要访问公网，请配置NAT网关。

   您可以通过kubectl管理和访问ASK集群，相关操作如下：

   - 如果您需要从本地算机管理集群，请安装并配置kubectl客户端。具体操作，请参见通过kubectl连接Kubernetes集群。
   - 您也可以在CloudShell上通过kubect管理集群。具体操作，请参见在CloudShell上通过kubectl管理Kubernetes集群。

3. 创建NAS文件系统，并添加挂载点。

   在NAS文件系统控制台上创建文件系统，并添加挂载点。更多信息，请参见管理文件系统和管理挂载点。

   > 说明
   >
   > NAS文件系统需和ASK集群处于同一VPC。

创建镜像缓存

镜像缓存功能已经集成到Kubernetes CRD中，可以加速镜像的拉取。更多信息，请参见使用镜像缓存CRD加速创建Pod。

操作如下：

1. 准备yaml文件。

   示例imagecache.yaml的内容如下：

   ```
   apiVersion: eci.alibabacloud.com/v1
   kind: ImageCache
   metadata:
    name: tensorflow
   spec:
    images:
    - registry-vpc.cn-beijing.aliyuncs.com/eci/tensorflow:1.0 # 训练任务的镜像，建议您上传到阿里云容器镜像仓库中。此处使用VPC私网地址，请确保对应VPC为集群所属的VPC。
   ```

2. 创建镜像缓存。

   ```
   kubectl create -f imagecache.yaml
   ```

创建镜像缓存时需要拉取镜像，受镜像大小影响，需要一定的时间。您可以通过以下命令查询镜像缓存的创建进度。

```
Kubectl get imagecache tensorflow
```

返回如下结果时，表示镜像缓存已经创建成功。

```
NAME     AGE  CACHEID          PHASE  PROGRESS
tensorflow  11m  imc-2ze1xczztv7tgesg****  Ready  100%
```

创建训练任务

您可以使用镜像缓存来创建训练任务。

1. 准备yaml文件。

   示例gpu_pod.yaml的内容如下：

```
apiVersion: v1
kind: Pod
metadata:
 name: tensorflow
 annotations:
  k8s.aliyun.com/eci-use-specs: "ecs.gn6i-c4g1.xlarge" # 指定GPU规格创建ECI实例
  k8s.aliyun.com/eci-image-cache: "true"        # 开启镜像缓存自动匹配
spec:
 containers:
 - name: tensorflow
  image: registry-vpc.cn-beijing.aliyuncs.com/eci/tensorflow:1.0 # 训练任务的镜像
  command:
   - "sh"
   - "-c"
   - "python models/tutorials/image/imagenet/classify_image.py" # 触发训练任务的脚本
  resources:
   limits:
    nvidia.com/gpu: "1"  # 容器所需的GPU个数
  volumeMounts:
  - name: nfs-pv
   mountPath: /tmp/imagenet
 volumes:
 - name: nfs-pv  #训练结果持久化，保存到NAS
  flexVolume:
    driver:  alicloud/nas
    fsType: nfs
    options:
     server: 16cde4****-ijv**.cn-beijing.nas.aliyuncs.com   #NAS文件系统挂载点
     path: /      #挂载目录
 restartPolicy: OnFailure
```

2. 执行命令创建Pod。

```
kubectl create -f gpu_pod.yaml
```

3. 查看执行情况。

   您可以根据需要查看事件或日志。

- 查看事件

```
kubectl describe pod tensorflow
```

- 查看日志

```
kubectl logs tensorflow
```

查看结果

您可以在控制台上查看训练任务的运行结果。

- 在NAS文件系统控制台，您可以看到训练完成的数据已占用存储容量。

> **说明**
>
> 训练结果存储在配置的NAS文件系统中，挂载NAS后，您可以在对应的路径下获取结果数据。

- 在弹性容器实例控制台，您可以看到运行成功的ECI实例。

> **说明**
>
> 运行成功表示实例中的容器已经运行终止，此时系统会回收底层计算资源，不再对Pod进行计费。

17.6. 搭建Spark应用

本文介绍如何使用阿里云Serverless Kubernetes（ASK）和弹性容器实例（ECI），快速搭建Spark应用。

背景信息

Apache Spark是一个在数据分析领域广泛使用的开源项目，它常被应用于众所周知的大数据和机器学习工作负载中。从Apache Spark 2.3.0版本开始，您可以在Kubernetes上运行和管理Spark资源。

