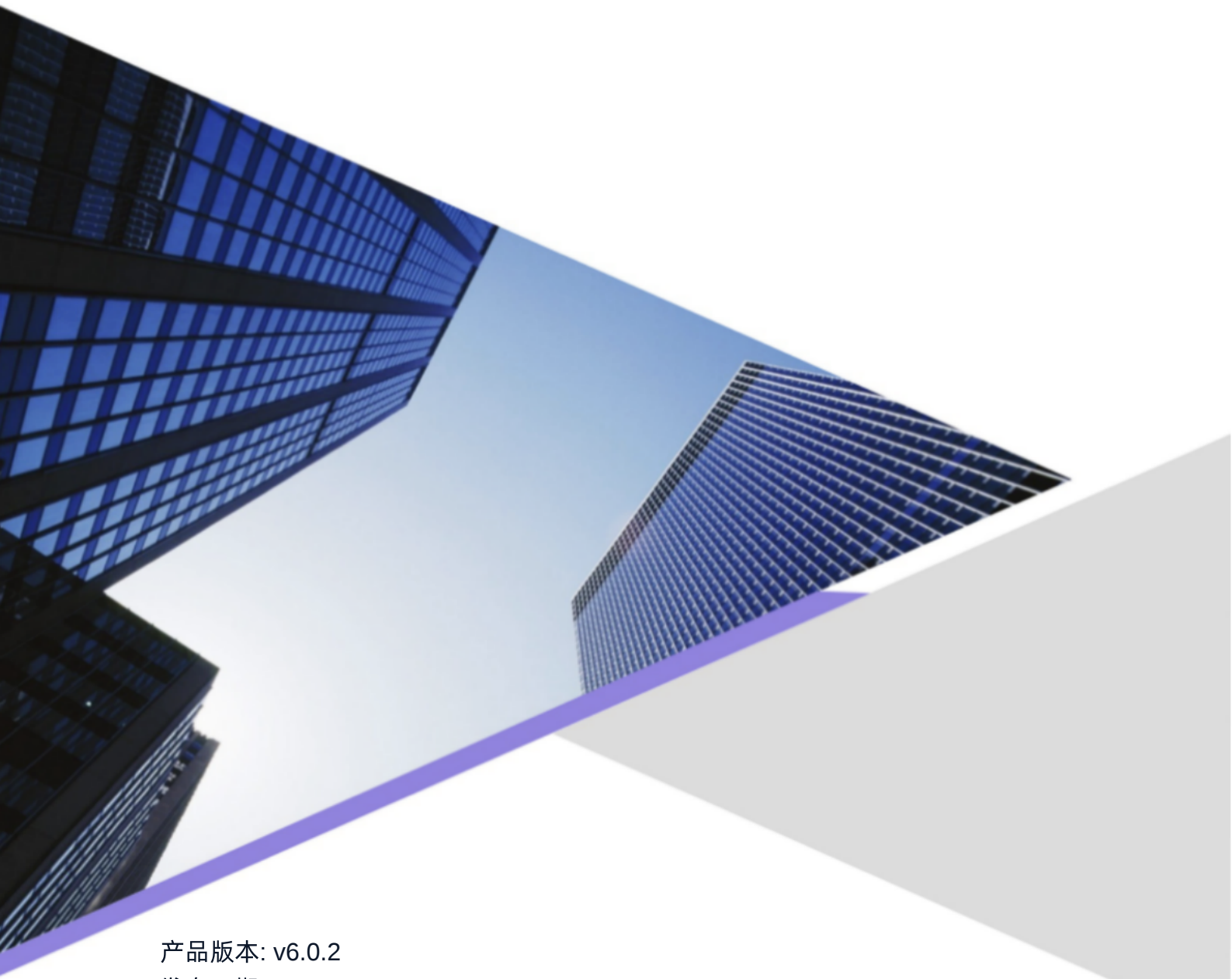
EasyStack
易捷行云

DevOps
使用手册



产品版本: v6.0.2
发布日期: 2024-10-10

目录

1 版本说明

1.1 版本说明书

版本信息

产品名称	产品版本	发布日期
DevOps	V6.0.2	2021-12-03

更新说明

新增功能

- 修复了前端页面兼容性问题。

依赖说明

> - 安装本产品前需确保平台版本为6.0.2。
> - 安装时依赖容器镜像服务V6.0.2，容器应用中心V6.0.2，Kubernetes容器服务V6.0.2。

2 产品介绍

2.1 什么是DevOps

DevOps源自Development（开发）和Operations（运维）的组合，是一种新的软件工程理念，旨在打破传统软件工程方法中“开发->测试->运维”的割裂模式，强调端到端高效一致的交付流程，实现开发和运维的统一。

DevOps云产品，以容器技术的持续集成（CI，Continuous Integration）、持续部署（CD，Continuous Deployment）为基础，面向从源代码获取到应用程序或软件生产上线的全流程，提供运行脚本、构建发布镜像、YAML部署、构建发布Chart模板和Chart模板部署等服务，并通过卡片式的可视化配置页面，提供精益、敏捷、可定制的企业CI/CD流水线创建模式，帮助企业精细化管理交付流程，缩短交付周期，提升交付效率。

产品优势

- **图形化编排**

  通过可视化方式按需定制流水线流程及任务，以便降低使用门槛，并且还可以随时查看流水线执行进度、结果和日志。

- **自动化执行**

  提供手动、定时和事件的触发方式，按照用户定制流程自动执行任务，为项目的持续性交付提供高度自动化能力。

- **无缝集成云原生产品**

  能够与Kubernetes容器服务、容器镜像服务和容器应用中心等云产品集成，提供统一开发体验，降低使用成本。

- **一体化设计**

  DevOps云产品与云基础设施平台实现一体化设计，统一权限、监控、日志、告警等能力。

- **流水线扩展**

  通过自有镜像与运行脚本相互配合使用的方式，实现自定义流水线执行逻辑。

- **低资源成本**

  以容器技术为基础，按需创建、使用和释放底层计算资源，大大提升资源使用效率，降低IT成本。

主要功能

- **关联Kubernetes集群以承载流水线**

  DevOps支持使用Kubernetes容器服务的集群资源，承载流水线的运行。

- **对接源码托管网站以搭建项目**

  DevOps支持对接GitHub和GitLab源码托管网站，直接获取项目源代码，用于项目的持续集成和持续部署。

- **提供容器化场景下的全流程CI/CD能力**

  DevOps拥有容器化场景下的全流程CI/CD能力，提供可视化、可定制的端到端自动交付流水线，实时监控流水线状态，同时集成代码获取、编译构建、部署等任务。您也可以灵活设置定时、提交代码自动触发等多种执行触发方式以满足不同使用场景。

- **自定义流水线执行参数**

  DevOps通过定义静态参数，实现全流程的常量传递。通过定义动态参数，实现不同场景下的任务变量传递。

- **扩展DevOps能力**

  DevOps支持通过将预置好的工具集镜像上传至容器镜像服务云产品后，与流水线的运行脚本类任务相互配合使用，达到扩展DevOps功能的目的。

2.2 使用场景

- **自动构建**

  适用于Web应用前台及后端应用程序的编译构建，在用户自定义的构建环境和构建脚本指令下，基于云端强大的计算能力和网络服务快速把源代码构建为可运行的目标产物，并存放入容器镜像中心。

- **持续交付**

  当需要全新开发应用程序或软件时，DevOps配合Kubernetes容器服务、容器镜像服务、容器应用中心等云产品，以自动化方式实现应用源码的构建、测试和部署，以备随时将应用发布到生产环境中。

- **流水线编排**

  当需要管理多项活动时，DevOps提供可视化的方式按需定制工作流程，自由配置执行阶段，并可以随时查看流水线执行进度、结果和日志等信息，满足CI/CD使用场景下的自动化交付。

2.3 基本概念

- **流水线**

  流水线用于自定义应用程序或软件持续集成和部署的过程，即将整个过程依次定义为几个不同的阶段，并为每个阶段依次定义一组任务，用于帮助开发人员快速交付应用程序或软件。

  DevOps云产品从零开始创建的流水线默认生成“Source”、“Build”、“Test”和“Deploy”四个阶段，可根据客户实际业务需求酌情增减。

- **阶段**

  阶段即流水线的组成内容。一条流水线可以包含多个阶段，各阶段依据流水线中的定义顺序依次执行。

- **任务**

  任务即阶段的组成内容。一个阶段可以包含多个任务，各任务依据流水线中的定义顺序依次执行。当当前阶段的所有任务均执行成功后，才开始执行下一阶段。若当前阶段存在任务执行失败，则将终止流水线运行。

- **代码仓库**

  代码仓库即业务代码所在仓库，用于存储代码项目的源代码，是实现持续交付的原始物料。

2.4 产品获取

前提条件

在执行下述产品获取操作步骤前，请确保以下条件均已满足：

- 已成功获取并安装“容器镜像服务”、“容器应用中心”和“Kubernetes容器服务”云产品。获取并安装云产品的具体操作说明，请参考“产品与服务管理”帮助中的相关内容。
- 如需获取正式版云产品，请提前将已获取的许可文件准备就绪。

操作步骤

1. 获取并安装DevOps云产品。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[产品与服务管理]-[云产品]，进入“云产品”页面获取“DevOps”云产品。具体的操作说明，请参考“产品与服务管理”帮助中“云产品”的相关内容。

2. 访问DevOps服务。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]后，在“DevOps”区域框中，单击“流水线”、“代码仓库”或“集群配置”，即可进入对应页面访问对应服务。

2.5 权限说明

本章节主要用于说明DevOps各功能的用户权限范围。其中，√ 代表该类用户可对云平台内所有项目的操作对象执行此功能，**XX项目** 代表该类用户仅支持对XX项目内的操作对象执行此功能，未标注代表该类用户无权限执行此功能。

功能		云管理员	部门管理员/项目管理员	普通用户
流水线	信息展示	√	仅已加入项目	仅该用户创建对象
	从零开始创建	仅Default/admin项目	仅已加入项目	
	从已有流水线复制	√	仅已加入项目	
	执行/停止	√	仅已加入项目	
	编辑流水线	仅该用户创建对象	仅该用户创建对象	
	编辑触发规则	仅该用户创建对象	仅该用户创建对象	
	高级设置	仅该用户创建对象	仅该用户创建对象	
	激活/停用	√	仅已加入项目	
	删除	√	仅已加入项目	
代码仓库	信息展示	仅该用户创建对象	仅该用户创建对象	仅该用户创建对象
	设置代码仓库			
	注销账户			
集群配置	信息展示	仅Default/admin项目	仅已加入项目	
	添加集群			

功能		云管理员	部门管理员/项目管理员	普通用户
	移除			

2.6 使用限制

- 针对试用版DevOps云产品，全局允许创建的流水线数量上限为10。针对正式版DevOps云产品，该数量上限由许可文件决定。当云产品中流水线数量已达全局创建上限时，请联系云管理员购买并更新云产品许可文件。
- 针对私有GitLab代码仓库，支持对接的版本为“gitlab-9.0.0及以上版本”。

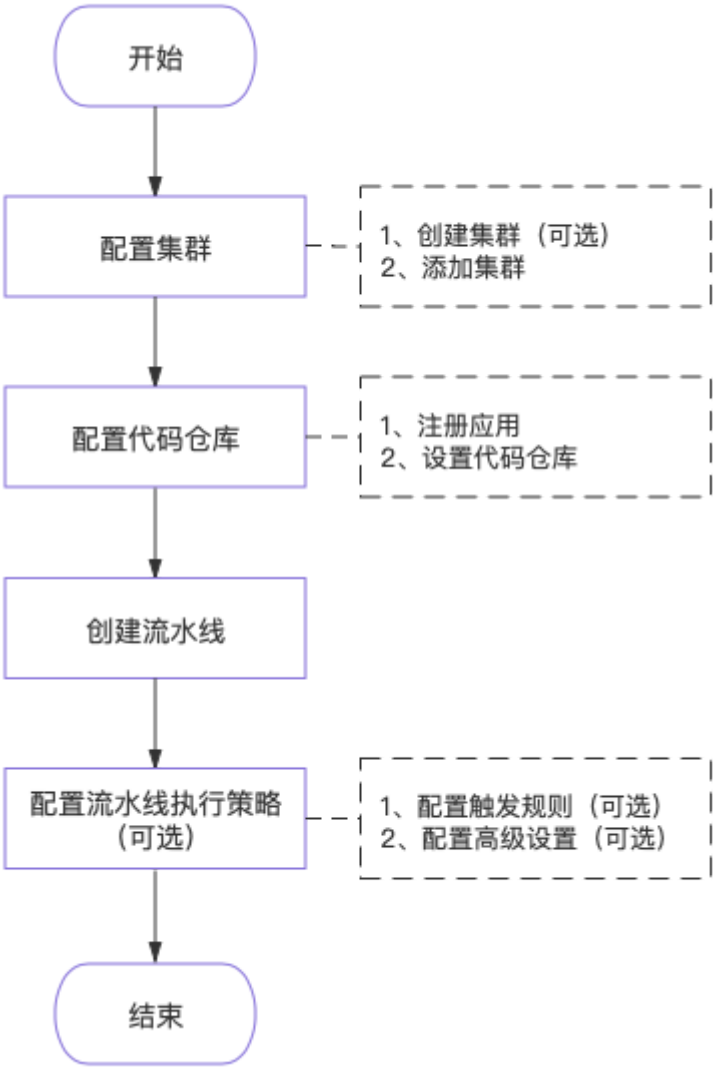
2.7 与其他服务的关系

服务	说明
Kubernetes容器服务	提供承载流水线运行的Kubernetes集群。
容器镜像服务	提供流水线“运行脚本”类任务运行所需的基本镜像文件，并为“构建并发布镜像”类任务提供存放和隔离镜像文件的工作空间。
容器应用中心	为“构建并发布Chart模板”类任务提供Chart模板存储功能，并为“通过Chart模板部署”类任务提供运行所需的Chart模板文件和存放模板实例的空间。

3 快速入门

3.1 操作指引

DevOps云产品的主线使用流程及具体说明如下：



操作流程	描述

操作流程		描述
配置集群	创建集群（可选）	在Kubernetes容器服务中创建承载流水线运行的集群。请根据客户实际业务需求酌情创建。如现有Kubernetes容器集群满足业务需要，可跳过本步骤。
	添加集群	在DevOps服务的集群配置中添加承载流水线运行的集群。
配置代码仓库	注册应用	在代码仓库中注册当前云平台应用，并获取客户端ID和密钥。
	设置代码仓库	设置DevOps服务需要关联的代码仓库，以便构建流水线时选择可构建的代码项目。目前，云平台支持GitHub和GitLab两种代码仓库。
创建流水线		根据实际业务需求自定义交付流水线。
配置流水线执行策略（可选）	配置触发规则（可选）	设置流水线自动执行的触发策略。目前，云平台支持事件触发（提交代码、合并代码和标签事件）和定时触发两类策略。请根据客户实际业务需求酌情配置，如不配置触发规则，也可手动执行流水线。
	配置高级设置（可选）	设置流水线自动执行的高级设置项。目前，云平台仅支持设置流水线超时时间。即如果流水线在设置时间内没有执行完成，将直接终止本次流水线执行任务。请根据客户实际业务需求酌情配置，如不配置超时时间（即超时时间为空时），代表流水线将在14天内持续执行直至成功或最终被自动停止。

3.2 配置集群

本操作用于为流水线配置承载其运行的Kubernetes集群，通过创建集群和添加集群两步骤，实现在指定Kubernetes集群中承载所有DevOps流水线的运行。

创建集群（可选）

本操作用于创建承载流水线运行的Kubernetes集群，请根据客户实际业务需求酌情创建。当现有Kubernetes容器集群满足业务需要时，可跳过本步骤。

1. 创建Kubernetes集群。

    1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[Kubernetes容器服务]，进入“管理视图”的“集群管理”页面。
    2. 在“集群管理”页面中，选择[业务集群]页签后，单击列表上方的 `创建集群` ，进入“创建Kubernetes集群”页面。
    3. 在“创建Kubernetes集群”页面中，配置相关集群参数后，单击 `创建` ，开始创建Kubernetes集群，并关闭“创建Kubernetes集群”页面。其中，“创建Kubernetes集群”页面中各参数的具体说明，请参考“Kubernetes容器服务”帮助中“创建集群”的相关内容。

2. （可选）创建命名空间。

    当流水线中需要创建“通过YAML部署”或“通过Chart模板部署”任务时，请执行本步骤。否则，可直接跳过该步骤。

    1. 在Kubernetes容器服务的“管理视图”页签导航栏中，单击“命名空间”，进入“命名空间”页面。
    2. 在“命名空间”页面中，选择[业务集群]页签后，单击列表上方的 `创建命名空间` ，进入“创建命名空间”页面。
    3. 在“创建命名空间”页面中，配置相关命名空间参数后，单击 `创建命名空间` ，开始创建命名空间，并关闭“创建命名空间”页面。其中，“创建命名空间”页面中各参数的具体说明，请参考“Kubernetes容器服务”帮助中“创建命名空间”的相关内容。

添加集群

本操作用于建立Kubernetes集群与流水线的关联，将该集群用于承载DevOps所有流水线的运行。

> 说明：
> 当成功添加集群后，在该集群下将会自动创建一个名称前缀为 **ecp-devops** 的命名空间，用于运行承载各流水线任务的容器组（Pod）。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[集群配置]，进入“集群配置”页面。

2. 在“集群配置”页面中，单击页面上方的 `添加集群` ，弹出“添加集群”对话框。

3. 在“添加集群”对话框中，选择上述新建的Kubernetes集群或原有用于承载流水线运行的集群后，单击 `确认` ，开始添加Kubernetes集群，并关闭对话框。

3.3 配置代码仓库

本操作用于建立源代码与流水线的关联，通过注册应用和设置代码仓库两步骤，实现在构建流水线时能够选择可构建的代码项目。

注册应用

本操作用于在待设置类型（目前仅支持GitHub和GitLab两类）的代码仓库中注册当前云平台应用，并获取客户端ID和密钥。请根据客户实际业务需求酌情选择代码仓库类型，执行以下注册应用操作。

GitHub操作步骤

1. 登录代码仓库，并进入应用程序注册页面。

   各版本代码仓库的具体操作方式如下：

   - 标准版GitHub（公有）：通过访问 此链接，登录并访问应用程序设置页面后，在应用程序设置页面中，单击“注册新应用”，进入新应用注册页面。
   - 企业版GitHub（私有）：通过企业版GitHub地址访问并登录企业版GitHub，然后依次选择[Settings]-[Applications]，进入应用程序设置页面后，在应用程序设置页面中，单击“注册新应用”，进入新应用注册页面。

2. 在新应用注册页面中，按照如下说明输入当前云平台信息后，单击 `注册应用` ，成功注册云平台应用，并关闭当前页面进入应用详情页面。

参数	说明
应用名称	在代码仓库中，当前云平台应用的识别名称。
主页URL	云平台的主页URL地址。该参数的输入格式如： **https://<云平台登录地址>** 。为方便客户操作并规避操作失误，建议您通过以下方式获取该参数内容：登录云平台，在顶部导航栏中依次选择[产品与服务]-[DevOps]-[代码仓库]后，在“代码仓库”页面中，选择[GitHub]页签后，单击列表上方的 `设置代码仓库` ，进入“设置代码仓库”页面。在该页面中，直接复制“配置GitHub应用”区域框中步骤2的“主页URL”地址即可。

参数	说明
应用描述（可选）	针对当前云平台应用的描述信息。
授权回调URL	云平台的回调URL地址。该参数的输入格式如： **https://<云平台登录地址>/devops/repository/github/redirect** 。为方便客户操作并规避操作失误，建议您通过以下方式获取该参数内容：在云平台“代码仓库”页面的[GitHub]页签中，单击列表上方的 `设置代码仓库` ，进入“设置代码仓库”页面。在该页面中，直接复制“配置GitHub应用”区域框中步骤2的“授权回调URL”地址即可。

3. 在云平台应用的详情页面中，单击“Client secrets”区域框中的 `Generate a new client secret` ，生成一个客户端密钥。此时，请记录并保存当前页面中“Client ID”和“Client secret”的参数值，以备后续设置代码仓库时使用。

GitLab操作步骤

1. 登录代码仓库，并进入应用程序注册页面。

   各版本代码仓库的具体操作方式如下：

   - 标准版GitLab（公有）：通过访问 此链接，登录并直接进入新应用注册页面。
   - 企业版GitLab（私有）：通过企业版GitLab地址访问并登录企业版GitLab，然后依次选择[Settings]-[Applications]，直接进入新应用注册页面。

2. 在新应用注册页面中，按照如下说明输入当前云平台信息后，单击 `注册应用` ，成功注册云平台应用，并关闭当前页面进入应用详情页面。

参数	说明
应用名称	在代码仓库中，当前云平台应用的识别名称。

参数	说明
回调URL	云平台的回调URL地址。该参数的输入格式如： **https://<云平台登录地址>/devops/repository/gitlab/redirect**。为方便客户操作并规避操作失误，建议您通过以下方式获取该参数内容：在云平台“代码仓库”页面的[GitLab]页签中，单击列表上方的 `设置代码仓库` ，进入“设置代码仓库”页面。在该页面中，直接复制“配置GitLab应用”区域框中步骤2的“回调URL”地址即可。
授权Scopes	对当前注册云平台的授权范围。该参数请选择“api”和“read_user”。