Spark Operator是专门针对Spark on Kubernetes设计的Operator，开发者可以通过使用CRD的方式，提交Spark任务到Kubernetes集群中。使用Spark Operator有以下优势：

- 能够弥补原生Spark对Kubernetes支持不足的部分。
- 能够快速和Kubernetes生态中的存储、监控、日志等组件对接。
- 支持故障恢复、弹性伸缩、调度优化等高阶Kubernetes特性。

准备工作

1. 创建ASK集群。

   在容器服务管理控制台上创建ASK集群。更多信息，请参见创建Serverless Kubernetes集群。

> 说明
> 如果您需要从公网拉取镜像，或者训练任务需要访问公网，请配置NAT网关。

您可以通过kubectl管理和访问ASK集群，相关操作如下：

- 如果您需要从本地算机管理集群，请安装并配置kubectl客户端。具体操作，请参见通过kubectl连接Kubernetes集群。
- 您也可以在CloudShell上通过kubect管理集群。具体操作，请参见在CloudShell上通过kubectl管理Kubernetes集群。

2. 创建OSS存储空间。

   您需要创建一个OSS存储空间（Bucket）用来存放测试数据、测试结果和测试过程中的日志等。关于如何创建OSS Bucket，请参见创建存储空间。

安装Spark Operator

1. 安装Spark Operator。
   i. 在容器服务管理控制台的左侧导航栏，选择**市场>应用目录**。
   ii. 在**应用目录**页面，找到并单击**ack-spark-operator**。
   iii. 在左侧单击**参数**页签，修改YAML配置信息，然后单击右侧的**创建**。

   配置信息中，sparkJobNamespace为需要部署Spark作业的命名空间，默认为 `default` ，如果设置为空字符串则表示所有命名空间都监听。

   > 说明
   > Spark作业需要一个ServiceAccount来获取创建Pod的权限，因此需要创建ServiceAccount、Role和Rolebinding，请根据需要修改三者的Namespace。

2. 执行以下命令确认Spark Operator是否已经启动。

```
kubectl -n spark-operator get pod
```

如果返回类似如下结果，则表示已经启动Spark Operator。

```
NAME                                  READY  STATUS    RESTARTS  AGE
ack-spark-operator-7698586d7b-pvwln  1/1    Running   0      5m9s
ack-spark-operator-init-26tvh      0/1    Completed  0      5m9s
```

（可选）构建Spark作业镜像

您需要编译Spark作业的Jar包，使用Dockerfile打包镜像。以阿里云容器服务的Spark基础镜像为例，设置Dockerfile内容如下：

```
FROM registry.cn-hangzhou.aliyuncs.com/acs/spark:ack-2.4.5-latest
RUN mkdir -p /opt/spark/jars
# 如果需要使用OSS（读取OSS数据或者离线Event到OSS），可以添加以下ar包到镜像中
ADD https://repo1.maven.org/maven2/com/aliyun/odps/hadoop-fs-oss/3.3.8-public/hadoop-fs-oss-3.3.8-public.jar $SPARK_HOME/jars
ADD https://repo1.maven.org/maven2/com/aliyun/oss/aliyun-sdk-oss/3.8.1/aliyun-sdk-oss-3.8.1.jar $SPARK_HOME/jars
ADD https://repo1.maven.org/maven2/org/aspectj/aspectjweaver/1.9.5/aspectjweaver-1.9.5.jar $SPARK_HOME/jars
ADD https://repo1.maven.org/maven2/org/jdom/jdom/1.1.3/jdom-1.1.3.jar $SPARK_HOME/jars
COPY SparkExampleScala-assembly-0.1.jar /opt/spark/jars
```

说明

- 阿里云提供了Spark2.4.5的基础镜像，针对Kubernetes场景（调度、弹性）进行了优化，建议您使用阿里云基础镜像。
- Spark镜像如果较大，则拉取需要较长时间，您可以通过ImageCache加速镜像拉取。更多信息，请参见使用镜像缓存CRD加速创建Pod。