3. 在云平台应用的详情页面中，记录并保存当前页面中“Application ID”和“Secret”的参数值，以备后续设置代码仓库时使用。

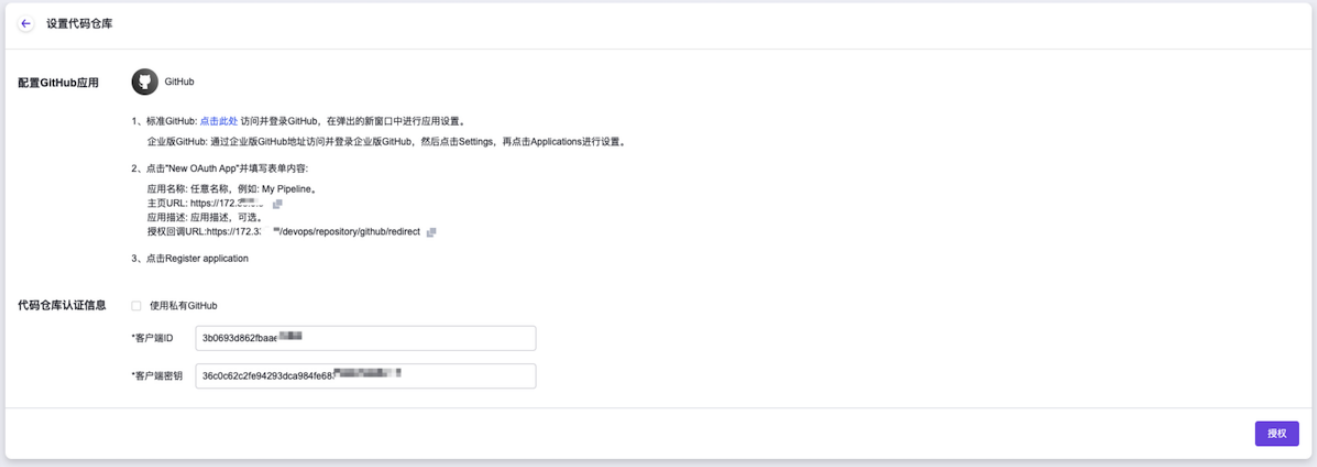
设置代码仓库

本操作用于建立代码仓库与云平台的关联，授权云平台访问代码仓库中的代码项目。

> 警告：
> 在设置代码仓库认证信息时，请确保输入信息为当前云平台的应用注册信息。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[代码仓库]，进入“代码仓库”页面。
2. 在“代码仓库”页面中，根据客户实际代码仓库类型选择[GitHub]或[GitLab]页签后，单击列表上方的 `设置代码仓库` ，进入“设置代码仓库”页面。

3. 在“设置代码仓库”页面中，按照如下说明输入代码仓库认证信息后，单击 `授权` ，跳转至代码仓库的授权页面，单击 `Authorize` ，完成授权并返回云平台设置代码仓库页面。



参数	说明
使用私有GitHub/GitLab（可选）	当需要绑定企业版GitHub或GitLab时，请先勾选“使用私有GitHub/GitLab”，再输入企业版GitHub或GitLab地址。
客户端ID	代码仓库中所注册的云平台应用的客户端ID。在代码仓库中注册云平台应用时，获取到的GitHub仓库的“Client ID”或GitLab仓库的“Application ID”。
客户端密钥	代码仓库中所注册的云平台应用的客户端密钥。在代码仓库中注册云平台应用时，获取到的GitHub仓库的“Client secret”或GitLab仓库的“Secret”。

3.4 创建流水线

本操作用于介绍在云产品初始化状态下，从零开始创建流水线的具体操作步骤，便于客户快速熟悉流水线创建。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。

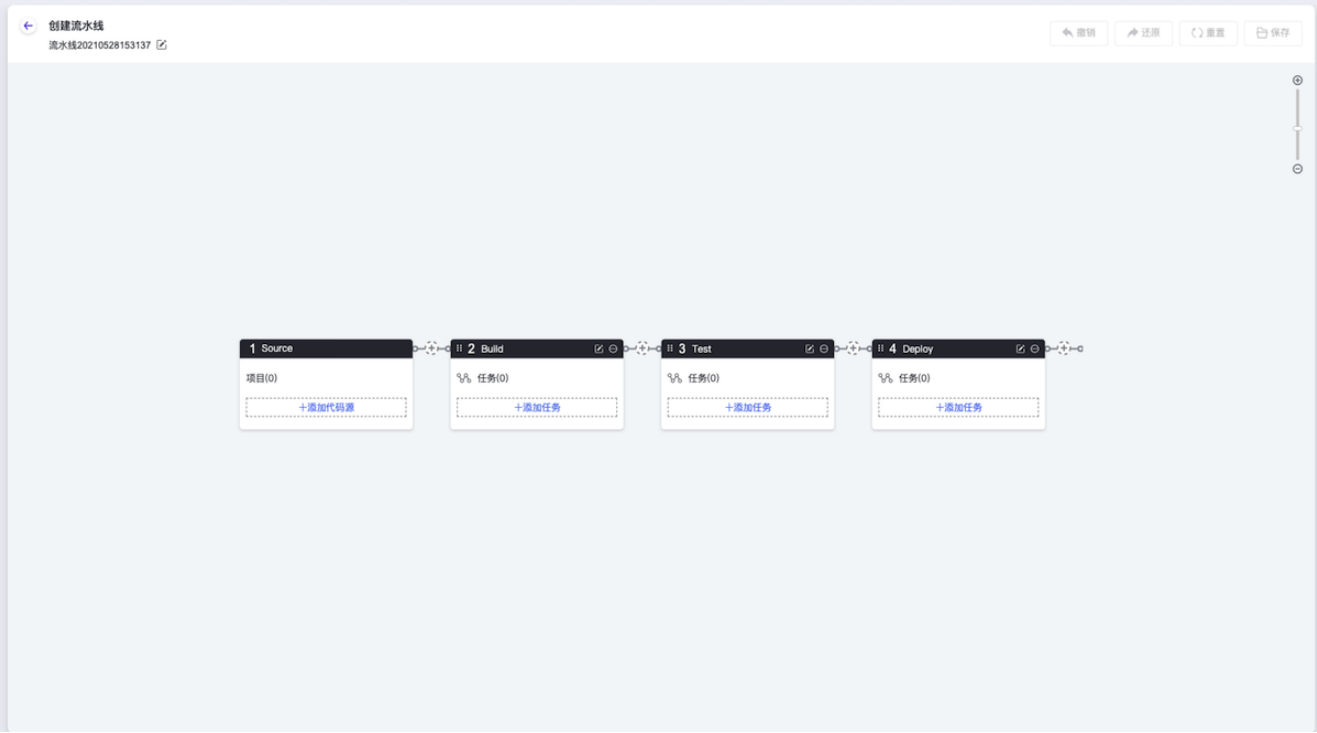
2. 在“流水线”页面中，单击页面上方的 `创建流水线` ，弹出“创建流水线”对话框。

3. 在“创建流水线”对话框中，选择“从零开始创建”后，单击 `创建` ，进入“创建流水线”页面。

4. 在“创建流水线”页面的画布中，默认生成“Source”、“Build”、“Test”和“Deploy”四个阶段，请根据客户实际业务需求酌情设置流水线名称、阶段和各阶段任务后，单击画布右上方的 `保存` ，并在弹出的“保存”对话框中，选择保存方式后，单击 `保存` ，完成流水线创建，并关闭当前页面。

> 说明：

- 在流水线创建过程中，当需要撤销上一步操作时，单击画布右上方的 `撤销` 即可；当需要将上一步撤销内容还原时，单击画布右上方的 `还原` 即可；当需要将画布内容恢复到本次初始化编辑状态时，单击画布右上方的 `重置` 即可。
- 流水线保存方式包括“仅保存”和“保存并立即”执行两种。具体区别如下：
  - 当用户选择“仅保存”状态时，将只保存该流水线的相关配置信息，并显示在流水线列表中，而不执行任何其它操作。
  - 当用户选择“保存并立即执行”时，将不仅保存该流水线的相关配置信息，并显示在流水线列表中，还会立即执行该流水线。



针对流水线中的代码源、阶段和任务三类对象，云平台均提供添加、编辑和删除等操作，各操作的具体操作说明如下表所示：

对象	操作项	说明
代码源	添加代码源	在代码源卡片中，单击 `添加代码源` ，弹出“添加代码源”对话框。在该对话框中，输入代码源信息后，单击 `保存` ，保存项目的代码源设置，并关闭对话框。

对象	操作项	说明
	编辑代码源	在代码源卡片中，单击编辑代码源图标（项目标题栏右侧第一个图标），弹出“编辑代码源”对话框。在该对话框中，编辑代码源信息后，单击 保存 ，保存项目的代码源设置，并关闭对话框。
	删除代码源	在代码源卡片中，单击更多图标（项目标题栏右侧第二个图标），并在下拉菜单中选择“删除代码源”后，将直接删除当前代码源项目卡片。
阶段	添加阶段	在当前阶段的卡片右侧，单击“+”，将直接在该阶段后新增一个新阶段的卡片。
	编辑阶段	在待操作阶段的卡片中，单击编辑阶段图标（阶段标题栏右侧第一个图标），弹出“编辑阶段”对话框。在该对话框中，输入阶段名称并选择阶段内任务执行顺序后，单击 保存 ，保存新阶段设置，并关闭对话框。
	复制阶段	在待操作阶段的卡片中，单击更多图标（阶段标题栏右侧第二个图标），并在下拉菜单中选择“复制阶段”后，将直接在当前阶段后，复制出一个与当前阶段内容一致的新阶段卡片。
	删除阶段	在待操作阶段的卡片中，单击更多图标（阶段标题栏右侧第二个图标），并在下拉菜单中选择“删除阶段”后，将直接删除当前阶段卡片。
	调整阶段顺序	将鼠标移动至阶段标题栏的名称前方后，再按住鼠标左键直接拖动待操作阶段的卡片至所需位置，即可调整阶段顺序。
任务	添加任务	在待操作阶段的卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，按照如下说明配置任务信息后，单击 保存 ，完成任务创建，并关闭对话框。目前，云平台支持运行脚本、构建并发布镜像、通过YAML部署、构建并发布Chart模板和通过Chart模板部署五种任务类型，各任务类型的具体配置参数说明如下： * 运行脚本 镜像地址：该任务运行所需的基础镜像。 镜像版本：该任务运行所需镜像的版本。

对象	操作项	说明
		脚本：该任务需要运行的具体脚本内容。 环境变量：该任务运行环境中的变量信息。该参数值以键值对形式保存，请逐个对应输入变量名和变量值。 * 构建并发布镜像 Dockerfile文件路径：该任务中用于构建镜像的文本文件在代码仓库中的路径。 镜像名称：该任务所构建并发布的镜像文件的名称。 镜像版本：该任务所构建并发布的镜像文件的版本。该参数值支持输入固定数值，也支持输入$引用变量。 工作空间：用于存放和隔离该任务所构建并发布的镜像文件。如需新建工作空间，请单击该任务对话框中的“创建工作空间”，跳转至“容器镜像服务”的“工作空间”页面进行创建。 * 通过YAML部署 YAML文件路径：该任务运行时所用YAML文件在代码仓库中的路径。 集群：该任务所部署的应用程序或软件的目标集群。 命名空间：该任务所部署的应用程序或软件的目标命名空间。 * 构建并发布Chart模板 Chart目录：该任务运行时所用Chart.yaml文件在代码仓库中的路径。 应用模板名称：该任务所构建并发布的模板文件的名称。 模板版本：该任务所构建并发布的模板文件的版本。 * 通过Chart模板部署 部署实例名称：该任务所部署的模板实例的名称。 应用模板名称：用于部署该任务模板实例的Chart模板的名称。 模板版本：用于部署该任务模板实例的Chart模板的版本。 集群：该模板实例的部署集群。 命名空间：该模板实例的部署命名空间。 参数：该任务运行过程中的参数信息。 > 说明： > > 在此类任务的配置过程中，“应用模板名称”和“模板版本”支持“选择”和“输入”两种配置方式。请参考下述说明，酌情选择配置方式：

对象	操作项	说明
		> > * 当待部署的应用模板为“容器应用中心”云产品中已有的自有模板时，请选择“选择”配置方式，依次选择本流水线任务所需的应用模板名称和版本。 > * 当待部署的应用模板为本流水线前述阶段新构建发布的Chart模板时，请选择“输入”配置方式，依次输入前述构建发布阶段所配置的Chart模板的名称和版本。
	编辑任务	在待操作任务的卡片中，单击编辑任务图标（任务标题栏右侧第一个图标），弹出“编辑任务”对话框。在该对话框中，编辑任务信息后，单击 `保存` ，保存任务设置，并关闭对话框。
	复制任务	在待操作任务的卡片中，单击更多图标（任务标题栏右侧第二个图标），并在下拉菜单中选择“复制任务”后，将直接在当前任务后，复制出一个与当前任务内容一致的新任务卡片。
	删除任务	在待操作任务的卡片中，单击更多图标（任务标题栏右侧第二个图标），并在下拉菜单中选择“删除任务”后，将直接删除当前任务卡片。
	调整任务顺序	在待操作任务所在阶段的卡片中，将鼠标移动至任务标题栏的名称前方后，再按住鼠标左键直接拖动待操作任务的卡片至所需位置，即可调整阶段内任务的顺序。任务仅支持在所在阶段内调节，不支持跨阶段调节。

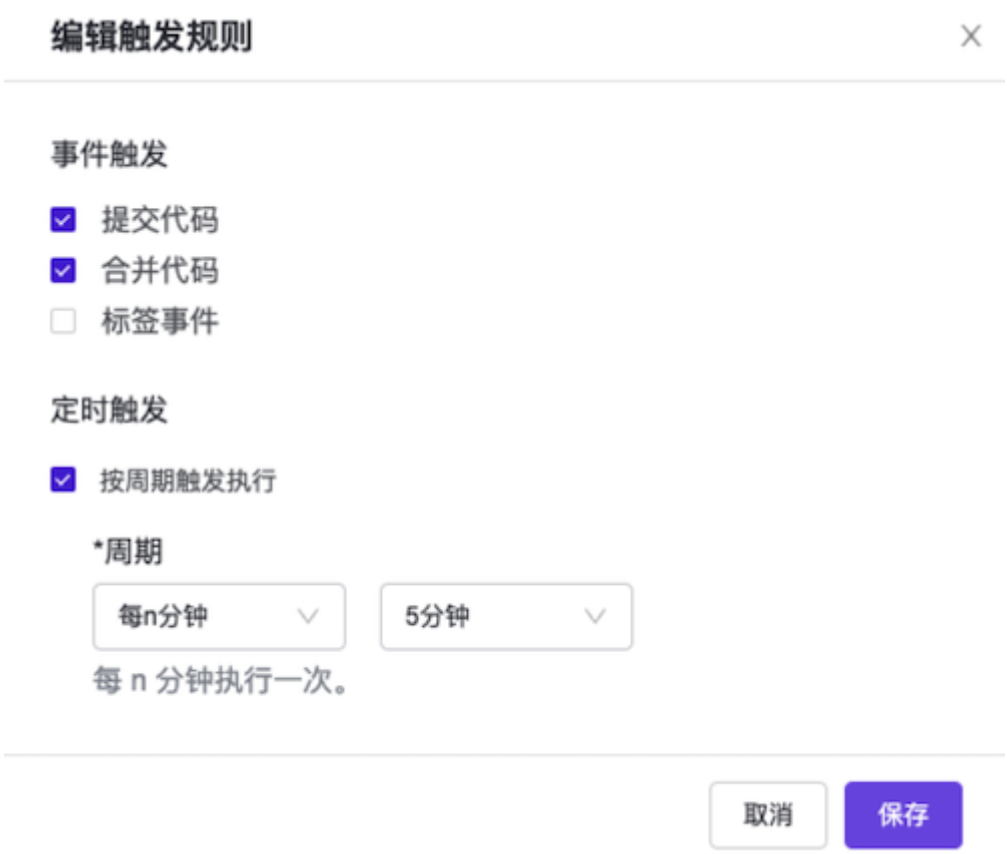
3.5 配置流水线执行策略（可选）

本操作用于配置流水线的执行策略，通过配置触发规则和配置高级设置两项内容，实现通过指定规则自动触发流水线执行，并在执行超时时停止执行。

配置触发规则（可选）

当前云平台中，触发流水线执行的规则包括手动触发和自动触发两种，本操作用于配置自动触发流水线执行的规则，请根据客户实际业务需求酌情配置。当不执行本操作时，即表示需要手动触发流水线执行。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。
2. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择“编辑触发规则”，弹出“编辑触发规则”对话框。
3. 在“编辑触发规则”对话框中，依据客户实际业务需求设置触发规则后，单击 `保存` ，完成自动触发规则设置，并关闭对话框。



参数	说明
触发规则	自动触发流水线执行的规则。当前云平台支持事件触发和定时触发两种，可同时设置也可分别设置。 * 事件触发：发生指定事件规则即可触发流水线执行。当前云平台支持提交代码、合并代码和标签事件三种事件规则，可同时设置也可分别设置，请根据客户实际业务需求酌情选择。 > 警告： > > * 当选择此类触发规则时，请确保代码仓库与DevOps云产品（即云平台外部访问IP地址）之间的网络互通。 > > * 当选择此类触发规则时，请确保在 配置代码仓库 操作中登录Git代码仓库的账号有Webhook操作权限。 > > 由于Git代码仓库对Webhook的数量设有上限，而Git事件将触发Webhook，所以当不同流水线均使用相同Git项目作为代码源并均设置事件触发时，若到达该上限将无法再为此类流水线新增事件触发规则。 * 定时触发：按照设定周期定时触发流水线执行。当前云平台支持每天、每n小时、每n分钟、每周和每月五种定时规则。 当选择“按周期触发执行”时，若“周期”选择“每n小时”或“每n分钟”，则具体执行时间点将从下一个n的整数倍时间开始。例如： * 在2021-03-25 14:53:21配置触发规则的“周期”输入框依次为“每n分钟”、“2分钟”，则具体执行时间将是2021-03-25 14:54:00、2021-03-25 14:56:00、2021-03-25 14:58:00等依次类推。 * 在2021-03-25 14:53:21配置触发规则的“周期”输入框依次为“每n小时”、“2小时”、“28”，则具体执行时间将是2021-03-25 16:28:00、2021-03-25 18:28:00、2021-03-25 20:28:00等依次类推。

配置高级设置（可选）

当前云平台中，高级设置项仅支持设置流水线超时时间，请根据客户实际业务需求酌情配置。当不配置超时时间（即超时时间为空）时，即表示流水线将在14天内持续执行直至成功或最终被自动停止。

> 说明：
> 流水线超时时间，主要用于控制各阶段中所有任务的实际执行时间不超过设置时间。该时间不包括向业务集群请求资源的时间，即当业务集群中无可用资源时，流水线将持续请求资源，而不会受超时时间限制导致执行终止。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。
2. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择“高级设置”，弹出“高级设置”对话框。
3. 在“高级设置”对话框中，依据客户实际业务需求输入超时时间后，单击 `保存` ，完成超时时间设置，并关闭对话框。

4 用户指南

4.1 流水线

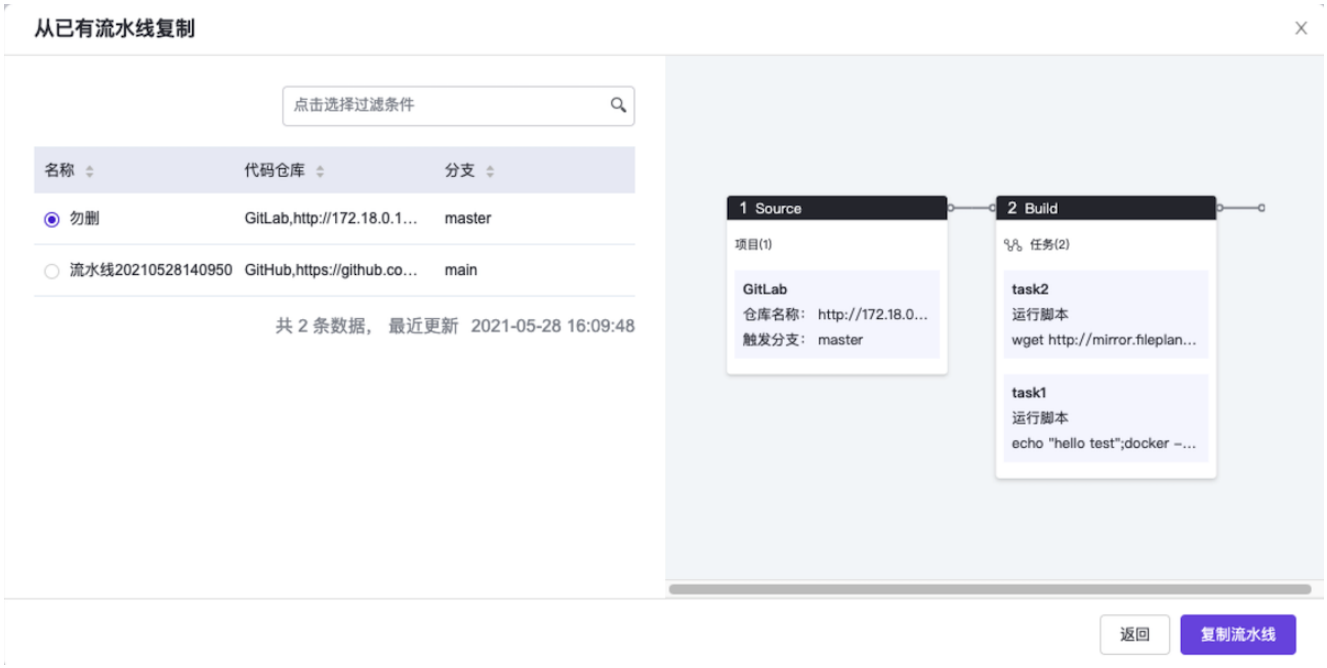
本章节主要介绍在流水线页面中，针对流水线的一系列运维管理操作，如：查看详情、执行、停止和编辑流水线等。其中，在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，即可进入“流水线”页面。