编写作业模板并提交作业

创建一个Spark作业的YMAL配置文件，并进行部署。

1. 创建spark-pi.yaml文件。

   一个典型的作业模板示例如下。更多信息，请参见spark-on-k8s-operator。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
 name: spark-pi
 namespace: default
spec:
 type: Scala
 mode: cluster
 image: "registry.aliyuncs.com/acs/spark-pi:ack-2.4.5-latest"
 imagePullPolicy: Always
 mainClass: org.apache.spark.examples.SparkPi
 mainApplicationFile: "local:///opt/spark/examples/jars/spark-examples_2.11-2.4.5.jar"
 sparkVersion: "2.4.5"
 restartPolicy:
  type: Never
 driver:
  cores: 2
  coreLimit: "2"
  memory: "3g"
  memoryOverhead: "1g"
  labels:
   version: 2.4.5
  serviceAccount: spark
  annotations:
   k8s.aliyun.com/eci-kube-proxy-enabled: 'true'
   k8s.aliyun.com/eci-image-cache: "true"
  tolerations:
  - key: "virtual-kubelet.io/provider"
   operator: "Exists"
 executor:
  cores: 2
  instances: 1
  memory: "3g"
  memoryOverhead: "1g"
  labels:
   version: 2.4.5
  annotations:
   k8s.aliyun.com/eci-kube-proxy-enabled: 'true'
   k8s.aliyun.com/eci-image-cache: "true"
  tolerations:
  - key: "virtual-kubelet.io/provider"
   operator: "Exists"
```

2. 部署一个Spark计算任务。

```
kubectl apply -f spark-pi.yaml
```

（可选）配置日志采集

以采集Spark的标准输出和错误输出日志为例，您可以在Spark driver和Spark executor的envVars字段中注入环境变量，实现日志的自动采集。

```
envVars:
  aliyun_logs_project: test-k8s-spark
  aliyun_logs_machinegroup: k8s-group-app-spark
  aliyun_logs_test-stdout: stdout
```

提交作业时，如果按上述方式设置了driver和executor的环境变量，即可实现日志的自动采集。

（可选）配置历史服务器

历史服务器用于审计Spark作业，您可以通过在Spark Applicaiton的CRD中增加SparkConf字段的方式，将event写入到OSS，再通过历史服务器读取OSS的方式进行展现。配置示例如下：

```
sparkConf:
  "spark.eventLog.enabled": "true"
  "spark.eventLog.dir": "oss://bigdatastore/spark-events"
  "spark.hadoop.fs.oss.impl": "org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem"
  # oss bucket endpoint such as oss-cn-beijing.aliyuncs.com
  "spark.hadoop.fs.oss.endpoint": "oss-cn-beijing.aliyuncs.com"
  "spark.hadoop.fs.oss.accessKeySecret": ""
  "spark.hadoop.fs.oss.accessKeyId": ""
```

阿里云也提供了spark-history-server的Chart，您可以在容器服务管理控制台的**应用目录**页面，搜索ack-spark-history-server进行安装。安装时，请在**参数**页签配置OSS的相关信息。示例如下：

```
oss:
  enableOSS: true
  # Please input your accessKeyId
  alibabaCloudAccessKeyId: ""
  # Please input your accessKeySecret
  alibabaCloudAccessKeySecret: ""
  # oss bucket endpoint such as oss-cn-beijing.aliyuncs.com
  alibabaCloudOSSEndpoint: "oss-cn-beijing.aliyuncs.com"
  # oss file path such as oss://bucket-name/path
  eventsDir: "oss://bigdatastore/spark-events"
```

安装完成后，您可以在集群详情页面的服务中看到ack-spark-history-server的对外地址，访问对外地址即可查看历史任务归档。

查看作业结果

1. 查看Pod的执行情况。

   ```
   kubectl get pods
   ```

   预期返回结果：

   ```
   NAME                               READY   STATUS    RESTARTS   AGE
   spark-pi-1547981232122-driver      1/1     Running   0          12s
   spark-pi-1547981232122-exec-1      1/1     Running   0          3s
   ```

2. 查看实时Spark UI。

   ```
   kubectl port-forward spark-pi-1547981232122-driver 4040:4040
   ```

3. 查看Spark App的状态。

```
kubectl describe sparkapplication spark-pi
```

预期返回结果：

```
Name:       spark-pi
Namespace:  default
Labels:     <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
              {"apiVersion":"sparkoperator.k8s.io/v1alpha1","kind":"SparkApplication","metadata":{"annota
tions":{},"name":"spark-pi","namespace":"defaul...
API Version: sparkoperator.k8s.io/v1alpha1
Kind:       SparkApplication
Metadata:
 Creation Timestamp: 2019-01-20T10:47:08Z
 Generation:       1
 Resource Version:  4923532
 Self Link:        /apis/sparkoperator.k8s.io/v1alpha1/namespaces/default/sparkapplications/spark-pi
 UID:              bbe7445c-1ca0-11e9-9ad4-062fd7c19a7b
Spec:
 Deps:
 Driver:
  Core Limit: 200m
  Cores:     0.1
  Labels:
   Version:     2.4.0
  Memory:       512m
  Service Account: spark
  Volume Mounts:
   Mount Path: /tmp
   Name:      test-volume
 Executor:
  Cores:    1
  Instances: 1
  Labels:
   Version: 2.4.0
  Memory:   512m
  Volume Mounts:
   Mount Path:      /tmp
   Name:            test-volume
 Image:             gcr.io/spark-operator/spark:v2.4.0
 Image Pull Policy:    Always
 Main Application File: local:///opt/spark/examples/jars/spark-examples_2.11-2.4.0.jar
 Main Class:         org.apache.spark.examples.SparkPi
 Mode:               cluster
 Restart Policy:
  Type: Never
 Type:   Scala
 Volumes:
  Host Path:
   Path: /tmp
   Type: Directory
  Name:   test-volume
Status:
```

```
Status:
 Application State:
  Error Message:
  State:       COMPLETED
 Driver Info:
  Pod Name:         spark-pi-driver
  Web UI Port:      31182
  Web UI Service Name:  spark-pi-ui-svc
 Execution Attempts:   1
 Executor State:
  Spark - Pi - 1547981232122 - Exec - 1:  COMPLETED
 Last Submission Attempt Time:         2019-01-20T10:47:14Z
 Spark Application Id:              spark-application-1547981285779
 Submission Attempts:               1
 Termination Time:                  2019-01-20T10:48:56Z
Events:
 Type    Reason              Age          From         Message
 ----    ------              ----         ----         -------
 Normal  SparkApplicationAdded     55m          spark-operator  SparkApplication spark-pi was added, E
nqueuing it for submission
 Normal  SparkApplicationSubmitted  55m          spark-operator  SparkApplication spark-pi was submi
tted successfully
 Normal  SparkDriverPending      55m (x2 over 55m)   spark-operator  Driver spark-pi-driver is pending
 Normal  SparkExecutorPending     54m (x3 over 54m)   spark-operator  Executor spark-pi-154798123212
2-exec-1 is pending
 Normal  SparkExecutorRunning     53m (x4 over 54m)   spark-operator  Executor spark-pi-15479812321
22-exec-1 is running
 Normal  SparkDriverRunning      53m (x12 over 55m)  spark-operator  Driver spark-pi-driver is running
 Normal  SparkExecutorCompleted    53m (x2 over 53m)   spark-operator  Executor spark-pi-1547981232
122-exec-1 completed
```

4. 查看日志获取结果。

   当Spark App的状态为succeed或者spark driver对应的Pod状态为completed时，可以查看日志获取结果。

```
kubectl logs spark-pi-1547981232122-driver
Pi is roughly 3.152155760778804
```

17.7. 搭建WordPress应用

本文介绍如何使用Cloud Shell来快速搭建基于阿里云Serverless Kubernetes（ASK）和弹性容器实例（ECI）的WordPress应用。

背景信息

WordPress是使用PHP语言开发的博客平台，在支持PHP和MySQL数据库的服务器上，您可以用WordPress架设自己的网站，也可以用作内容管理系统（CMS）。

Clos

本教程已在Cloud Shell中集成，您也可以打开Cloud Shell，

创建ASK集群

1. 打开Cloud Shell。

2. 输入以下命令切换地域。