从已有流水线复制

1. 在“流水线”页面中，单击页面上方的 `创建流水线` ，弹出“创建流水线”对话框。

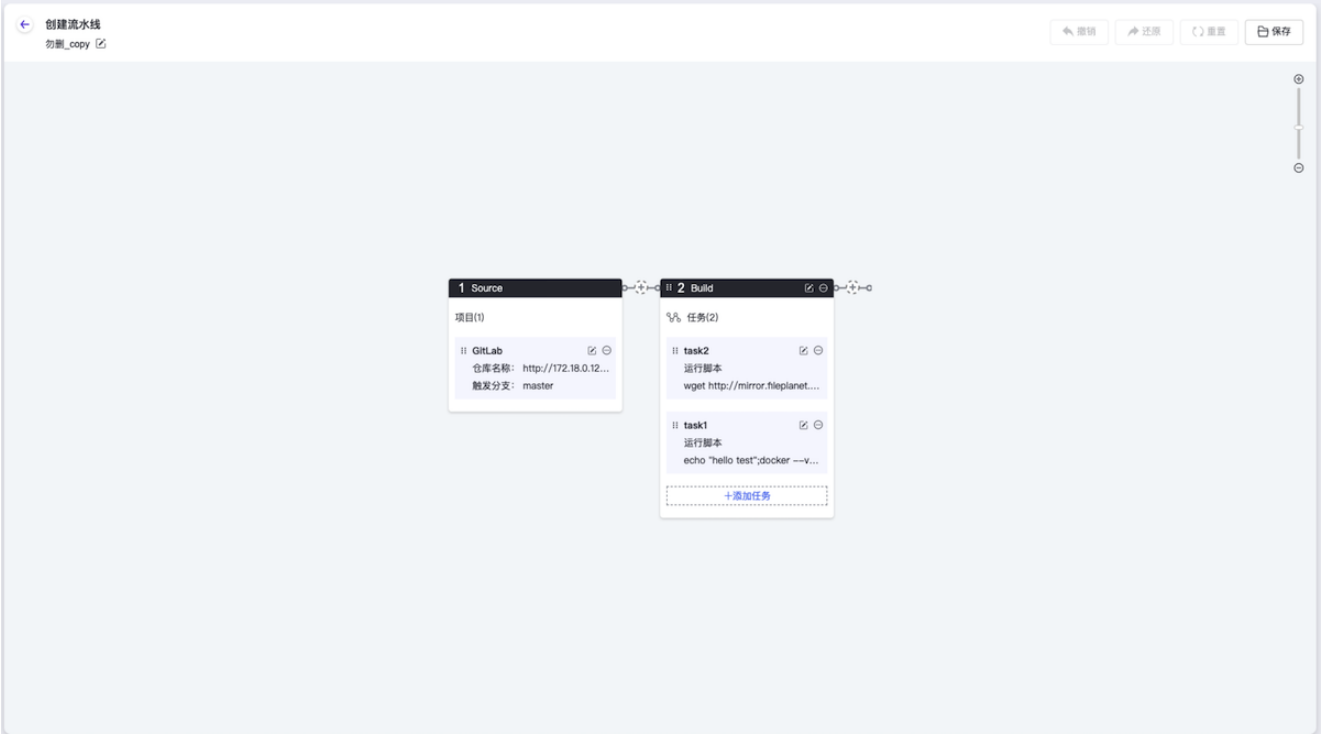
2. 在“创建流水线”对话框中，选择“从已有流水线复制”后，单击 `创建` ，弹出“从已有流水线复制”对话框。

3. 在“从已有流水线复制”对话框中，选择待复制流水线后，单击 `复制流水线` ，进入“创建流水线”页面。



4. 在“创建流水线”页面的画布中，将直接显示所复制的流水线，请根据客户实际业务需求酌情编辑流水线名称、阶段和各阶段任务后，单击画布右上方的 `保存` ，并在弹出的“保存”对话框中，选择保存方式后，单击 `保存` ，完成流水线创建，并关闭当前页面。

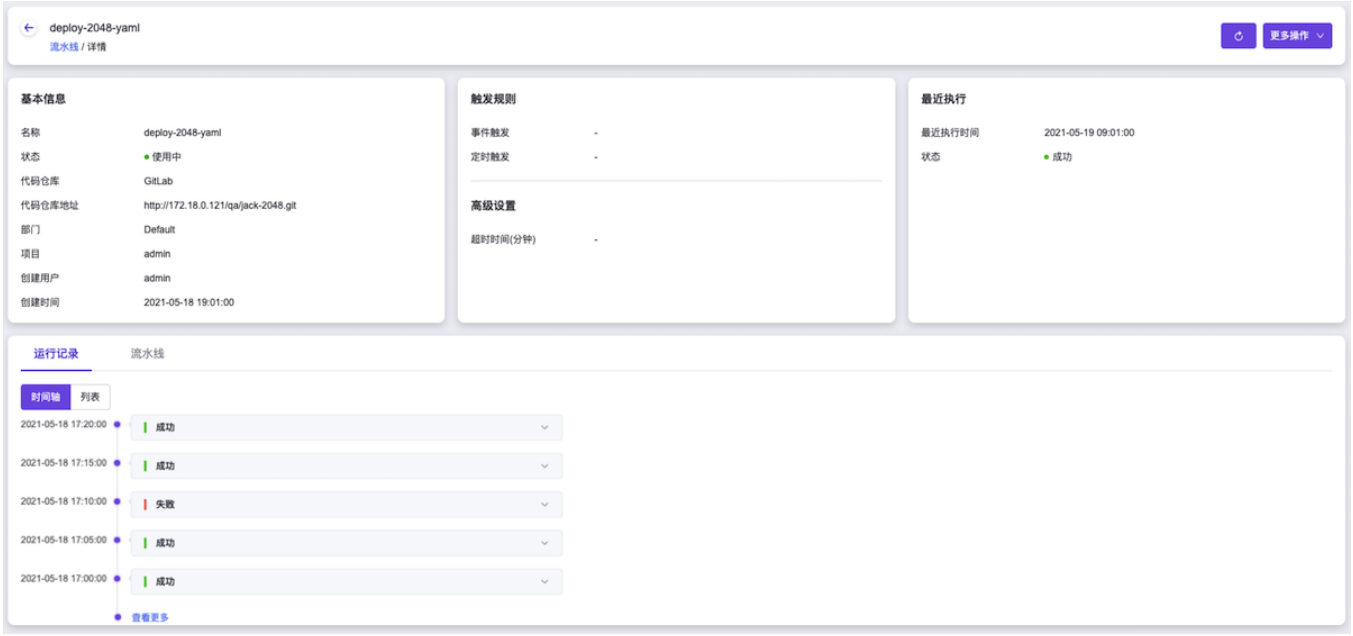
> 说明：
> - “从已有流水线复制”的流水线创建方式，将会同时复制该已有流水线的所有阶段配置（包含所有任务配置）和超时时间设置。
> - 在流水线创建过程中，当需要撤销上一步操作时，单击画布右上方的 `撤销` 即可；当需要将上一步撤销内容还原时，单击画布右上方的 `还原` 即可；当需要将画布内容恢复到本次初始化编辑状态时，单击画布右上方的 `重置` 即可。
> - 流水线保存方式包括“仅保存”和“保存并立即”执行两种。具体区别如下：
>   - 当用户选择“仅保存”方式时，将只保存该流水线的相关配置信息，并显示在流水线列表中，而不执行任何其它操作。
>   - 当用户选择“保存并立即执行”方式时，将不仅保存该流水线的相关配置信息，并显示在流水线列表中，还会立即执行该流水线。



针对流水线中代码源、阶段和任务三类对象的具体操作说明，请参考 创建流水线。

查看详情

在“流水线”页面中，单击流水线名称 ，可进入其详情页面。在详情页面中，可查看该流水线的基本信息、触发规则、高级设置、最近执行、运行记录和流水线图示信息。

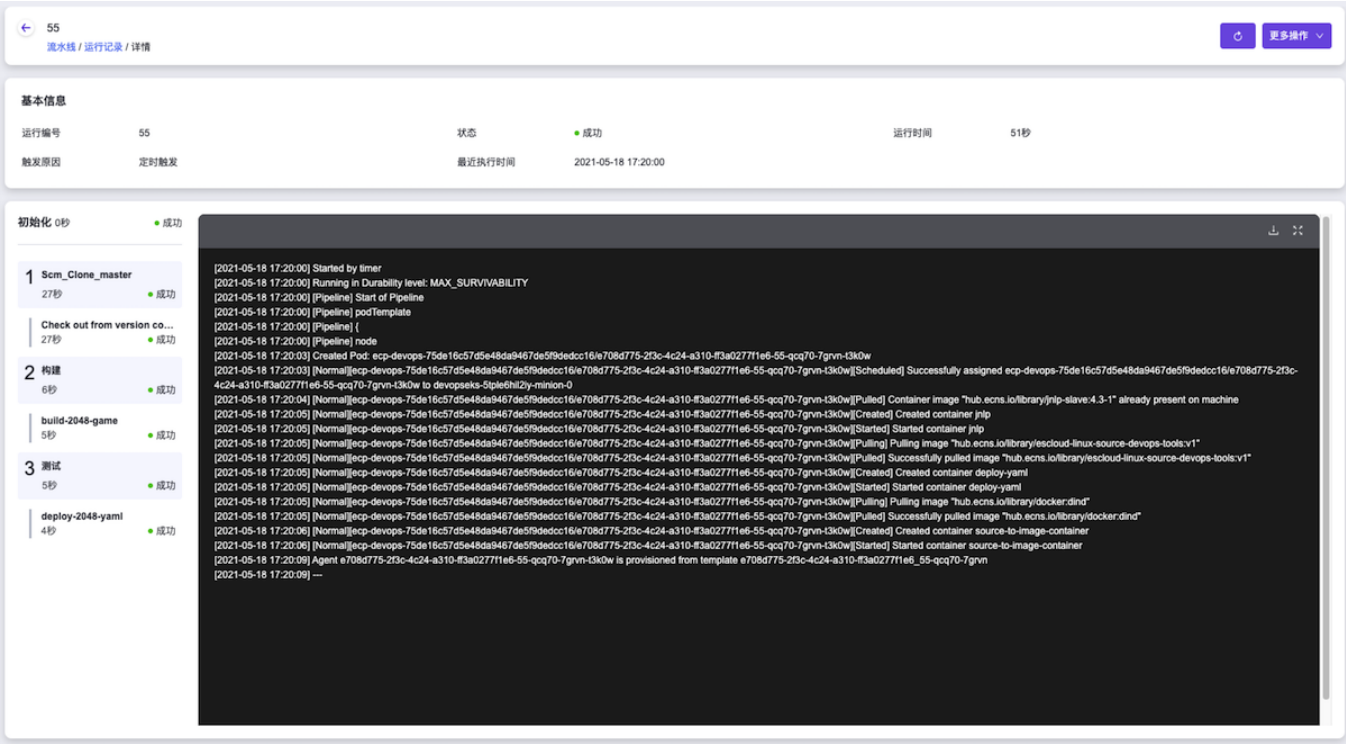


在流水线详情页面的[运行记录]页签中，还支持对流水线的运行记录执行查看运行详情、执行和停止等操作。具体操作说明如下：

查看运行详情

在流水线详情页面的[运行记录]页签中，可以以“时间轴”或“列表”形式查看流水线执行信息。此外，在此页签的任意展示形式下，单击运行编号，均可进入此次运行记录的详情页面，查看此次运行的详情信息，包括基本信息以及各阶段、各任务的执行状态和各任务的实时执行日志。

> 说明：
> - 当流水线执行失败时，可以通过查看此次运行记录的详情页面，获取各任务执行日志信息以便故障定位。若由于页面当前显示的日志信息有限而无法进行故障定位时，请单击此页面上方的 `更多操作` ，在下拉列表中选择“下载全量日志”，下载全量日志进行故障定位。
> - 在流水线详情页面的[运行记录]页签中，最多仅支持展示最近50条运行记录。



执行

本操作用于使流水线按照指定运行记录当时编排的配置信息重新执行。

1. 在流水线详情页面的[运行记录]页签中，选择“时间轴”形式后展开待操作运行记录卡片并单击该卡片中的 `执行` ，或选择“列表”形式后勾选待操作运行记录并单击列表上方的 `执行` ，弹出“重新执行流水线”提示框。
2. 在“重新执行流水线”提示框中，单击 `重新执行` ，重新执行本次运行过程，并关闭提示框。

停止

本操作用于中止处于“执行中”状态的指定运行记录。

1. 在流水线详情页面的[运行记录]页签中，选择“时间轴”形式后展开待操作运行记录卡片并单击该卡片中的 `停止` ，或选择“列表”形式后勾选待操作运行记录并单击列表上方的 `停止` ，弹出“停止流水线”提示框。
2. 在“停止流水线”提示框中，单击 `停止` ，中止本次运行过程，并关闭提示框。

执行/停止

本操作中的执行操作用于使流水线按照当前编排的配置信息执行。停止操作用于中止该流水线当前最新的一次运行过程。

1. 在“流水线”页面中，单击待操作流水线所在行的 `执行` 或 `停止` ，弹出“执行/停止流水线”提示框。
2. 在“执行/停止流水线”提示框中，单击 `执行` 或 `停止` ，完成流水线操作，并关闭提示框。

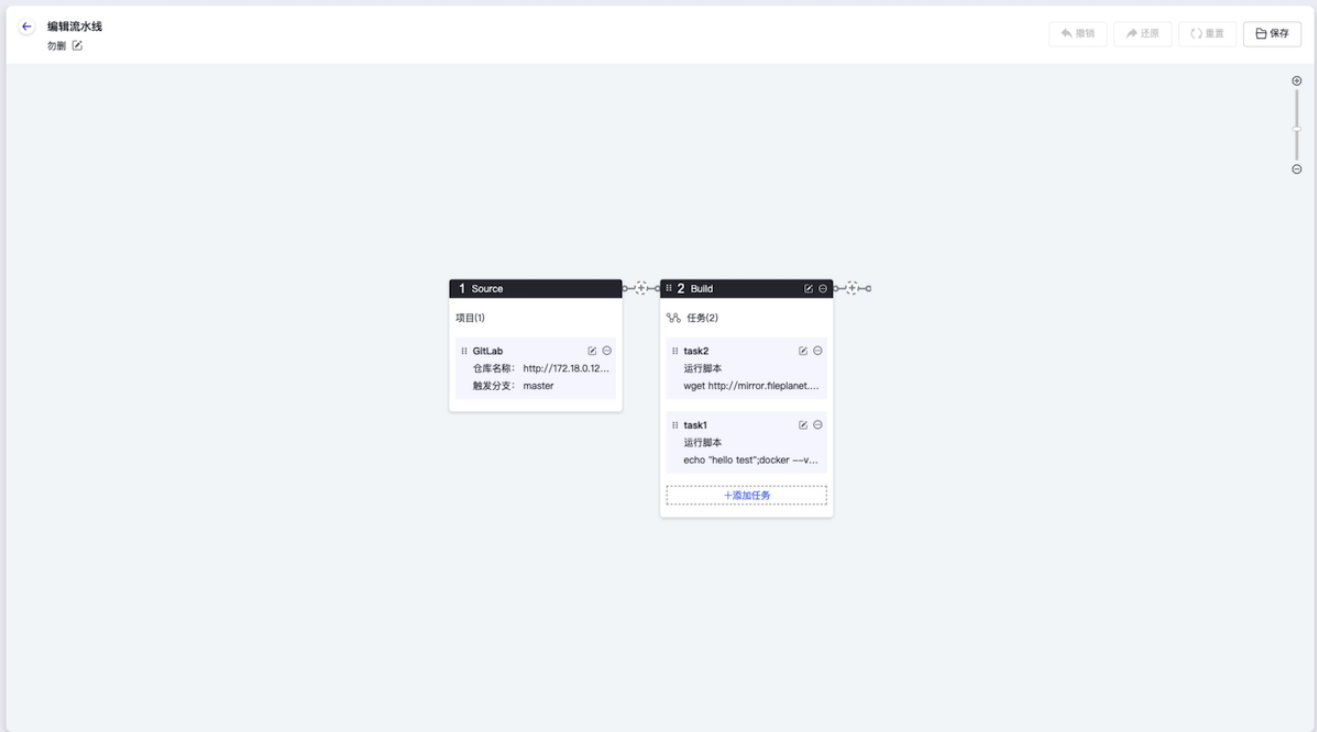
编辑流水线

1. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择“编辑流水线”，进入“编辑流水线”页面。

2. 在“编辑流水线”页面的画布中，请根据客户实际业务需求酌情编辑流水线内容后，单击画布右上方的 `保存` ，并在弹出的“保存”对话框中，选择保存方式后，单击 `保存` ，完成流水线编辑，并关闭当前页面。

> 说明：
> - 在流水线编辑过程中，当需要撤销上一步操作时，单击画布右上方的 `撤销` 即可；当需要将上一步撤销内容还原时，单击画布右上方的 `还原` 即可；当需要将画布内容恢复到本次初始化编辑状态时，单击画布右上方的 `重置` 即可。
> - 流水线保存方式包括“仅保存”和“保存并立即”执行两种。具体区别如下：
>   - 当用户选择“仅保存”方式时，将只保存该流水线的相关配置信息，并显示在流水线列表中，而不执行任何其它操作。

- 当用户选择“保存并立即执行”方式时，将不仅保存该流水线的相关配置信息，并显示在流水线列表中，还会立即执行该流水线。



针对流水线中代码源、阶段和任务三类对象的具体操作说明，请参考 创建流水线。

编辑触发规则

1. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择“编辑触发规则”，弹出“编辑触发规则”对话框。
2. 在“编辑触发规则”对话框中，依据客户实际业务需求设置触发规则后，单击 `保存` ，完成自动触发规则设置，并关闭对话框。


编辑触发规则
事件触发
提交代码
合并代码
标签事件
定时触发
按周期触发执行
*周期
每n分钟
5分钟
每 n 分钟执行一次。
取消
保存

参数	说明

参数	说明
触发规则	自动触发流水线执行的规则。当前云平台支持事件触发和定时触发两种，可同时设置也可分别设置。 * 事件触发：发生指定事件规则即可触发流水线执行。当前云平台支持提交代码、合并代码和标签事件三种事件规则，可同时设置也可分别设置，请根据客户实际业务需求酌情选择。 > 警告： > > * 当选择此类触发规则时，请确保代码仓库与DevOps云产品（即云平台外部访问IP地址）之间的网络互通。 > > * 当选择此类触发规则时，请确保在 配置代码仓库 操作中登录Git代码仓库的账号有Webhook操作权限。 > > 由于Git代码仓库对Webhook的数量设有上限，而Git事件将触发Webhook，所以当不同流水线均使用相同Git项目作为代码源并均设置事件触发时，若到达该上限将无法再为此类流水线新增事件触发规则。 * 定时触发：按照设定周期定时触发流水线执行。当前云平台支持每天、每n小时、每n分钟、每周和每月五种定时规则。 当选择“按周期触发执行”时，若“周期”选择“每n小时”或“每n分钟”，则具体执行时间点将从下一个n的整数倍时间开始。例如： * 在2021-03-25 14:53:21配置触发规则的“周期”输入框依次为“每n分钟”、“2分钟”，则具体执行时间将是2021-03-25 14:54:00、2021-03-25 14:56:00、2021-03-25 14:58:00等依次类推。 * 在2021-03-25 14:53:21配置触发规则的“周期”输入框依次为“每n小时”、“2小时”、“28”，则具体执行时间将是2021-03-25 16:28:00、2021-03-25 18:28:00、2021-03-25 20:28:00等依次类推。

高级设置

说明：

- 流水线超时时间，主要用于控制各阶段中所有任务的实际执行时间不超过设置时间。该时间不包括向业务集群请求资源的时间，即当业务集群中无可用资源时，流水线将持续请求资源，而不会受超时时

> 间限制导致执行终止。
> - 当不配置超时时间（即超时时间为空）时，即表示流水线将在14天内持续执行直至成功或最终被自动停止。

1. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择“高级设置”，弹出“高级设置”对话框。
2. 在“高级设置”对话框中，依据客户实际业务需求输入超时时间后，单击 `保存` ，完成超时时间设置，并关闭对话框。

激活/停用

1. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择 `激活` 或 `停用` ，弹出“激活/停用流水线”提示框。
2. 在“激活/停用流水线”提示框中，单击 `激活` 或 `停用` ，完成流水线操作，并关闭提示框。

删除

1. 在“流水线”页面中，单击待操作流水线所在行的 `更多` ，并在下拉列表中选择 `删除` ，弹出“删除流水线”提示框。
2. 在“删除流水线”提示框中，单击 `删除` ，删除流水线，并关闭提示框。

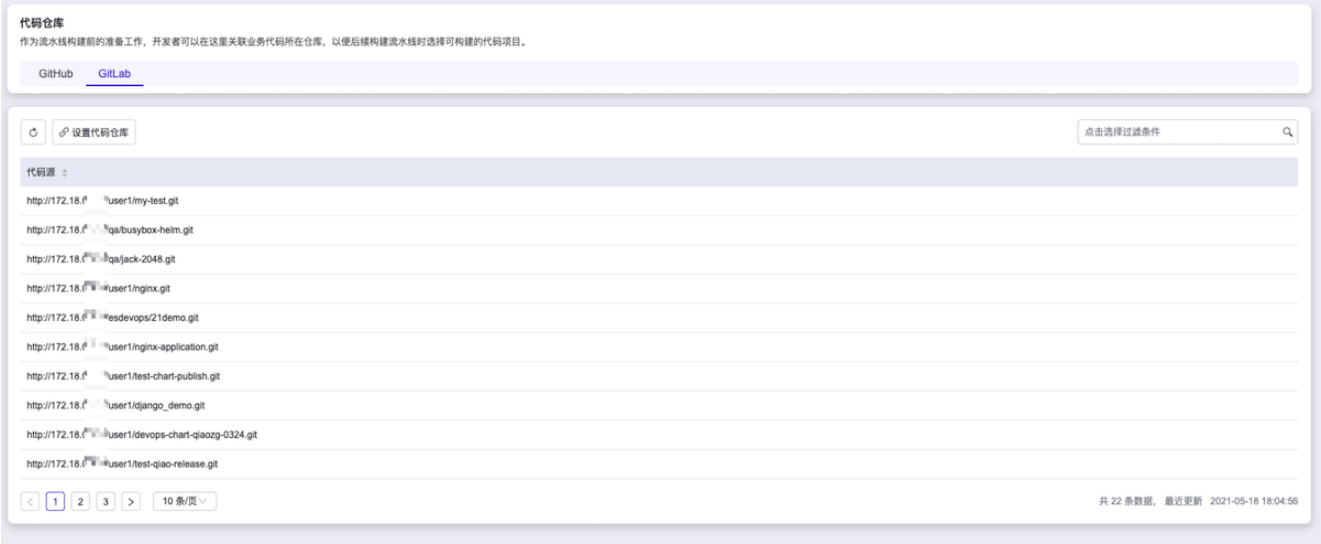
4.2 代码仓库

本章节主要介绍在代码仓库页面中，针对代码仓库的一系列运维管理操作，如：注销账户。其中，在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[代码仓库]，即可进入“代码仓库”页面。

注销

在执行注销操作前，针对使用该代码仓库的所有流水线，请先编辑其源代码项目所在仓库为其他代码仓库或直接执行删除操作。

1. 在“代码仓库”页面中，根据客户实际代码仓库类型选择[GitHub]或[GitLab]页签后，单击列表上方的 `设置代码仓库` ，进入“设置代码仓库”页面。



2. 在“设置代码仓库”页面中，单击 `注销` ，弹出“注销”提示框。

3. 在“注销”提示框中，单击 `确认` ，注销该类型代码仓库的账号，并关闭提示框。

4.3 集群配置

本章节主要介绍在集群配置页面中，针对集群的一系列运维管理操作，如：移除。其中，在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[集群配置]，即可进入“集群配置”页面。

移除

1. 在“集群配置”页面中，单击待操作集群所在行的 `移除` ，弹出“移除集群”提示框。

2. 在“移除集群”提示框中，单击 `移除` ，移除承载流水线运行的集群，并关闭提示框。

> 警告：
> 当集群中存在流水线时，该操作将导致已有流水线无法继续正常工作，请谨慎操作。

5 最佳实践

5.1 通过YAML部署自定义构建镜像

背景描述

通过DevOps云产品，可以创建从源代码获取到项目构建、测试和部署上线的全流程自动交付流水线，大大缩短交付周期，提升交付效率。本文将以2048游戏应用程序为例，介绍如何快速通过DevOps流水线在构建并发布应用程序镜像后，通过YAML部署该应用程序。

本实践方案中，流水线各项规划信息规划如下：

流水线-YAML部署2048游戏镜像	
Source阶段	项目代码使用 **jack-2048.git** 仓库下的 **master** 分支代码
Build阶段	通过“构建并发布镜像”类型任务构建2048游戏镜像 * Dockerfile文件路径：名为 **Dockerfile** 的文件，其直接放置在代码仓库根目录下，方便任务获取 * 镜像名称：jack-2048 * 镜像版本：alpha-${BUILD_ID} * 工作空间：devops
Deploy阶段	通过“通过YAML部署”类型任务部署2048游戏 * YAML文件：名为2048_env.yaml的文件。其直接放置在代码仓库根目录下，方便任务获取 * 访问域名：2048.example.cn

前提条件

- DevOps流水线需要预先配置承载其运行的Kubernetes集群，具体步骤请参考 配置集群。
- DevOps流水线需要预先关联应用程序源代码的代码仓库，具体步骤请参考 配置代码仓库。

- 本DevOps流水线需要预先制作用于构建镜像的Dockerfile文本文件，并放置在代码仓库的根目录下。本实践方案中，根据规划信息Dockerfile文件的内容如下：

```
FROM hub.ecns.io/devops/jack-2048:latest

COPY 2048 /var/lib/nginx/html/

COPY nginx.conf /etc/nginx/nginx.log.conf

COPY nginx.8880.conf /etc/nginx/nginx.8880.conf

EXPOSE 80

CMD ["nginx", "-c", "/etc/nginx/nginx.8880.conf"]
```

- 本DevOps流水线需要预先制作用于YAML部署的 **2048_env.yaml** 文件，并放置在代码仓库的根目录下。本实践方案中，根据规划信息2048_env.yaml文件的内容如下。其中，**image: hub.ecns.io/devops/jack-2048:alpha-${BUILD_ID}** 参数配置中，**devops** 为流水线“构建”阶段所选择的工作空间，**jack-2048** 为流水线“构建”阶段所构建镜像的名称，**alpha-${BUILD_ID}** 为流水线“构建”阶段所构建镜像的版本。**host: 2048.example.cn** 参数配置中，**2048.example.cn** 为流水线“部署”阶段所规划的访问域名。

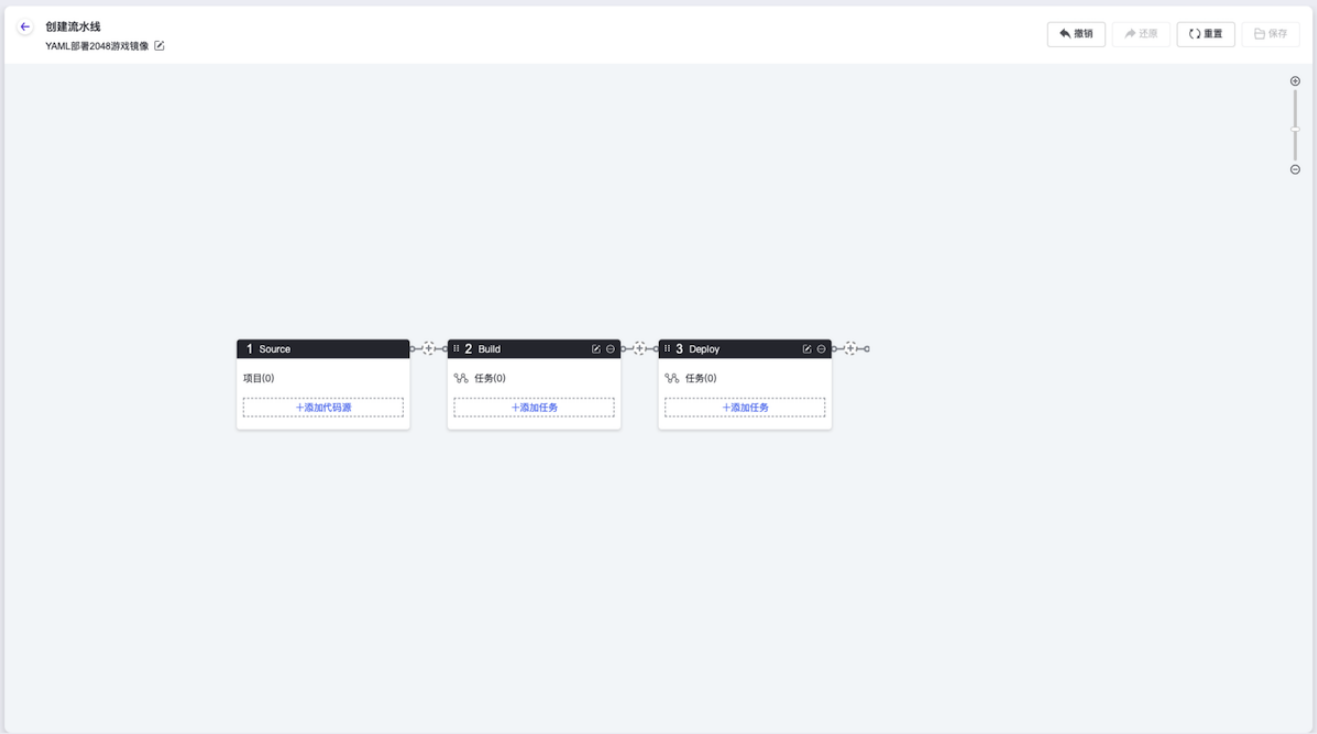
```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jack-2048
spec:
  selector:
    matchLabels:
      application: jack-2048
  strategy:
    rollingUpdate:
      maxSurge: 3
      maxUnavailable: 70%
    type: RollingUpdate
  template:
    metadata:
      labels:
        application: jack-2048
```

```
    spec:
      containers:
      - image: hub.ecns.io/devops/jack-2048:alpha-${BUILD_ID}
        imagePullPolicy: IfNotPresent
        name: jack-2048
        ports:
        - containerPort: 80
          protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: svc-2048
  labels:
    application: jack-2048
spec:
  ports:
  - name: port-2048
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    application: jack-2048
  type: ClusterIP
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ing-2048
spec:
  rules:
  - host: 2048.example.cn
    http:
      paths:
      - backend:
          serviceName: svc-2048
          servicePort: 80
```

操作步骤

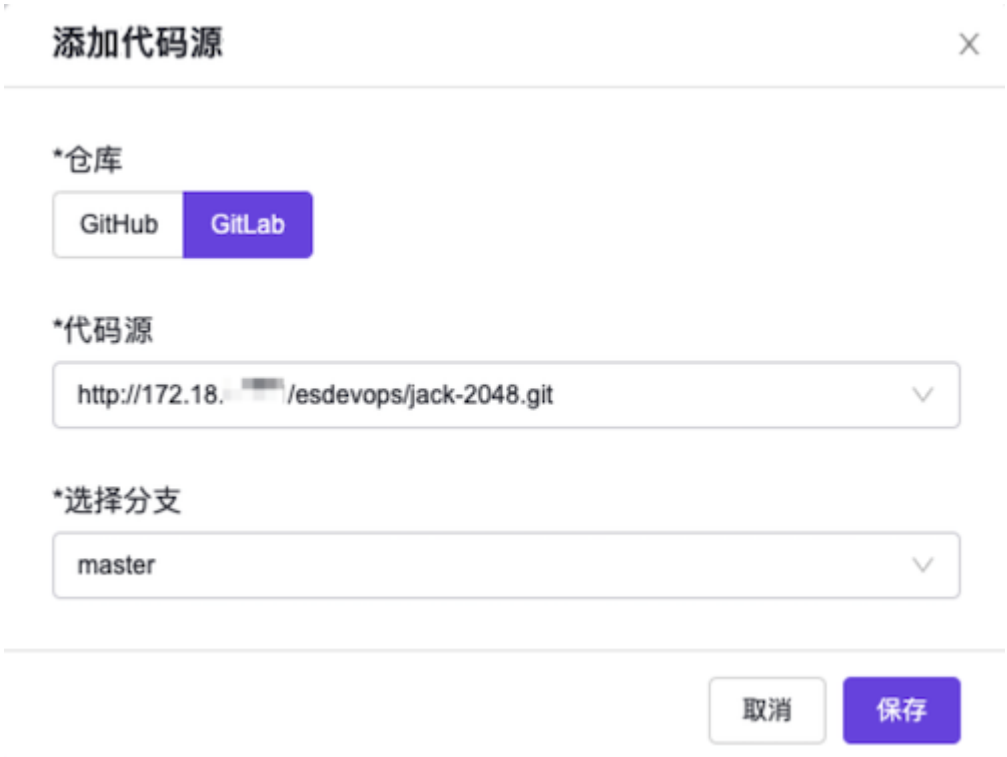
1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。
2. 在“流水线”页面中，单击页面上方的 `创建流水线` ，弹出“创建流水线”对话框。
3. 在“创建流水线”对话框中，选择“从零开始创建”后，单击 `创建` ，进入“创建流水线”页面。

4. 在“创建流水线”页面的画布中，根据本次业务需求配置“Source”、“Build”和“Deploy”阶段后，依次在各阶段完成以下任务配置。



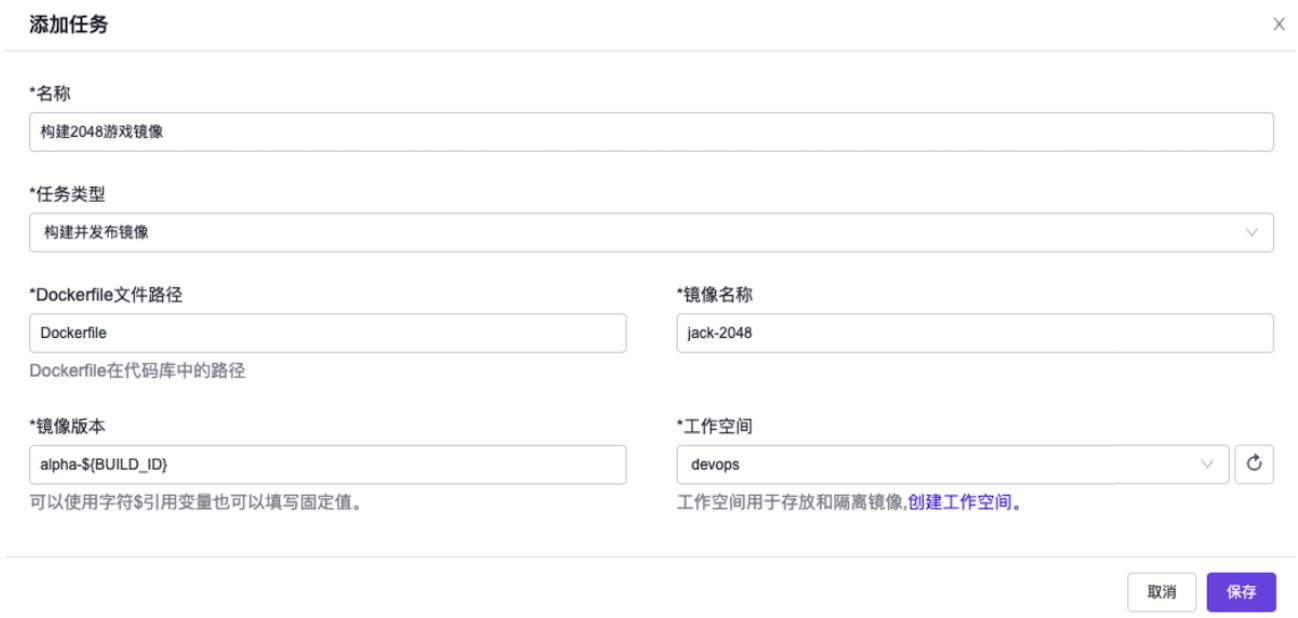
1. 在“Source”阶段添加项目代码源。

   在当前画布的“Source”卡片中，单击``添加代码源``，弹出“添加代码源”对话框。在该对话框中，配置代码源信息后，单击 保存 ，保存项目的代码源设置，并关闭对话框。



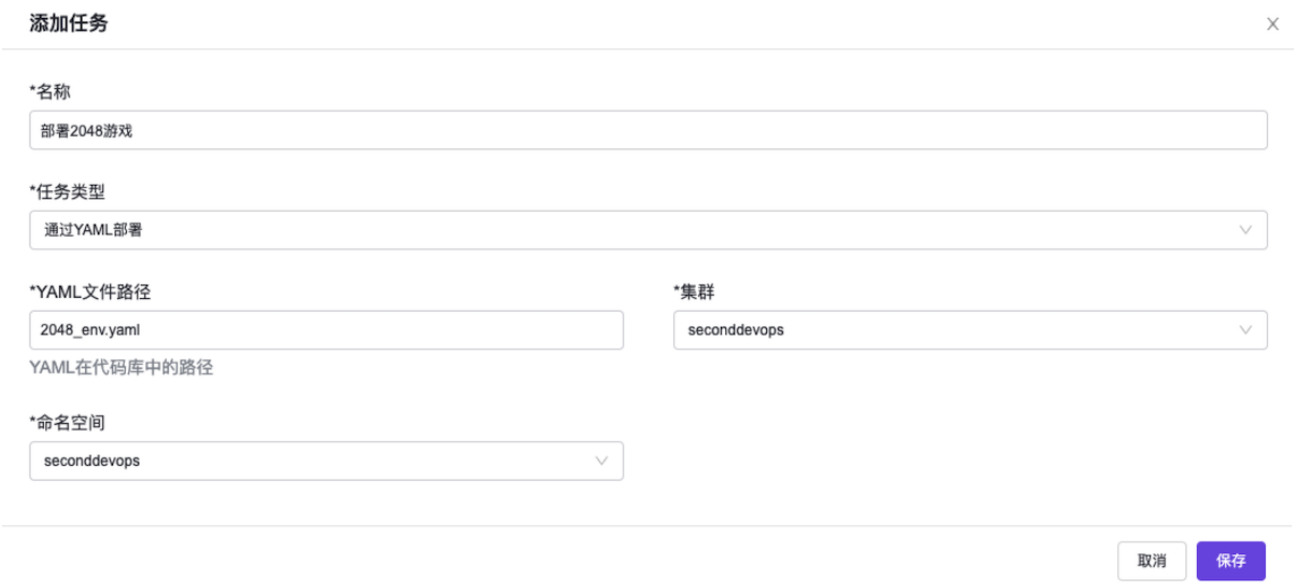
2. 在“Build”阶段添加“构建并发布镜像”任务。

在当前画布的“Build”卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“构建并发布镜像”，“Dockerfile文件路径”输入“Dockerfile”，“镜像名称”输入“jack-2048”，“镜像版本”输入“alpha-${BUILD_ID}”，“工作空间”选择“devops”，并配置名称后，单击 保存 ，完成任务创建，并关闭对话框。



3. 在“Deploy”阶段添加“通过YAML部署”任务。

   在当前画布的“Deploy”卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“通过YAML部署”，“YAML路径”输入“2048_env.yaml”，并配置名称和应用程序的部署目标集群及命名空间后，单击 保存 ，完成任务创建，并关闭对话框。



5. 在“创建流水线”页面的画布中，单击画布右上方的 保存 后，在弹出的“保存”对话框中，选择保存方式后，单击 保存 ，完成流水线创建，并关闭当前页面。

6. 执行流水线。

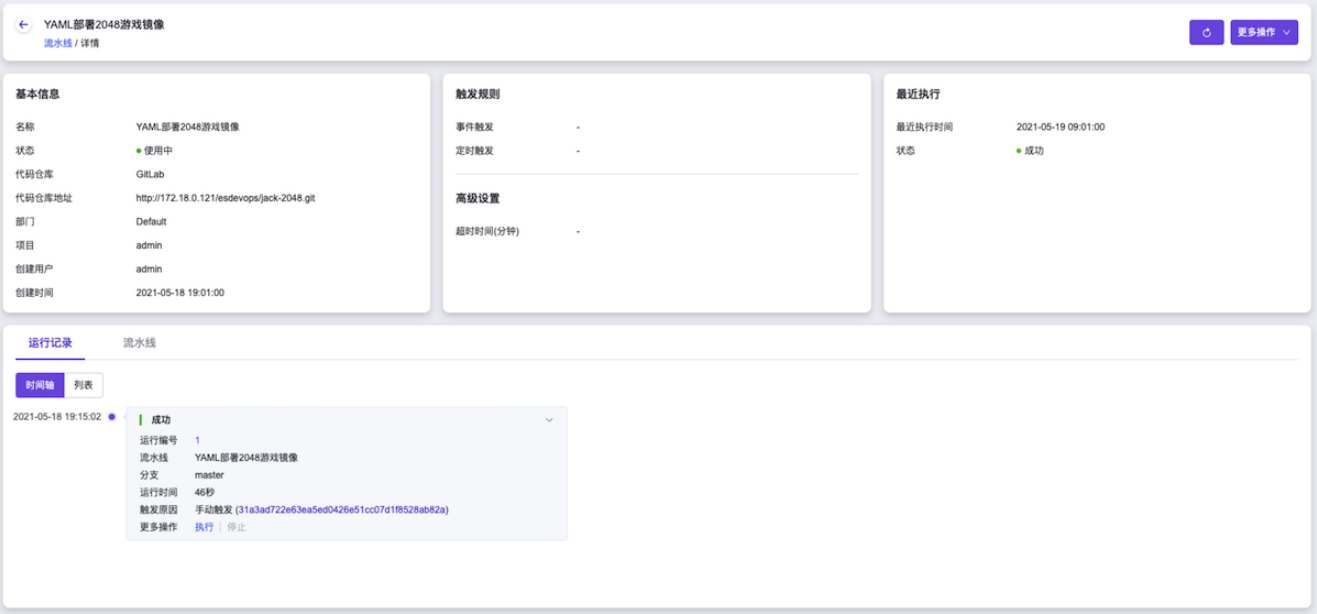
   本实践方案中以手动触发方式为例，触发流水线执行。如需配置流水线自动触发，请参考 配置流水线执行策略（可选）。

   1. 在“流水线”页面中，单击上述流水线所在行的 执行 ，弹出“执行流水线”提示框。
   2. 在“执行流水线”提示框中，单击 执行 ，执行该流水线，并关闭提示框。

结果验证

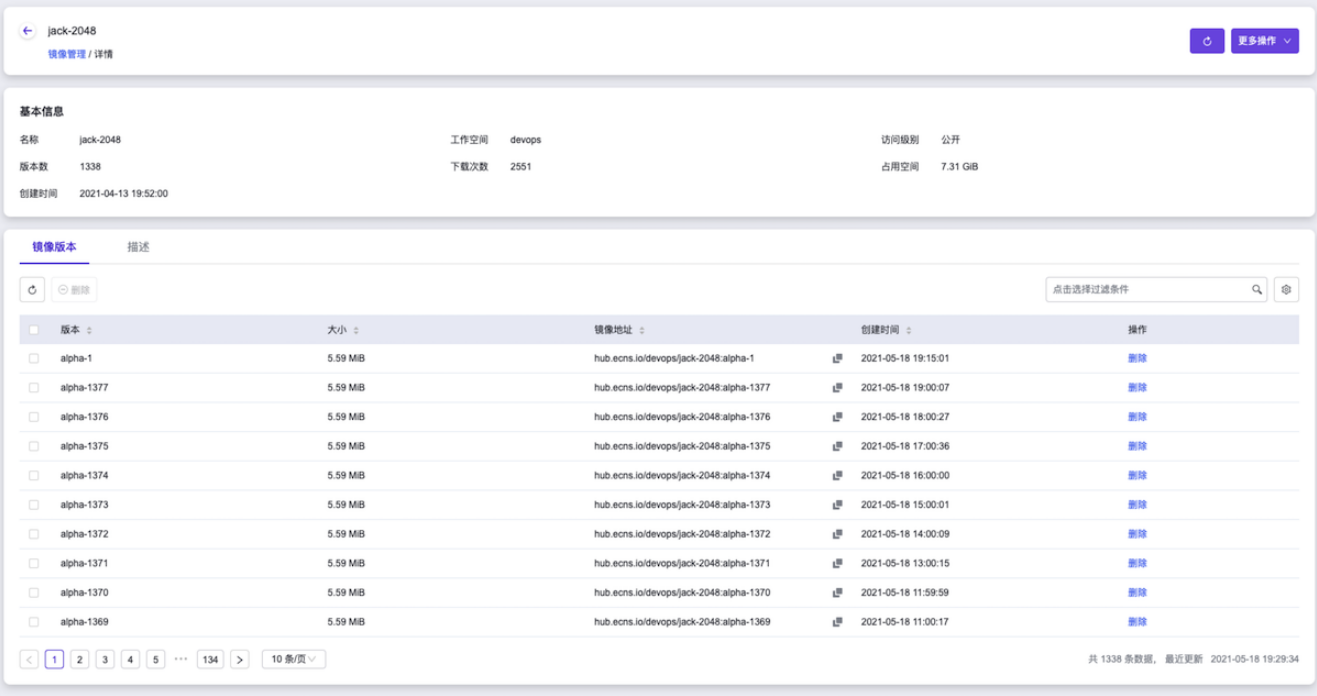
1. 确认流水线成功执行。

   在“流水线”页面中，单击上述流水线名称 ，进入其详情页面。在详情页面的[运行记录]页签中，确认该流水线执行成功并记录此次“运行编号”。



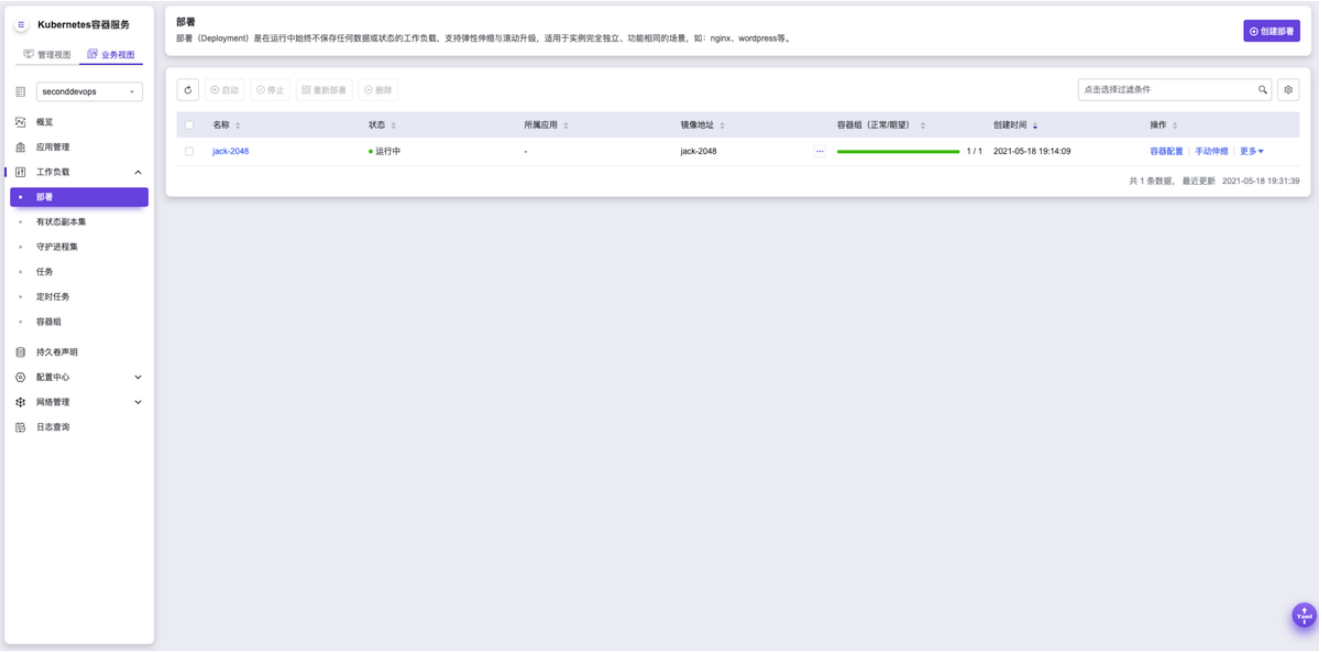
2. 确认镜像成功构建。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[容器镜像服务]，进入“镜像管理”页面。在该页面中，确认已生成名为“jack-2048”的镜像文件。然后，单击此镜像文件名称，进入其详情页面后，在[镜像版本]页签中，确认该镜像文件的生成版本为“alpha-<运行编号>”。