```
switch-region cn-beijing
```

3. （可选）修改集群配置文件。
4. 执行以下命令创建ASK集群。
5. 根据需要执行以下命令查看集群信息。
   - 查看集群的属性

     ```
     aliyun cs GET /clusters/<YOUR-CLUSTER-ID>
     ```

   - 查看集群的配置信息

安装WordPress

访问WordPress

打开Cloud Shell

> **说明**
> 以下步骤均已在Cloud Shell中集成，可以通过打开上面的链接来快速体验通过Cloud Shell操作ECI。

创建Serverless Kubernetes集群

请参见创建Serverless Kubernetes集群。

安装WordPress

> **注意**
> 请确保上一步中创建的 Serverless Kubernetes 集群，已完成初始化（一般需要3~5分钟），再开始以下的操作。

使用Cloud Shell来管理上一步中创建中的Serverless Kubernetes集群。

```
source use-k8s-cluster ${集群ID}
```

执行WordPress安装yaml文件。

```
kubectl apply -f wordpress-all-in-one-pod.yaml
```

观察安装进度，直到STATUS为Running。

```
kubectl get pods
```

查询EIP地址。

```
kubectl get -o json pod wordpress |grep "k8s.aliyun.com/allocated-eipAddress"
```

由于安全组默认没有开放 80 端口的访问，需要给安全组添加 80 端口的 ACL。

首先获取Serverless Kubernetes集群自动创建的安全组，找到最近创建的以 alicloud-cs-auto-created 为命名前缀的安全组ID。

```
aliyun ecs DescribeSecurityGroups|grep -B 1 'alicloud-cs-auto-created'|head -1
```

对安全组进行授权操作。

```
aliyun ecs AuthorizeSecurityGroup --RegionId cn-chengdu --SecurityGroupId ${安全组ID} --IpProtocol tcp --PortRange 80/80 --SourceCidrIp 0.0.0.0/0 --Priority 100
```

使用WordPress

在浏览器起输入上一步获取到的EIP地址，即可开始使用WordPress。

17.8. 使用抢占式实例运行Job任务

抢占式ECI实例特别适合于Job任务或者无状态应用，可以有效地节约实例使用成本。本文介绍如何使用抢占式ECI实例来运行Job任务。

背景信息

使用抢占式实例前，请了解以下相关信息：

- 抢占式实例的市场价格随供需变化而浮动，您需要在创建实例时指定出价模式，当市场价格低于出价且资源库存充足时，就能成功创建抢占式实例。
- 抢占式实例创建成功后有一个小时的保护期，超出保护期后，如果出价低于市场价格或者资源库存不足，实例会被释放。在释放前三分钟左右，ECI会通过Kubernetes Events事件通知的方式告知您。在此期间，您可以做一定的处理来确保业务不受实例释放所影响（如切断该实例的业务入口流量）。
- 抢占式实例由于竞价失败而释放后数据会随之一起释放，但实例信息会保留（状态变更为Failed，失败原因为BidFailed）。如果您有重要数据需要保留，建议配置持久化存储，如云盘，NAS等。

更多信息，请参见抢占式实例概述和创建抢占式实例。

配置方法

您可以在Pod的声明中设置Annotation来指定创建抢占式ECI实例。相关配置项如下：

- k8s.aliyun.com/eci-spot-strategy：设置抢占式实例的出价策略。取值如下：
  - SpotAsPriceGo：系统自动出价，跟随当前市场实际价格。
  - SpotWithPriceLimit：自定义设置抢占实例价格上限。
- k8s.aliyun.com/eci-spot-price-limit：设置抢占式实例的每小时价格上限，最多支持精确到小数点后三位。当k8s.aliyun.com/eci-spot-strategy配置为SpotWithPriceLimit时必须设置。

> **说明**
>
> 创建抢占式ECI实例时，建议您设置k8s.aliyun.com/eci-use-specs来指定多个规格，可以有效提升实例创建的成功率。

配置示例如下：

- SpotAsPriceGo

  该模式下，系统跟随当前市场价格自动出价。

```
apiVersion: v1
kind: Pod
metadata:
 name: spot-as-price-go
 annotations:
  k8s.aliyun.com/eci-use-specs: ecs.sn1ne.large,2-4Gi
  # SpotAsPriceGo不需要设置价格上限；系统自动出价，跟随当前市场实际价格
  k8s.aliyun.com/eci-spot-strategy: SpotAsPriceGo
```

- SpotWithPriceLimit

  该模式下，系统根据您设置的价格上限来出价，可能会出现以下几种情况：

  - 出价 < 当前市场价格：实例会处于Pending状态，并每5分钟自动进行一次出价，直到出价等于或高于市场价格时，开始自动创建实例。
  - 出价≥当前市场价格：如果库存充足，则自动创建实例。