3. 确认应用程序成功部署。

在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[Kubernetes容器服务]，进入Kubernetes容器服务页面后，再在该页面的左侧导航栏中，先选择“业务视图”以及该应用程序的所在项目、部署集群和命名空间，再选择[工作负载]-[部署]，进入部署页面。在该页面中，确认已生成名为“jack-2048”的部署且状态为“运行中”。



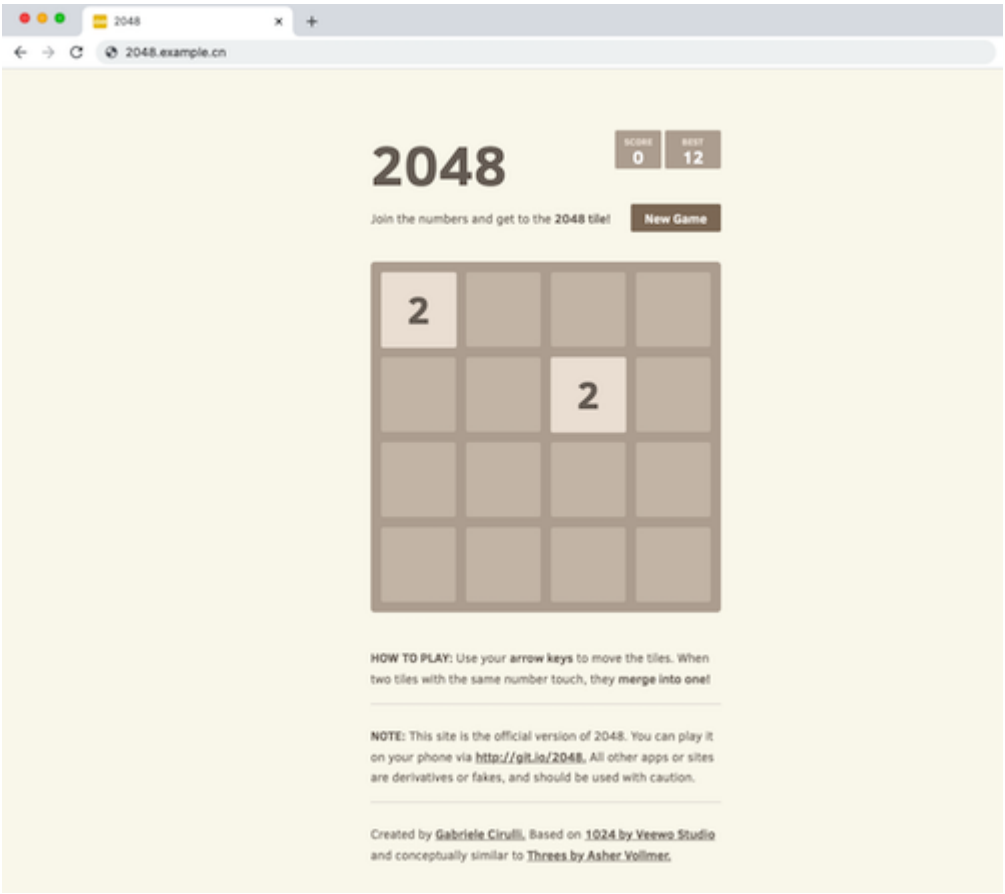
4. 确认应用程序成功访问。

   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[Kubernetes容器服务]，进入“管理视图”的“集群管理”页面。在该页面中，查看并记录该应用程序所在Kubernetes集群的IP地址，即该Kubernetes集群所在行中，API Server列https://后的IP地址。

   2. 在本地计算机的 **hosts** 文件中，添加该应用程序所在Kubernetes集群IP地址与YAML文件定义域名的访问映射。

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
172.18.(        2048.example.cn
```

   3. 在本地计算机的浏览器地址栏中输入YAML文件定义域名，访问部署的应用程序。


2048
2048.example.cn
2048
SCORE 0
BEST 12
Join the numbers and get to the 2048 tile!
New Game
2
2
HOW TO PLAY: Use your arrow keys to move the tiles. When two tiles with the same number touch, they merge into one!
NOTE: This site is the official version of 2048. You can play it on your phone via http://gt.io/2048. All other apps or sites are derivatives or fakes, and should be used with caution.
Created by Gabriele Cirulli. Based on 1024 by Veewo Studio and conceptually similar to Threes by Asher Vollmer.

5.2 通过Chart部署自定义构建镜像和Chart模板

背景描述

通过DevOps云产品，可以创建从源代码获取到项目构建、测试和部署上线的全流程自动交付流水线，大大缩短交付周期，提升交付效率。本文将以nginx应用程序为例，介绍如何快速通过DevOps流水线在构建并发布应用程序镜像后，先打包发布Chart模板，再通过此Chart模板部署该应用程序。

本实践方案中，流水线各项规划信息规划如下：

流水线-Chart部署nginx镜像和Chart模板	
Source阶段	项目代码使用 **nginx.git** 仓库下的 **master** 分支代码
Build阶段	通过“构建并发布镜像”类型任务构建nginx镜像 * Dockerfile文件路径：名为Dockerfile的文件，其直接放置在代码仓库根目录下，方便任务获取 * 镜像名称：nginx * 镜像版本：6.0.1-alpha.${BUILD_ID} * 工作空间：devops
Publish阶段	通过“构建并发布Chart模板”类型任务发布nginx模板 * Chart目录：nginx。即名为Chart.yaml的文件放置在代码仓库的nginx目录下 * 应用模板名称：nginx * 模板版本：6.0.1-alpha.${BUILD_ID}
Deploy阶段	通过“通过Chart模板部署”类型任务部署nginx应用程序 * 部署实例名称：nginx * 应用模板名称：nginx（与发布阶段的应用模板名称保持一致） * 模板版本：6.0.1-alpha.${BUILD_ID}（与发布阶段的模板版本保持一致） * 访问域名：nginx.example.cn

前提条件

- DevOps流水线需要预先配置承载其运行的Kubernetes集群，具体步骤请参考 配置集群。

- DevOps流水线需要预先关联应用程序源代码的代码仓库，具体步骤请参考 配置代码仓库。
- 本DevOps流水线需要预先制作用于构建镜像的Dockerfile文本文件，并放置在代码仓库的根目录下。本实践方案中，根据规划信息Dockerfile文件的内容如下：

```
FROM alpine:latest

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tuna.tsinghua.edu.cn/g'
/etc/apk/repositories \
    && apk --update add nginx \
    && mkdir -p /run/nginx

ADD default.conf /etc/nginx/http.d/
ADD source/html /var/lib/nginx/html/

EXPOSE 80

ENTRYPOINT [ "nginx", "-g", "daemon off;" ]
```

- 本DevOps流水线需要预先制作用于发布Chart模板的 **values.yaml** 文件，并放置在代码仓库的nginx目录下。本实践方案中，根据规划信息values.yaml文件的内容如下。其中，**repository: hub.ecns.io/devops/nginx** 参数配置中，**devops** 为流水线“构建”阶段所选择的工作空间，**nginx** 为流水线“构建”阶段所构建镜像的名称。**tag: 6.0.1-alpha.${BUILD_ID}** 参数配置中，**6.0.1-alpha.${BUILD_ID}** 为流水线“构建”阶段所构建镜像的版本。**host: nginx.example.cn** 参数配置中，**nginx.example.cn** 为流水线“部署”阶段所规划的访问域名。

```
# Default values for nginx-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

image:
  repository: hub.ecns.io/devops/nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: 6.0.1-alpha.${BUILD_ID}

imagePullSecrets: []
```

```
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname
template
  name: ""

podAnnotations: {}

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

service:
  type: ClusterIP
  port: 80

ingress:
  enabled: true
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: nginx.example.cn
      paths:
        - path: /

  #  - secretName: chart-example-tls
```

```
  #    hosts:
  #      - chart-example.local

resources: {}
  # We usually recommend not to specify default resources and to leave this
as a conscious
  # choice for the user. This also increases chances charts run on
environments with little
  # resources, such as Minikube. If you do want to specify resources,
uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after
'resources:'.
  # limits:
  #   cpu: 100m
  #   memory: 128Mi
  # requests:
  #   cpu: 100m
  #   memory: 128Mi

autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
  # targetMemoryUtilizationPercentage: 80

nodeSelector: {}

tolerations: []

affinity: {}
```

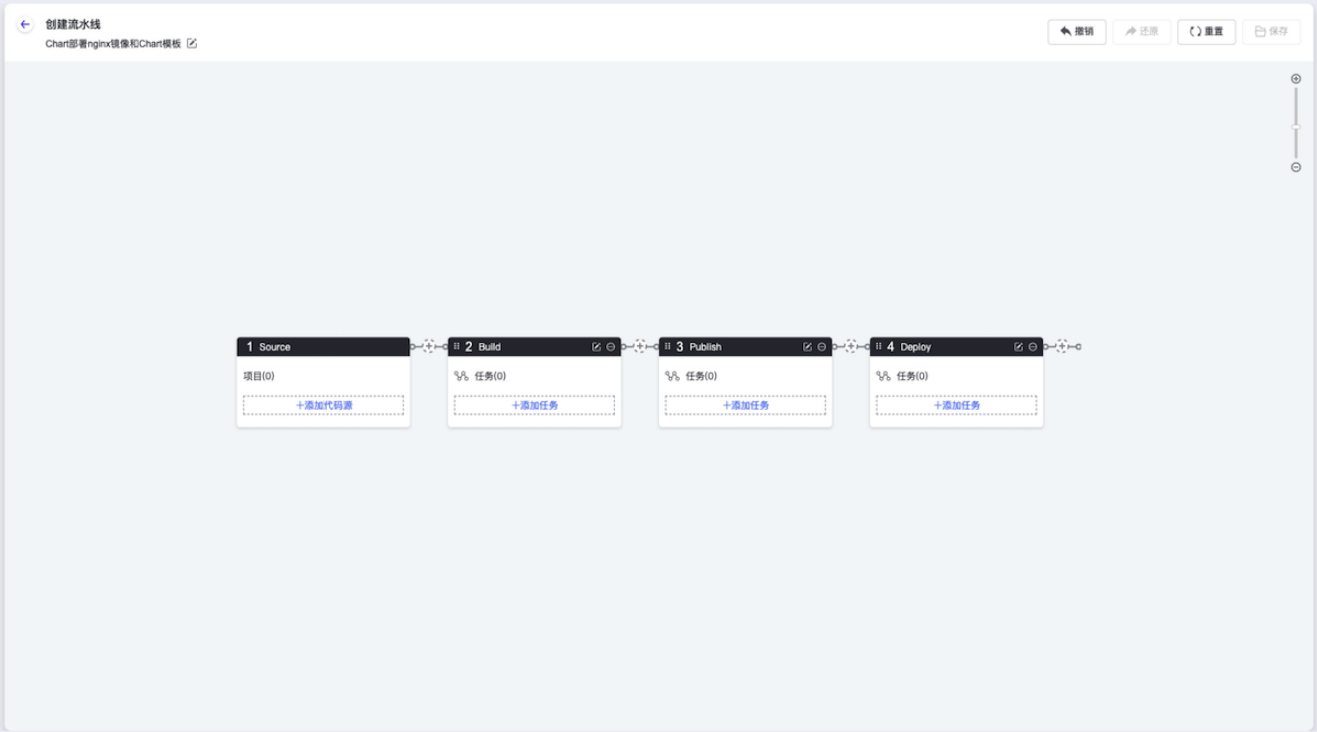
操作步骤

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。

2. 在“流水线”页面中，单击页面上方的 `创建流水线` ，弹出“创建流水线”对话框。

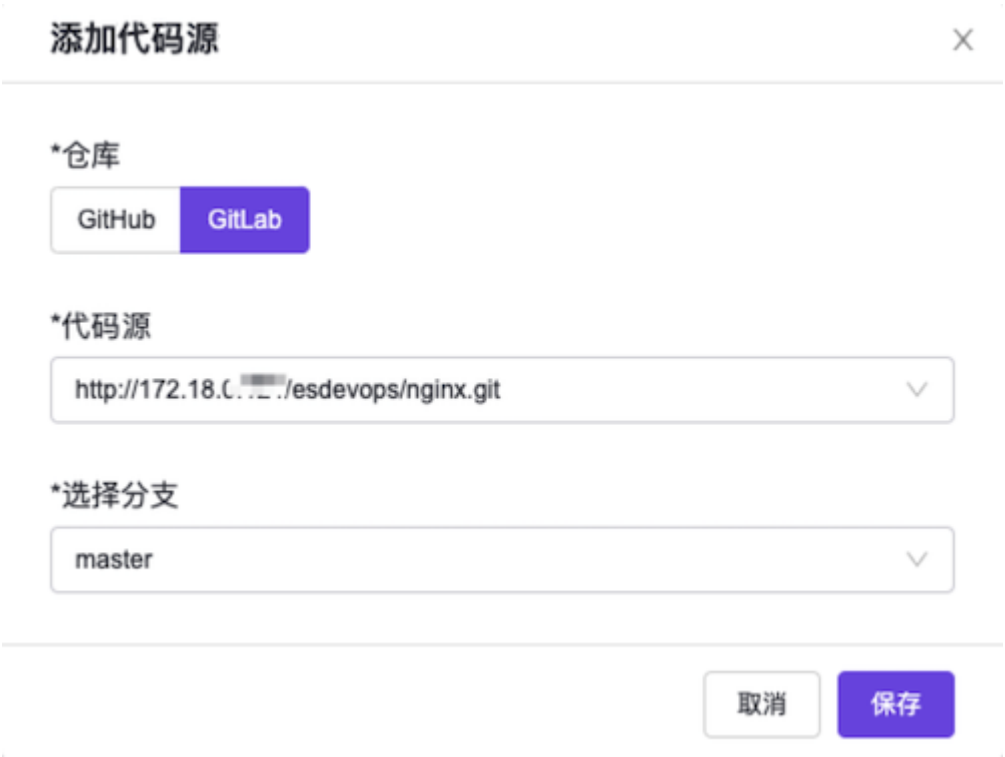
3. 在“创建流水线”对话框中，选择“从零开始创建”后，单击 `创建` ，进入“创建流水线”页面。

4. 在“创建流水线”页面的画布中，根据本次业务需求配置“Source”、“Build”、“Publish”和“Deploy”阶段后，依次在各阶段完成以下任务配置。



1. 在“Source”阶段添加项目代码源。

   在当前画布的“Source”卡片中，单击``添加代码源``，弹出“添加代码源”对话框。在该对话框中，配置代码源信息后，单击 `保存` ，保存项目的代码源设置，并关闭对话框。



2. 在“Build”阶段添加“构建并发布镜像”任务。

在当前画布的“Build”卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“构建并发布镜像”，“Dockerfile”输入“Dockerfile”，“镜像名称”输入“nginx”，“镜像版本”输入“6.0.1-alpha.${BUILD_ID}”，“工作空间”选择“devops”，并配置名称后，单击 保存 ，完成任务创建，并关闭对话框。

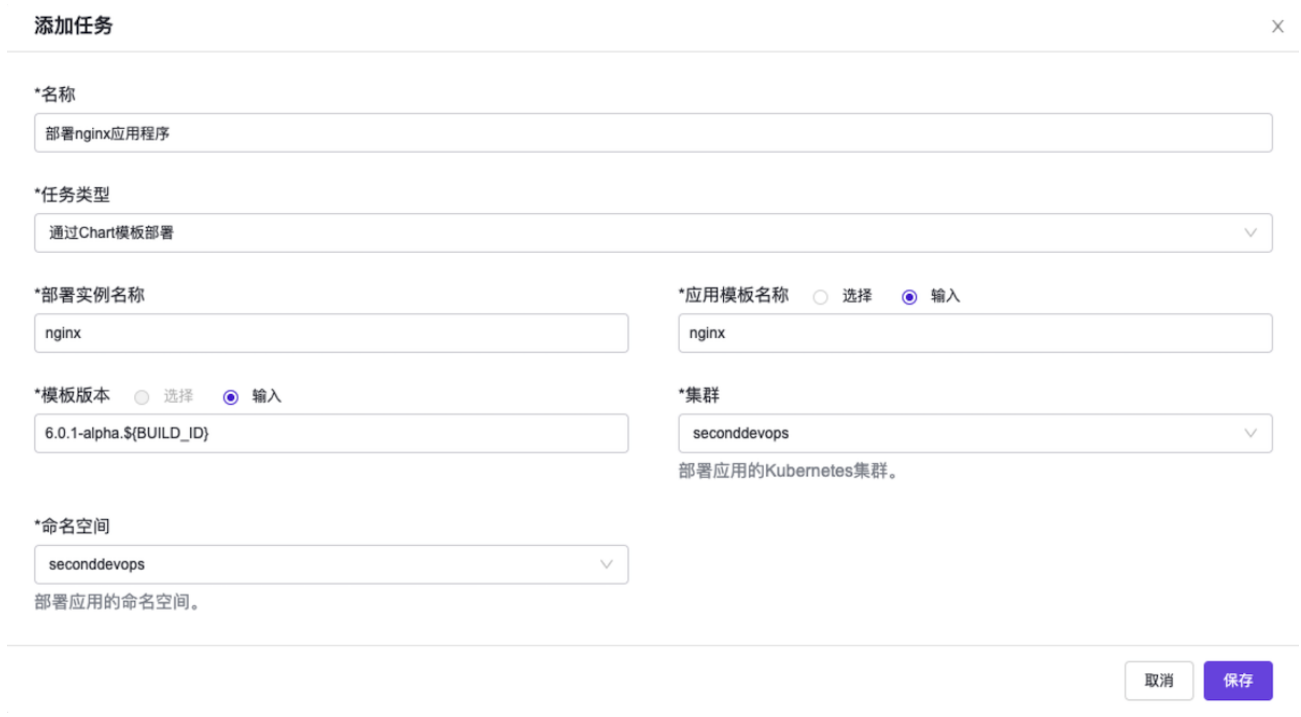
3. 在“Publish”阶段添加“构建并发布Chart模板”任务。

   在当前画布的“Publish”卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“构建并发布Chart模板”，“Chart目录”输入“nginx”，“应用模板名称”输入“nginx”，“模板版本”输入“6.0.1-alpha.${BUILD_ID}”，并配置名称后，单击 保存 ，完成任务创建，并关闭对话框。

4. 在“Deploy”阶段添加“通过Chart模板部署”任务。

   在当前画布的“Deploy”卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“通过Chart模板部署”，“部署实例名称”输入“nginx”，“应用模板名称”输入“nginx”，“模板版本”输入“6.0.1-alpha.${BUILD_ID}”，并配置名称和Chart模板的部署目标集群及命名空间后，单击 保存 ，完成任务创建，并关闭对话框。

   > 说明：
   > 在上述对话框中，“应用模板名称”和“模板版本”支持“选择”和“输入”两种配置方式，本实践方案需要选择“输入”配置方式。关于两种配置方式的具体说明如下，在实际使用过程中请酌情选择：
   > - 当待部署的应用模板为“容器应用中心”云产品中已有的自有模板时，请选择“选择”配置方式，依次选择本流水线任务所需的应用模板名称和版本。
   > - 当待部署的应用模板为本流水线前述阶段新构建发布的Chart模板时，请选择“输入”配置方式，依次输入前述构建发布阶段所配置的Chart模板的名称和版本。



5. 在“创建流水线”页面的画布中，单击画布右上方的 `保存` 后，在弹出的“保存”对话框中，选择保存方式后，单击 `保存` ，完成流水线创建，并关闭当前页面。

6. 执行流水线。

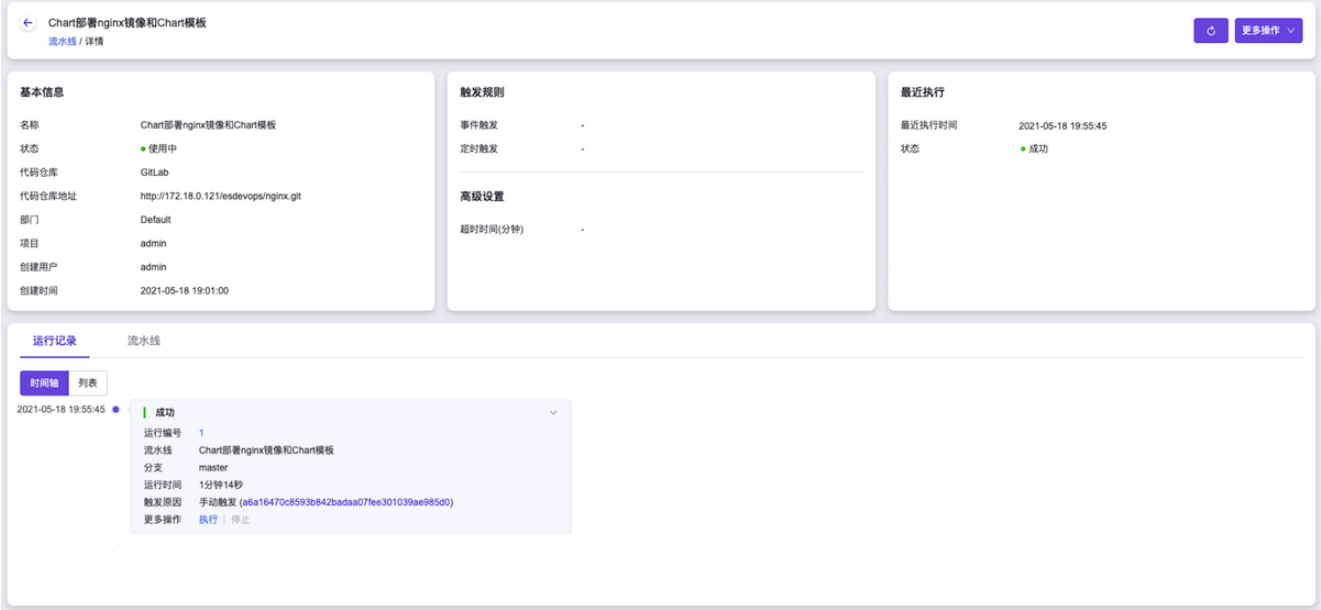
   本实践方案中以手动触发方式为例，触发流水线执行。如需配置流水线自动触发，请参考 配置流水线执行策略（可选）。

   1. 在“流水线”页面中，单击上述流水线所在行的 `执行` ，弹出“执行流水线”提示框。
   2. 在“执行流水线”提示框中，单击 `执行` ，执行该流水线，并关闭提示框。

结果验证

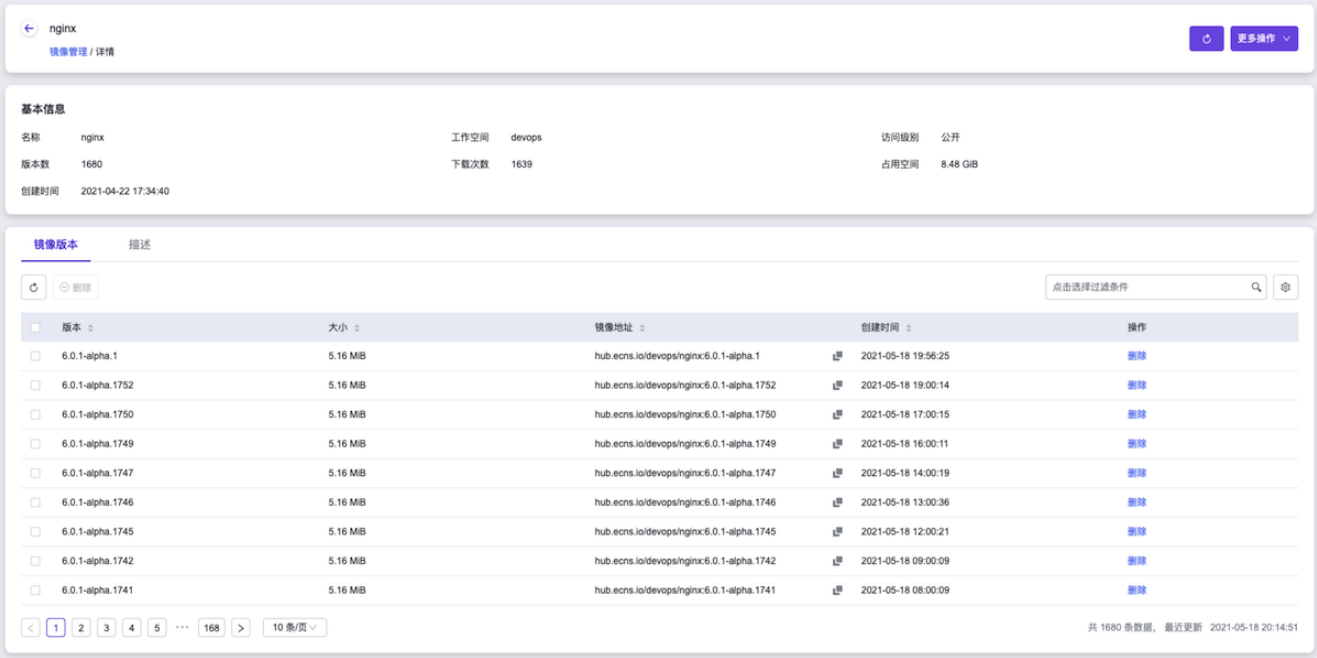
1. 确认流水线成功执行。

   在“流水线”页面中，单击上述流水线名称 ，进入其详情页面。在详情页面的[运行记录]页签中，确认该流水线执行成功并记录此次“运行编号”。



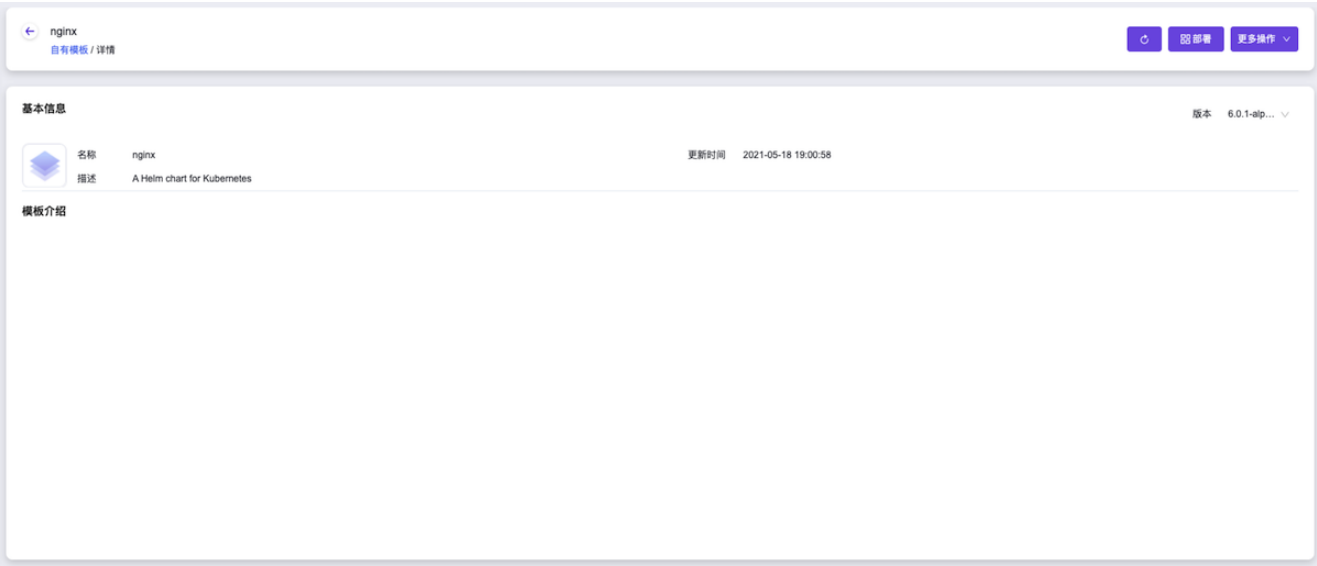
2. 确认镜像成功构建。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[容器镜像服务]，进入“镜像管理”页面。在该页面中，确认已生成名为“nginx”的镜像文件。然后，单击此镜像文件名称，进入其详情页面后，在[镜像版本]页签中，确认该镜像文件的生成版本为“6.0.1-alpha.<运行编号>”。



3. 确认Chart模板成功发布。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[容器应用中心]，进入“应用模板”页面。在该页面中，选择[自有模板]页签后，确认已生成名为“nginx”的模板文件。然后，单击此模板文件名称，进入其详情页面后，在“基本信息”区域框的右上方，确认最新生成的该模板文件的版本为“6.0.1-alpha.<运行编号>”。

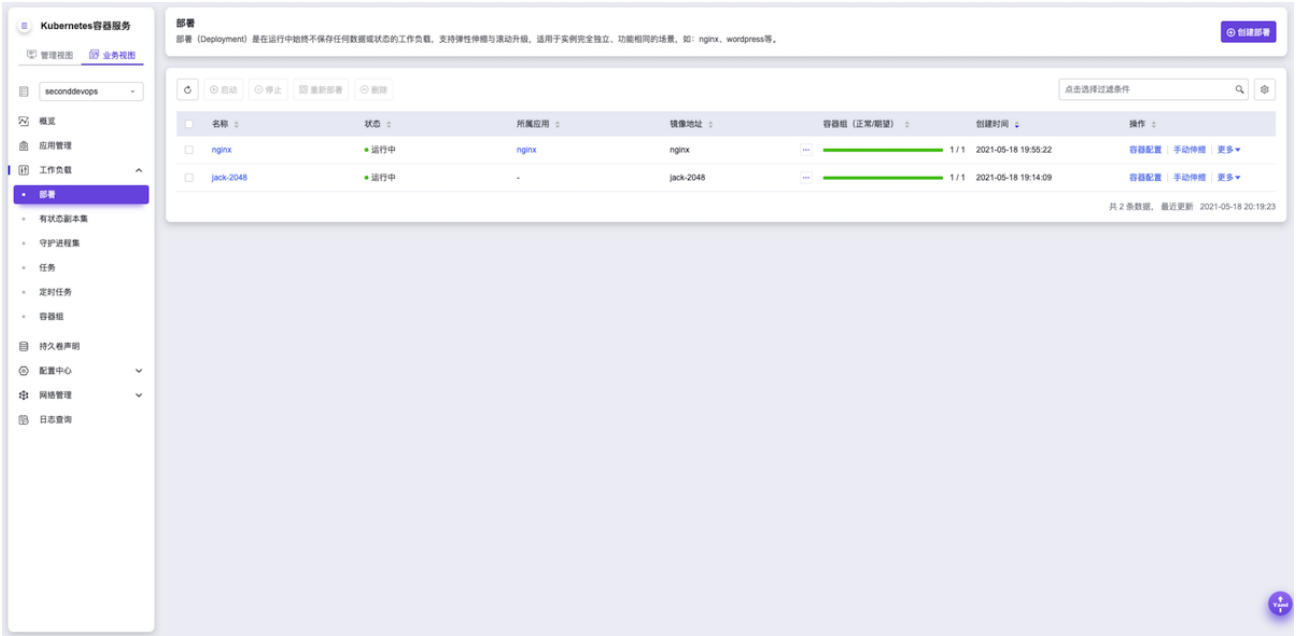


4. 确认应用程序成功部署。

   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[容器应用中心]，进入容器应用中心服务页面后，再在该页面的左侧导航栏中，选择“模板实例”，进入“模板实例”页面。在该页面中，确认已生成名

为“nginx”的模板实例，且“模板来源”为“nginx: 6.0.1-alpha.<运行编号>”。然后，单击此模板实例名称，进入其详情页面后，在[工作负载]页签中，确认所用镜像为“hub.ecns.io/devops/nginx:6.0.1-alpha.<运行编号>”。

2. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[Kubernetes容器服务]，进入Kubernetes容器服务页面后，再在该页面的左侧导航栏中，先选择“业务视图”以及该应用程序的所在项目、部署集群和命名空间，再选择[工作负载]-[部署]，进入部署页面。在该页面中，确认已生成名为“nginx”的部署且状态为“运行中”。



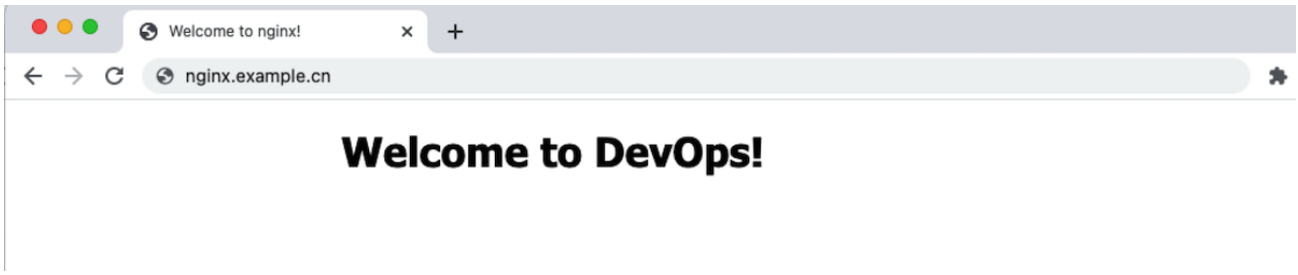
5. 确认应用程序成功访问。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[Kubernetes容器服务]，进入“管理视图”的“集群管理”页面。在该页面中，查看并记录该应用程序所在Kubernetes集群的IP地址，即该Kubernetes集群所在行中，API Server列https://后的IP地址。

2. 在本地计算机的 **hosts** 文件中，添加该应用程序所在Kubernetes集群IP地址与values.yaml文件定义域名的访问映射。

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
172.18.[redacted]     2048.example.cn
172.18.[redacted]     nginx.example.cn
```

3. 在本地计算机的浏览器地址栏中输入values.yaml文件定义域名，访问部署的应用程序。



5.3 通过DevOps扩展功能自动构建发布云产品

背景描述

DevOps支持通过将预置好的工具集镜像上传至容器镜像服务云产品后，与流水线的运行脚本类任务相互配合使用，达到扩展DevOps功能的目的。本公司秉承“eat your own dog food & 使用自己的产品设计开发云产品”的设计理念，所以，本公司云平台中所有云产品的发布均是通过DevOps扩展功能支撑实现的，其中不乏外部开发者开发的云产品，如恩耐博慧海数据平台和安全狗云眼平台等。本文将以此为例，介绍如何快速通过DevOps扩展功能，实现云产品的自动化构建并同步发布在OTA服务器中。

本实践方案中，流水线各项规划信息规划如下：

流水线-DevOps扩展功能自动构建发布云产品	
基本信息	* 云产品版本号：6.0.1-alpha.${BUILD_ID} * 本流水线各阶段的所有“运行脚本”类型任务中均使用此工具集镜像文件： * 镜像地址：名为 **docker** 的镜像文件，其作为自有镜像，上传在“容器镜像服务”云产品的工作空间中 * 镜像版本：dind-centos-final-v4
Source阶段	项目代码使用 **build-config.git** 仓库下的 **master** 分支代码
Initial阶段	通过“运行脚本”类型任务设置 **Config** 配置文件中的云产品名称 * 脚本：bash common/replace_config_content.sh * 环境变量：COMPONENTS_TO_BUILD = *云产品名称*

流水线-DevOps扩展功能自动构建发布云产品	
Build阶段	通过“登录Harbor”、“检查OTA服务器metadata版本”、“下载registry-cloud-product全量镜像”和“生成云产品manifest”四个任务获得云产品包构建所需的manifest * 登录Harbor：通过“运行脚本”类型任务登录Harbor仓库 * 脚本： docker login hub.example.cn -u ${dockerHubUser} -p $ {dockerHubPassword} docker login hub.example.io -u ${dockerHubUser} -p $ {dockerHubPassword} * 环境变量： dockerHubUser = *登录用户名* dockerHubPassword = *登录密码* * 检查OTA服务器metadata版本：通过“运行脚本”类型任务检查OTA服务器的metadata版本是否重复 * 脚本： bash common/check_cloud_product_metadata.sh * 下载云产品注册镜像：通过“运行脚本”类型任务下载registry-cloud-product全量镜像文件 * 脚本：（ *172.16.XX.XX* 为本公司File Server的IP地址） cd / && wget -q http://172.16.XX.XX:8000/build_cloud_product_file/x86_64/registry-cloud-product.tar tar -zxvf registry-cloud-product.tar > /dev/null * 生成云产品manifest：通过“运行脚本”类型任务生成云产品的manifest * 脚本： bash product/manifest_generator.sh
Deploy阶段	通过“运行脚本”类型任务构建云产品包并发布在OTA服务器中 * 脚本： bash product/ota_file_uploader.sh

前提条件

- DevOps流水线需要预先配置承载其运行的Kubernetes集群，具体步骤请参考 配置集群。
- DevOps流水线需要预先关联应用程序源代码的代码仓库，具体步骤请参考 配置代码仓库。

- 本DevOps流水线需要预先获取名为 **docker** 镜像文件存放在本地计算机中，用于在流水线执行过程中提供工具集镜像。

操作步骤

1. 上传镜像文件。

   本实践方案中以在云平台界面上传的方式为例，上传镜像文件。如需从Docker客户端和Containerd客户端中直接推送镜像，请参考“容器镜像服务”帮助中“上传镜像”的相关内容。

   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[容器镜像服务]，进入“镜像管理”页面。
   2. 在“镜像管理”页面中，选择[自有镜像]页签后，单击列表上方的 `上传镜像` ，弹出“上传镜像”对话框。

   3. 在“上传镜像”对话框中，选择工作空间和本地镜像文件后，单击 `上传` ，开始上传镜像文件，并关闭当前对话框。

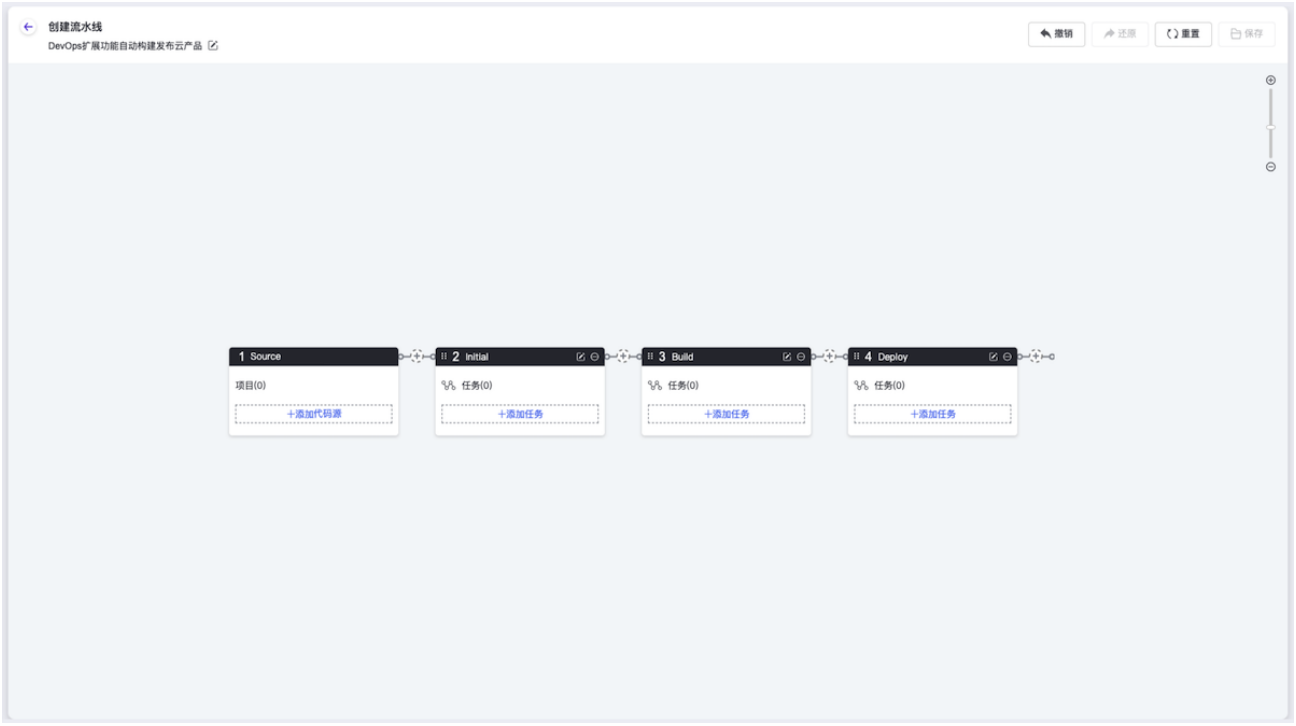
2. 创建流水线。

   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。

   2. 在“流水线”页面中，单击页面上方的 `创建流水线` ，弹出“创建流水线”对话框。

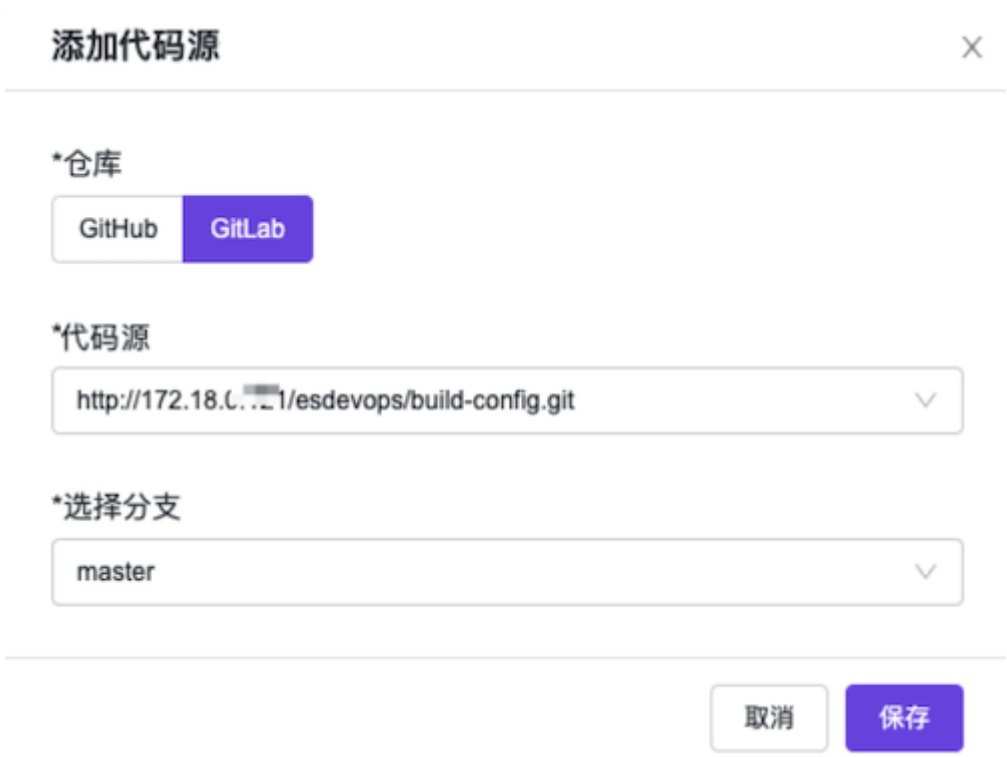
   3. 在“创建流水线”对话框中，选择“从零开始创建”后，单击 `创建` ，进入“创建流水线”页面。