```
apiVersion: v1
kind: Pod
metadata:
 name: spot-with-price-limit
 annotations:
  k8s.aliyun.com/eci-use-specs: ecs.sn1ne.large
  k8s.aliyun.com/eci-spot-strategy: SpotWithPriceLimit
  # SpotWithPriceLimit模式必须设置价格上限，如果设置的出价高于ECI实例当前的按量价格，使用按量价格来创建实例。
  k8s.aliyun.com/eci-spot-price-limit: "0.05"
```

配置示例

在Kubernetes中，Job负责批量处理短暂的一次性任务。使用抢占式实例运行Job任务的配置示例如下：

1. 准备Job的配置文件，命名为spot_job.yaml。

```
apiVersion: batch/v1
kind: Job
metadata:
 name: pi
spec:
 template:
  metadata:
   annotations:
    k8s.aliyun.com/eci-use-specs: ecs.t5-c1m2.large,2-4Gi
    k8s.aliyun.com/eci-spot-strategy: SpotAsPriceGo  #系统自动出价，跟随当前市场实际价格
  spec:
   containers:
   - name: pi
    image: registry-vpc.cn-beijing.aliyuncs.com/ostest/perl
    command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
   restartPolicy: Never
```

2. 创建Job。

```
kubectl create -f spot_job.yaml
```

3. 查看运行情况。

```
kubectl get pod
```

返回信息中可以看到对应Pod已经成功创建，示例如下：

```
NAME    READY  STATUS    RESTARTS  AGE
pi-frmr8  1/1    Running   0      5s
```

抢占式实例创建成功后，在一个小时的保护期内，即使市场价格高于出价，抢占式实例也不会被释放，您可以在该实例上正常运行业务。超出保护期后，当出价低于市场价格或者资源库存不足，实例将被释放。释放有以下几种方式通知方式：

- 释放前事件通知

  抢占式实例在释放前三分钟左右，会产生释放事件，ECI会将释放事件同步到Kubernetes Events中，您可以通过以下命令查看：

  - 查看Pod详细信息

    ```
    kubectl describe pod pi-frmr8
    ```

    返回信息的Events中可以看到释放事件。示例如下：

    ```
    Events:
     Type    Reason        Age   From      Message
     ----    ------        ----  ----      -------
     Warning  SpotToBeReleased  3m32s  kubelet, eci  Spot ECI will be released in 3 minutes
    ```

  - 查看事件信息

    ```
    kukubectl get events
    ```

    返回信息中可以看到释放事件。示例如下：

    ```
    LAST SEEN  TYPE    REASON        OBJECT      MESSAGE
    3m39s    Warning  SpotToBeReleased   pod/pi-frmr8   Spot ECI will be released in 3 minutes
    ```

- 释放后状态展示

  抢占式实例释放后，实例信息仍会保留，状态变更为Failed，Failed原因为BidFailed。您可以通过以下命令查看：

  - 查看Pod信息

    ```
    kubectl get pod
    ```

    返回信息中可以看到Pod的STATUS为BidFailed。示例如下：

    ```
    NAME    READY  STATUS    RESTARTS  AGE
    pi-frmr8  1/1    BidFailed  0      3h5m
    ```

- 查看Pod详细信息

  ```
  kubectl describe pod pi-frmr8
  ```

  返回信息中可以看到Pod的Status为Failed，Reason为BidFailed。示例如下：

  ```
  Status:       Failed
  Reason:       BidFailed
  Message:      The pod is spot instance, and have been released at 2020-04-08T12:36Z
  ```

实例释放后，Kubernetes中的Job Controller会自动创建新的实例来继续运行任务。您可以通过`kubectl get pod`命令查看Pod，返回示例如下：

```
NAME    READY  STATUS    RESTARTS  AGE
pi-frmr8  1/1   BidFailed  0      4h53m
pi-kp5zx  1/1   Running    0      3h45m
```