4. 在“创建流水线”页面的画布中，根据本次业务需求配置“Source”、“Initial”、“Build”和“Deploy”阶段后，依次在各阶段完成对应任务配置。



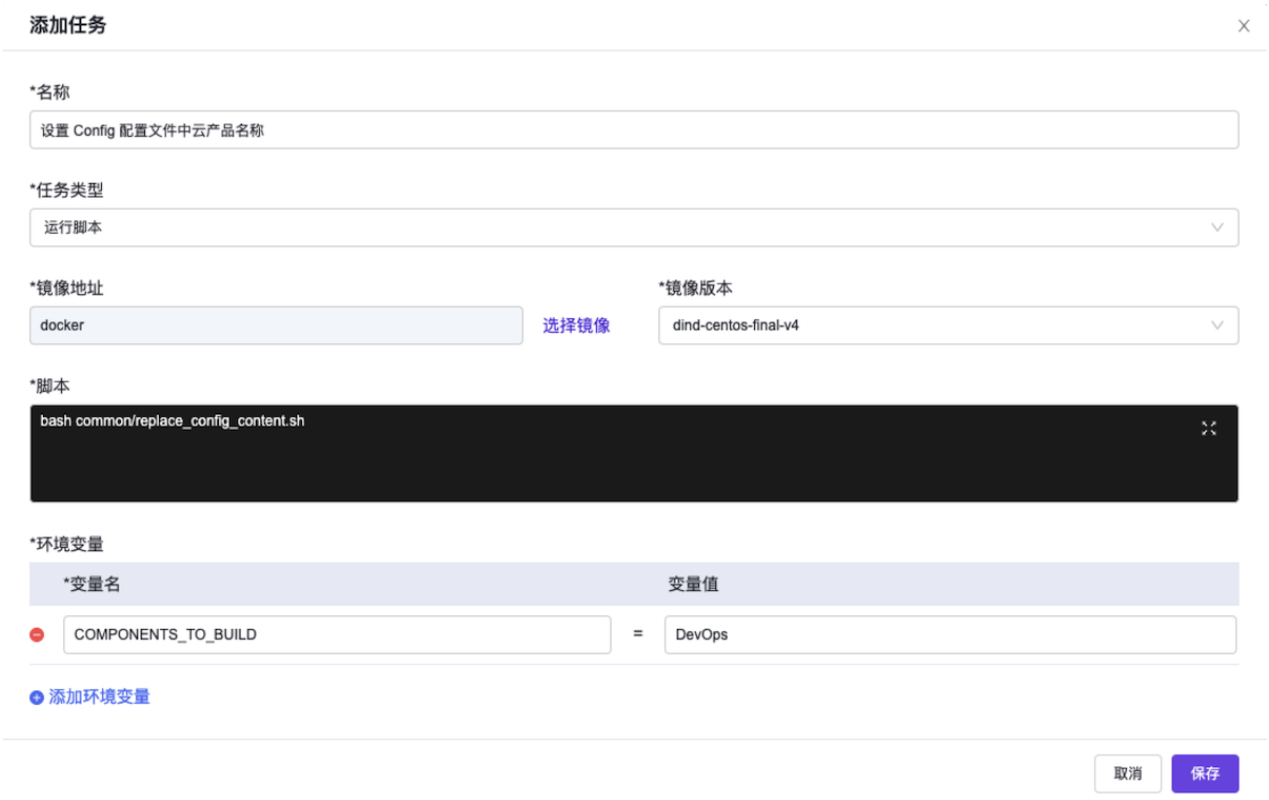
1. 在“Source”阶段添加项目代码源。

   在当前画布的“Source”卡片中，单击``添加代码源``，弹出“添加代码源”对话框。在该对话框中，配置代码源信息后，单击 `保存` ，保存项目的代码源设置，并关闭对话框。



2. 分别在“Initial”、“Build”和“Deploy”阶段添加对应的“运行脚本”任务。

   在当前画布的对应阶段卡片中，单击 添加任务 ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“运行脚本”，“镜像地址”选择[自有镜像]页签中的“docker”，“镜像版本”选择“dind-centos-final-v4”，并配置名称及对应方案规划信息中的脚本内容和环境变量后，单击 保存 ，完成任务创建，并关闭对话框。



5. 在“创建流水线”页面的画布中，单击画布右上方的 保存 后，在弹出的“保存”对话框中，选择保存方式后，单击 保存 ，完成流水线创建，并关闭当前页面。

3. 执行流水线。

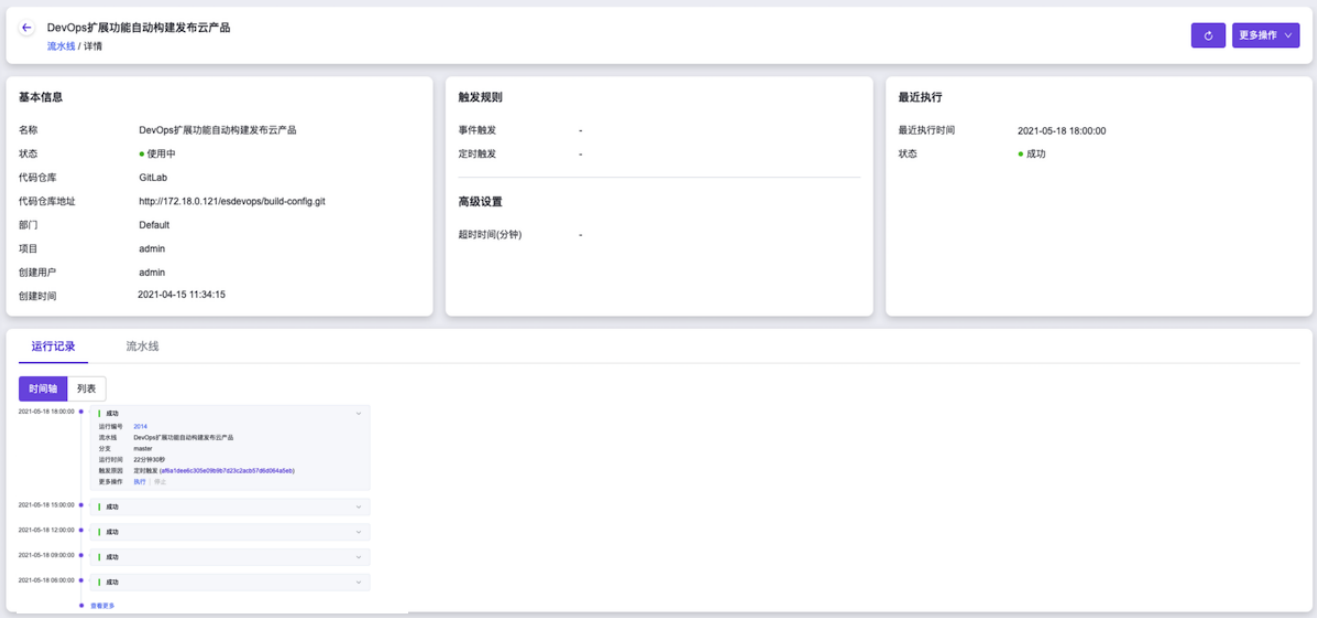
本实践方案中以手动触发方式为例，触发流水线执行。如需配置流水线自动触发，请参考 配置流水线执行策略（可选）。

1. 在“流水线”页面中，单击上述流水线所在行的 `执行` ，弹出“执行流水线”提示框。
2. 在“执行流水线”提示框中，单击 `执行` ，执行该流水线，并关闭提示框。

结果验证

1. 确认流水线成功执行。

   在“流水线”页面中，单击上述流水线名称 ，进入其详情页面。在详情页面的[运行记录]页签中，确认该流水线执行成功并记录此次“运行编号”。



2. 确认云产品成功发布。

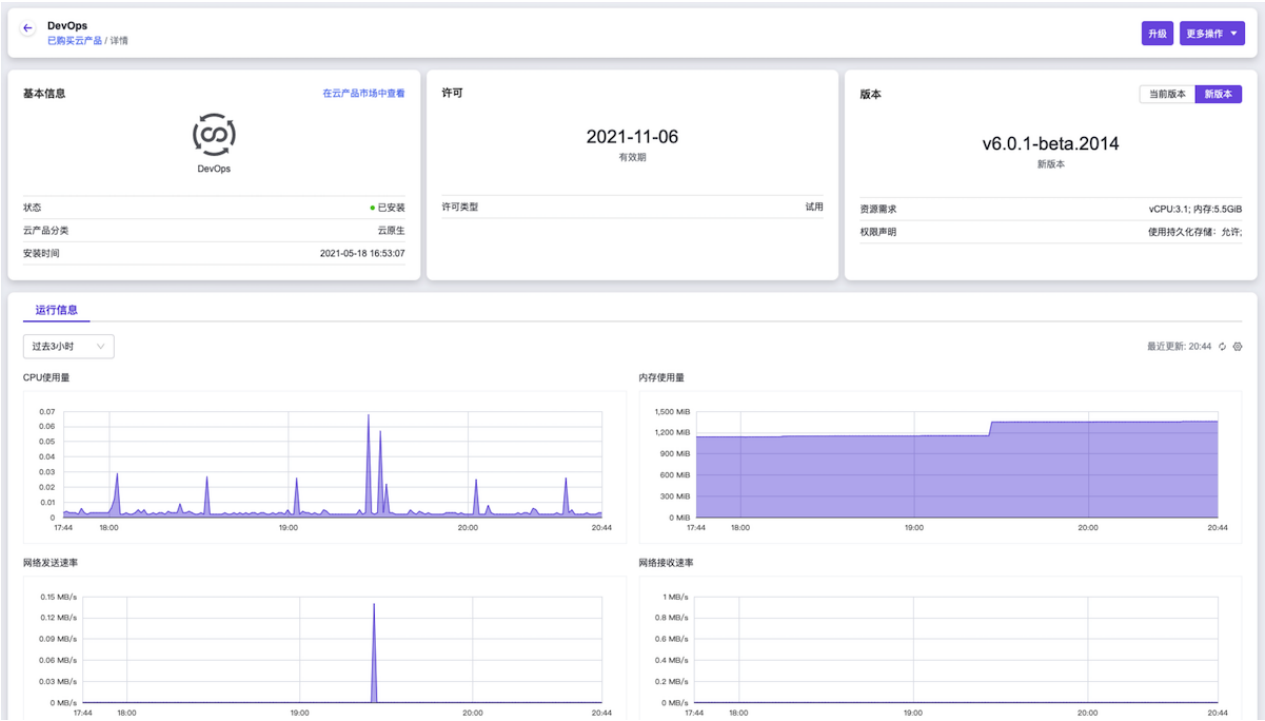
   当该云产品在云平台已安装时，请直接执行下述操作。当该云产品还未在云平台安装时，请先在云产品市场中获取该云产品，再执行下述操作。

   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[产品与服务管理]-[已购买云产品]，进入“已购买云产品”页面。

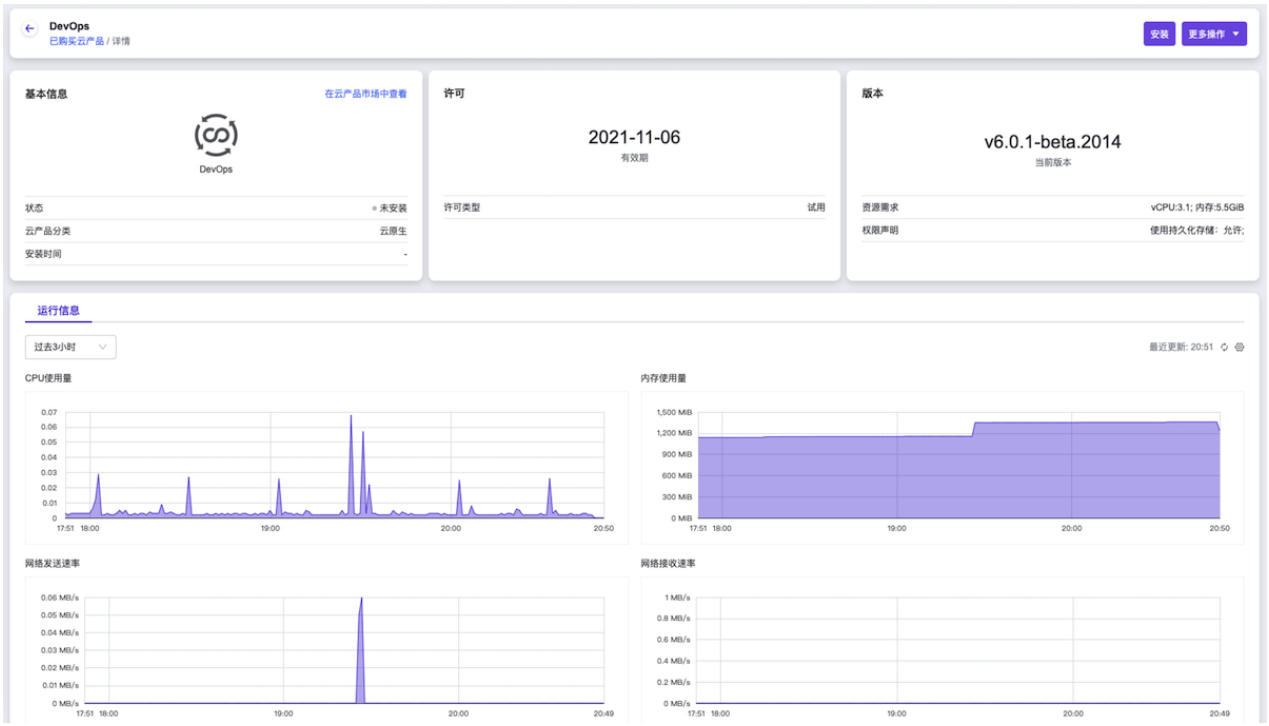
   2. 在“已购买云产品”页面中，根据该云产品安装状态，酌情选择下述对应操作确认该云产品成功发布。

- 当该云产品为已安装状态时，请首先在该页面中确认，该云产品所在行的“已安装版本”后有“可升级”提示。

然后，在该页面中单击该云产品名称，进入其详情页面。在详情页面的“版本”区域框中，选择[新版本]页签，确认该云产品新版本为“6.0.1-alpha.<运行编号>”。



- 当该云产品为未安装状态时，单击该云产品名称，进入其详情页面。在该页面的“版本”区域框中，确认该云产品版本为“6.0.1-alpha.<运行编号>”。


DevOps
已购买云产品 / 详情
安装
更多操作
基本信息
在云产品市场中查看
DevOps
状态
未安装
云产品分类
云原生
安装时间
-
许可
2021-11-06
有效期
许可类型
试用
版本
v6.0.1-beta.2014
当前版本
资源需求
vCPU:3.1; 内存:5.5GiB
权限声明
使用持久化存储：允许;
运行信息
过去3小时
最近更新: 20:51
CPU使用量
内存使用量
网络发送速率
网络接收速率

5.4 通过DevOps扩展功能部署虚拟化云平台

背景描述

DevOps支持通过将预置好的工具集镜像上传至容器镜像服务云产品后，与流水线的运行脚本类任务相互配合使用，达到扩展DevOps功能的目的。本公司秉承“eat your own dog food & 使用自己的产品设计开发云产品”的设计理念，所以，在日常工作中本公司通过利用云基础设施的资源编排能力和DevOps扩展功能，在一个云平台中编排部署多个虚拟化云平台，用于为工程技术团队提供每天最新的研发或测试环境。本文将以此为例，介绍如何快速通过DevOps扩展功能部署虚拟化云平台。

本实践方案中，流水线各项规划信息规划如下：

流水线-DevOps扩展功能部署虚拟化云平台	
Source阶段	项目代码使用 **os-in-os.git** 仓库下的 **master** 分支代码
Deploy阶段	通过“运行脚本”类型任务部署虚拟化云平台 * 镜像地址：名为 **escloud-linux-source-busybox** 的镜像文件，其作为自有镜像，上传在“容器镜像服务”云产品的工作空间中 * 镜像版本：6.0.1

前提条件

- DevOps流水线需要预先配置承载其运行的Kubernetes集群，具体步骤请参考 配置集群。
- DevOps流水线需要预先关联应用程序源代码的代码仓库，具体步骤请参考 配置代码仓库。
- 本DevOps流水线需要预先获取名为 **escloud-linux-source-busybox** 镜像文件存放在本地计算机中，用于在流水线执行过程中提供工具集镜像。
- 本DevOps流水线需要预先获取名为 **stack-3nodes-3cloud-product-nodes.yaml** 的YAML文件放置在代码仓库的根目录下，用于在流水线执行过程中进行资源编排部署。
- 本DevOps流水线需要预先制作用于检查虚拟化云平台部署状态的 **check_stack_status.sh** 脚本文件，并放置在代码仓库的根目录下。本实践方案中，根据规划信息check_stack_status.sh脚本文件的内容如下：

```
#!/usr/bin/env bash

source ./openrc-sz

if heat stack-list | grep CREATE_IN_PROGRESS | grep $1 ; then
    echo 'Heat stack is creating'
    while true; do
        if heat stack-list | grep CREATE_COMPLETE | grep $1; then
            echo "stack create complete"
            exit
        fi
        if heat stack-list | grep CREATE_FAILED | grep $1; then
            echo "stack create failed"
            exit 1
        fi
        sleep 300
    done
else
    echo 'No stack is creating or stack create failed, so exit with 1'
    exit 1
fi
```

操作步骤

1. 上传镜像文件。

   本实践方案中以在云平台界面上传的方式为例，上传镜像文件。如需从Docker客户端和Containerd客户端中直接推送镜像，请参考“容器镜像服务”帮助中“上传镜像”的相关内容。

   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[容器镜像服务]，进入“镜像管理”页面。
   2. 在“镜像管理”页面中，选择“自有镜像”页签后，单击列表上方的 `上传镜像` ，弹出“上传镜像”对话框。

3. 在“上传镜像”对话框中，选择工作空间和本地预先获取的镜像文件后，单击 上传 ，开始上传镜像文件，并关闭当前对话框。

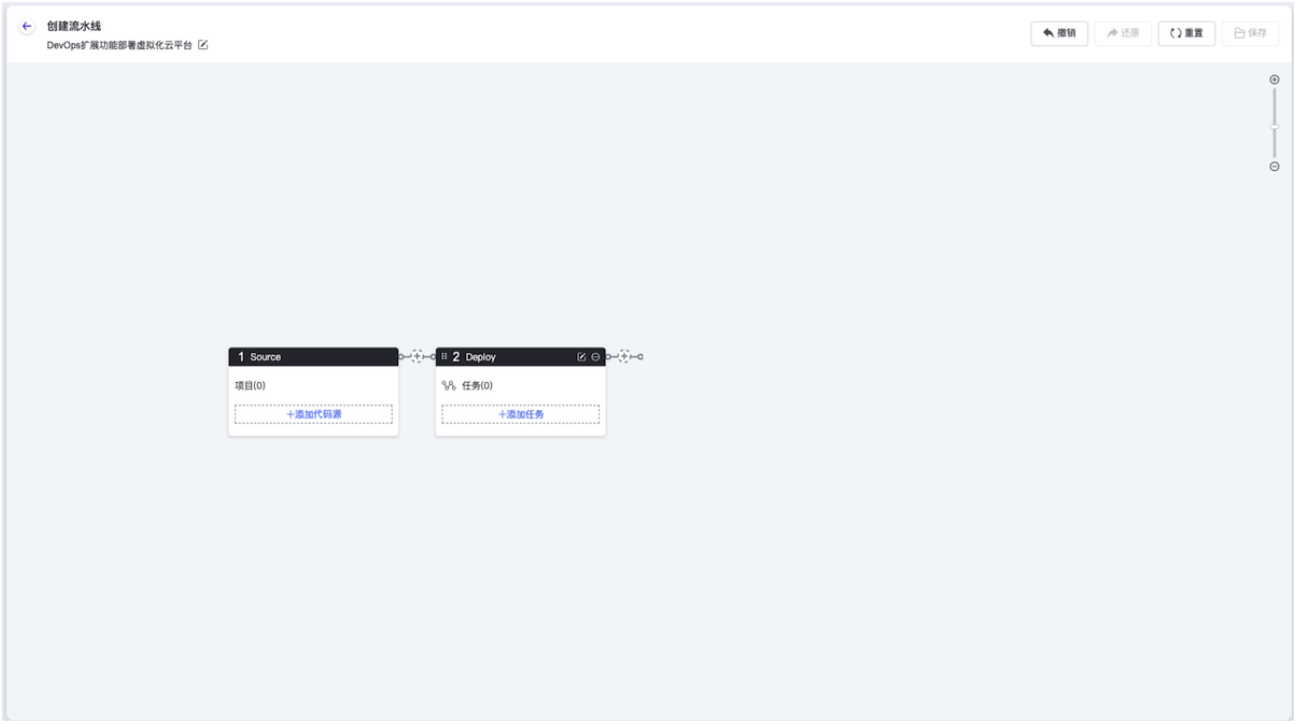
2. 创建流水线。

1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[流水线]，进入“流水线”页面。

2. 在“流水线”页面中，单击页面上方的 `创建流水线` ，弹出“创建流水线”对话框。

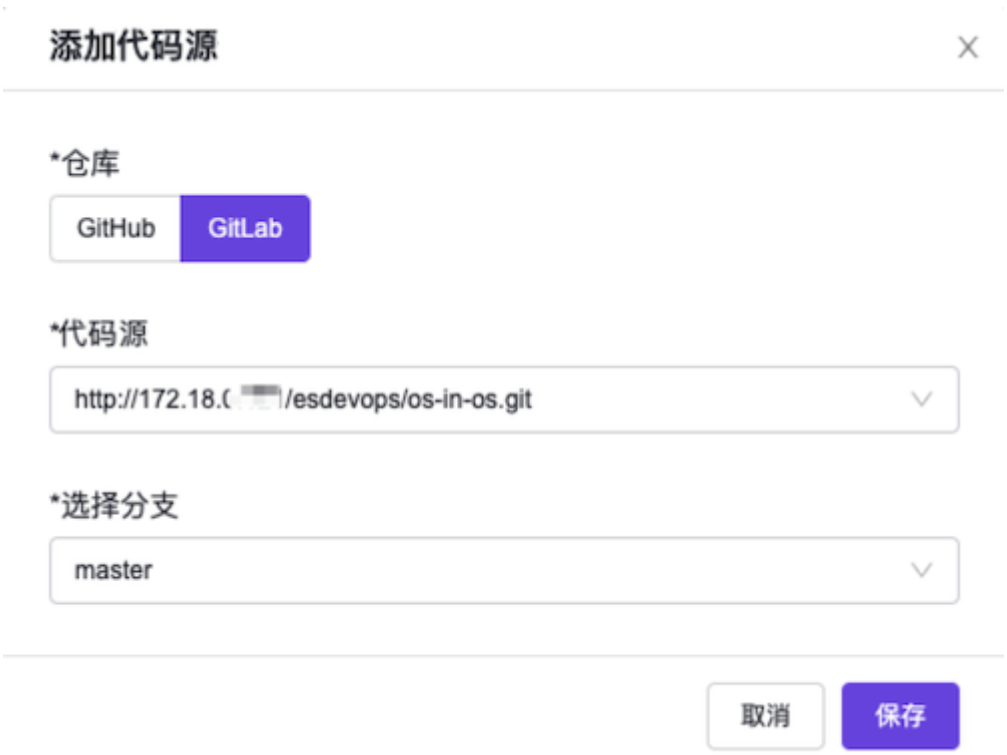
3. 在“创建流水线”对话框中，选择“从零开始创建”后，单击 `创建` ，进入“创建流水线”页面。

4. 在“创建流水线”页面的画布中，根据本次业务需求配置“Source”和“Deploy”阶段后，依次在各阶段完成以下任务配置。



1. 在“Source”阶段添加项目代码源。

   在当前画布的“Source”卡片中，单击``添加代码源``，弹出“添加代码源”对话框。在该对话框中，配置代码源信息后，单击 `保存` ，保存项目的代码源设置，并关闭对话框。



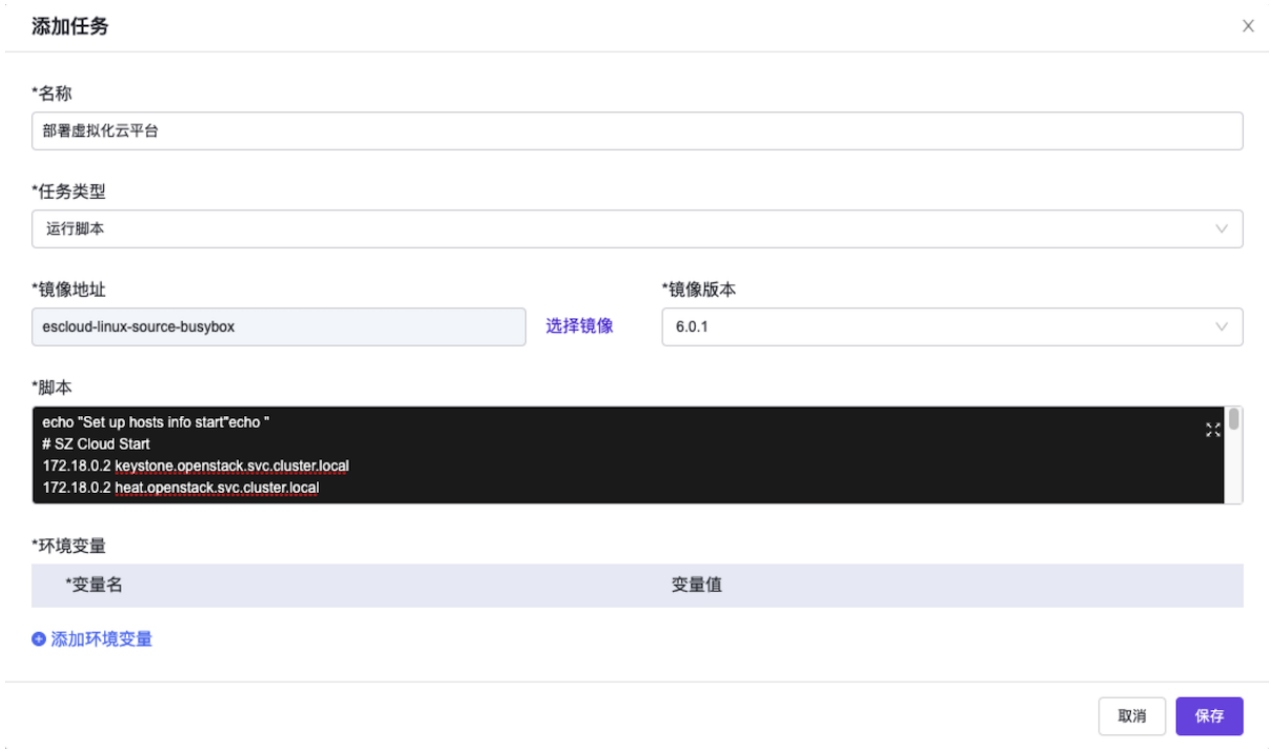
2. 在“Deploy”阶段添加“运行脚本”任务。

   在当前画布的“Deploy”卡片中，单击 `添加任务` ，弹出“添加任务”对话框。在该对话框中，“任务类型”选择“运行脚本”，“镜像地址”选择[自有镜像]页签中的“escloud-linux-source-busybox”，“镜像版本”选择“6.0.1”，并配置名称和脚本后，单击 `保存` ，完成任务创建，并关闭对话框。

   在上述“添加任务”对话框中，“脚本”输入内容如下。其中， *172.18.XX.XX* 为用于部署虚拟化云平台的云平台外部访问IP地址:

```
echo "Set up hosts info start"echo "
# SZ Cloud Start
172.18.XX.XX keystone.openstack.svc.cluster.local
172.18.XX.XX heat.openstack.svc.cluster.local
172.18.XX.XX neutron.openstack.svc.cluster.local
172.18.XX.XX nova.openstack.svc.cluster.local
172.18.XX.XX glance.openstack.svc.cluster.local
172.18.XX.XX cinder.openstack.svc.cluster.local
# SZ Cloud End
" >> /etc/hosts
echo "Set up hosts info done"

source ./openrc-sz
#get the correct version,through pkg server
VERSION=6.0.1
python update_version.py $VERSION
STACK_NAME=devops-deploy-$RANDOM-os-in-os
echo 'Try to create stack 3 nodes 3 cloud product nodes os in os env'
 heat stack-create $STACK_NAME -f ./stack-3nodes-3cloud-product-
nodes.yaml
# loop to check stack status
bash ./check_stack_status.sh  $STACK_NAME
```



5. 在“创建流水线”页面的画布中，单击画布右上方的 `保存` 后，在弹出的“保存”对话框中，选择保存方式后，单击 `保存` ，完成流水线创建，并关闭当前页面。

3. 执行流水线。

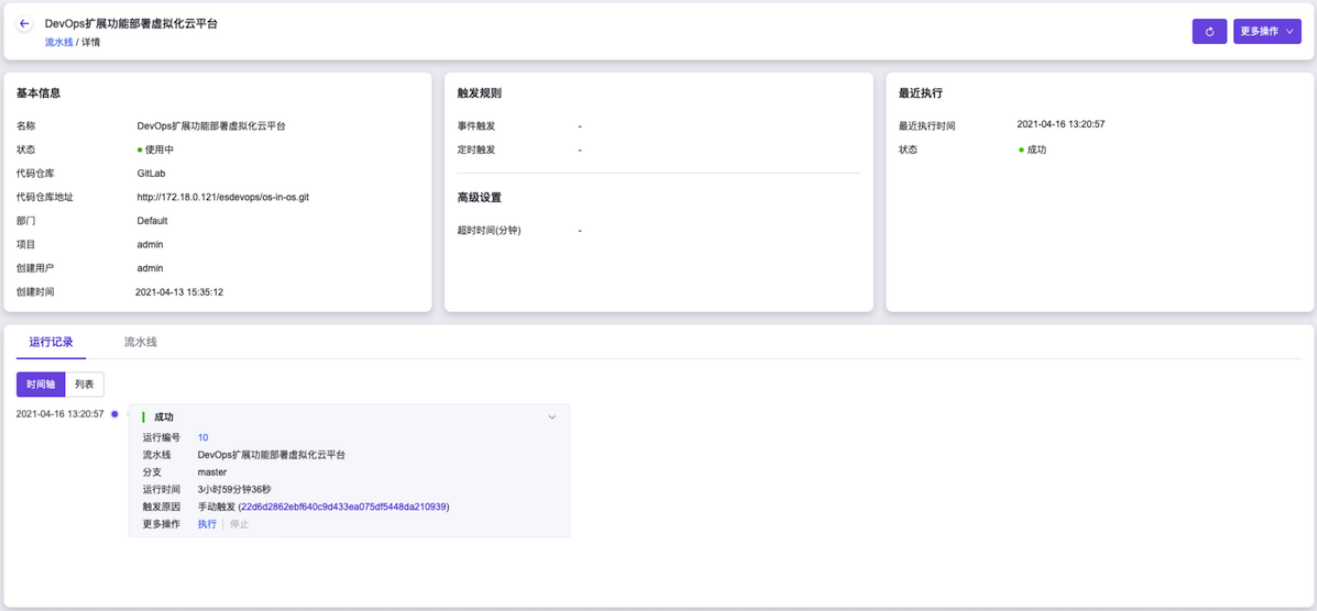
本实践方案中以手动触发方式为例，触发流水线执行。如需配置流水线自动触发，请参考 配置流水线执行策略（可选）。

1. 在“流水线”页面中，单击上述流水线所在行的 `执行` ，弹出“执行流水线”提示框。
2. 在“执行流水线”提示框中，单击 `执行` ，执行该流水线，并关闭提示框。

结果验证

1. 确认流水线成功执行。

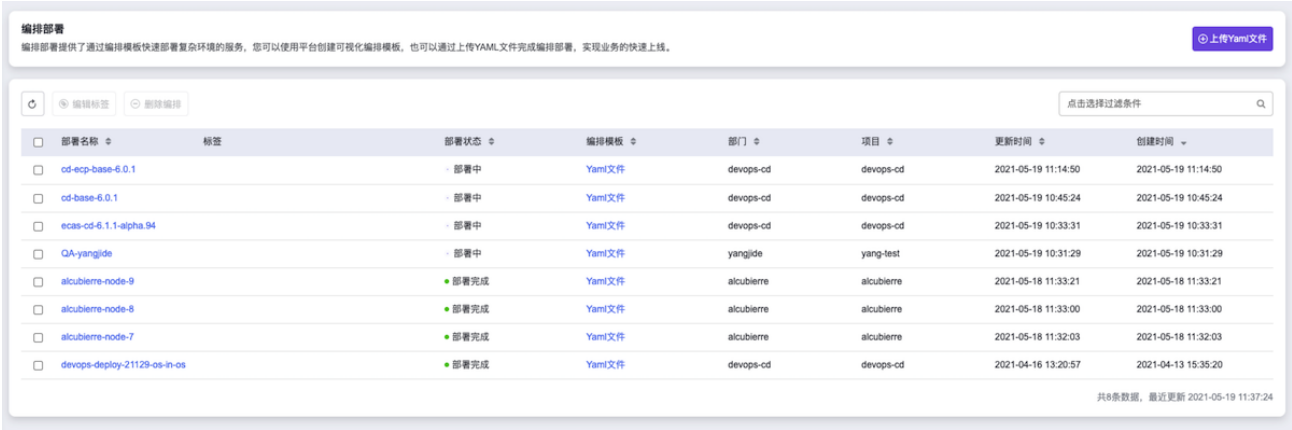
   在“流水线”页面中，单击上述流水线名称 ，进入其详情页面。在详情页面的[运行记录]页签中，确认该流水线执行成功。



2. 确认云平台成功编排部署。

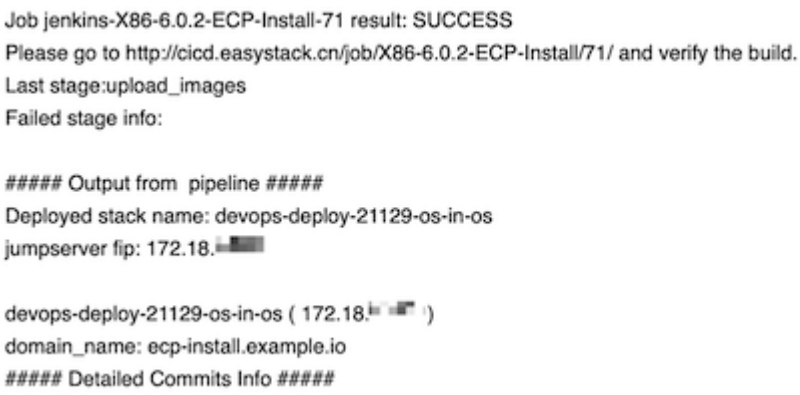
   1. 在云平台的顶部导航栏中，依次选择[产品与服务]-[资源编排]-[编排部署]，进入“编排部署”页面。

   2. 在“编排部署”页面中，确认已生成名如 *devops-deploy-$RANDOM-os-in-os* 的部署，且部署状态为“部署完成”。



3. 确认云平台成功访问。

1. 在流水线执行完成后，会将执行结果以邮件形式发送至工程技术团队所有成员的个人邮箱中。在此邮件中，**Deployed stack name** 参数的值即为上述部署名称 *devops-deploy-$RANDOM-os-in-os* ，**jumpserver fip** 参数的值即为虚拟化云平台的外网访问IP地址。



2. 在本地计算机的浏览器地址栏中输入虚拟化云平台的外网访问IP地址，访问部署的虚拟化云平台。

6 常见问题

6.1 当执行流水线时提示Avoid second fetch，如何排查解决?

问题描述

当执行流水线时，执行状态显示失败。之后，在查看此次运行记录的详情时，发现在Source阶段提示Avoid second fetch报错。

```
[2021-03-29 17:31:54] The recommended git tool is: NONE
[2021-03-29 17:32:05] using credential a25680ea-c325-43c5-907f-2407c99ed6f3
[2021-03-29 17:32:05] Cloning the remote Git repository
[2021-03-29 17:32:04] Cloning repository http://172.16.0.24/user1/jack-2048.git
[2021-03-29 17:32:04] > git init /home/jenkins/agent/workspace/ff7214c9-710c-4e93-a2c9-2a03a59df822 # timeout=10
[2021-03-29 17:32:06] Avoid second fetch
```

问题原因

当前流水线的源代码分支可能由于已被移除等原因导致无法正常获取。

解决方案

1. 在云平台中查看上述流水线的源代码分支。

   在“流水线”页面中，单击上述流水线名称 ，可进入其详情页面。在详情页面中，选择[流水线]页签，查看该流水线的源代码触发分支。

2. 在相应代码仓库中确认该分支是否可以正常获取。

   - 若无法正常获取，请编辑该流水线，酌情选择可正常获取的分支。编辑流水线的具体操作，请参考 流水线。
   - 若是该源代码分支可以正常获取但仍有此报错，请立即联系相关技术支持人员获取帮助。

6.2 当流水线始终为执行中状态且提示资源不足时，如何排查解决?

问题描述

当执行流水线时，执行状态长时间显示为执行中，并且提示相关资源不足的信息（如：0/5 nodes are available: 1 node(s) had taints that the pod didn’t tolerate, 4 Insufficient memory）。

```
[2021-04-12 11:30:03] Created Pod: devops/7531afbc-6229-4eab-a9c9-dc4597aec0be-1326-7m9rr-chr2n-85xjx
[2021-04-12 11:30:03] [Warning][devops/7531afbc-6229-4eab-a9c9-dc4597aec0be-1326-7m9rr-chr2n-85xjx][FailedScheduling] 0/5 nodes are available: 1 node(s) had taints that the pod didn't tolerate, 4 Insufficient memory.
[2021-04-12 11:30:03] [Warning][devops/7531afbc-6229-4eab-a9c9-dc4597aec0be-1326-7m9rr-chr2n-85xjx][FailedScheduling] 0/5 nodes are available: 1 node(s) had taints that the pod didn't tolerate, 4 Insufficient memory.
```

问题原因

当前承载流水线运行的Kubernetes集群的资源不足。

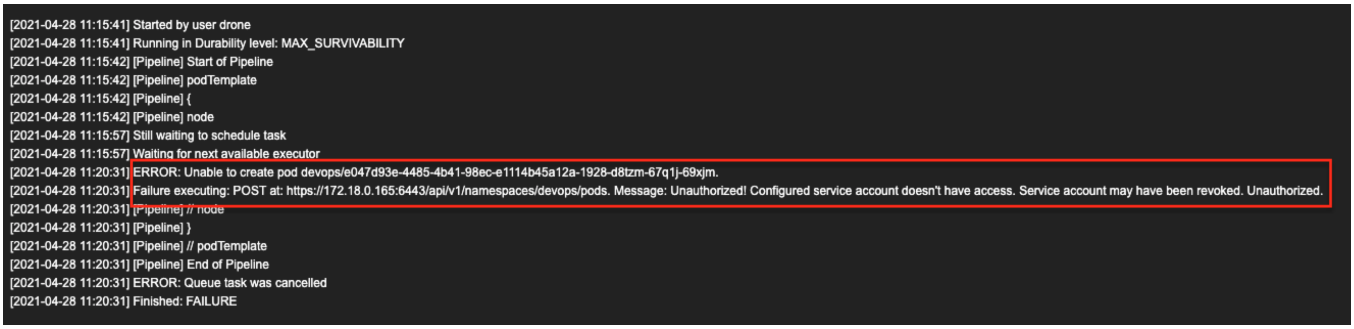
解决方案

请扩容该Kubernetes集群。扩容Kubernetes集群的具体操作步骤，请参考“Kubernetes容器服务”帮助中“扩容集群”的相关内容。

6.3 当执行流水线时提示Unable to create pod XX，如何排查解决?

问题描述

当执行流水线时，执行状态显示失败。之后，在查看此次运行记录的详情时，发现提示Unable to create pod XX报错。



此时，在集群配置页面中，可同时看到集群名称后有异常提示，提示信息为“业务集群已经被删除，请配置新的业务集群”或“集群不健康或者不能连接，请到kubernetes容器服务中检查集群状态”。

问题原因

当前承载流水线运行的Kubernetes集群状态异常，无法正常连接。

解决方案

在云平台的顶部导航栏中，依次选择[产品与服务]-[容器服务]-[Kubernetes容器服务]，进入Kubernetes容器服务后，检查上述Kubernetes集群的状态并进行修复，直至恢复为“健康”状态。

当上述Kubernetes集群为“健康”状态时，DevOps功能会自动恢复正常，可以直接按需重新执行上述流水线。

6.4 当执行流水线时提示 java.lang.NullPointerException，如何排查解决?

问题描述

当执行流水线时，执行状态显示失败。之后，在查看此次运行记录的详情时，发现提示 **java.lang.NullPointerException** 报错。

```
[2021-05-08T07:00:40.996Z] java.lang.NullPointerException
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.KubernetesSlave.getKubernetesCloud(K
ubernetesSlave.java:233)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.KubernetesSlave.getKubernetesCloud(K
ubernetesSlave.java:225)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.KubernetesNodeContext.conne
ctToCloud(KubernetesNodeContext.java:59)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.ContainerExecDecorator.getC
lient(ContainerExecDecorator.java:150)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.ContainerExecDecorator$1.do
Launch(ContainerExecDecorator.java:371)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.ContainerExecDecorator$1.la
unch(ContainerExecDecorator.java:336)
[2021-05-08T07:00:40.996Z]  at
hudson.Launcher$ProcStarter.start(Launcher.java:454)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.durabletask.BourneShellScript.launchWithCookie(BourneS
hellScript.java:234)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.durabletask.FileMonitoringTask.launch(FileMonitoringTa
sk.java:103)
[2021-05-08T07:00:40.996Z]  at
```

```
org.jenkinsci.plugins.workflow.steps.durable_task.DurableTaskStep$Execution.
start(DurableTaskStep.java:317)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.DSL.invokeStep(DSL.java:286)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.DSL.invokeMethod(DSL.java:179)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsScript.invokeMethod(CpsScript.java:122
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1138)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:810)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.GroovyObjectSupport.invokeMethod(GroovyObjectSupport.java:46)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1172)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:810)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.GroovyObjectSupport.invokeMethod(GroovyObjectSupport.java:46)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1172)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:810)
[2021-05-08T07:00:40.996Z]  at
```

```
groovy.lang.GroovyObjectSupport.invokeMethod(GroovyObjectSupport.java:46)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1172)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.call(PogoMetaClassSit
e.java:42)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray
.java:48)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
java:113)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.sandbox.DefaultInvoker.methodCall(DefaultInvoker.ja
va:20)
[2021-05-08T07:00:40.996Z] Caused: java.lang.RuntimeException
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.ContainerExecDecorator.getC
lient(ContainerExecDecorator.java:152)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.ContainerExecDecorator$1.do
Launch(ContainerExecDecorator.java:371)
[2021-05-08T07:00:40.996Z]  at
org.csanchez.jenkins.plugins.kubernetes.pipeline.ContainerExecDecorator$1.la
unch(ContainerExecDecorator.java:336)
[2021-05-08T07:00:40.996Z]  at
hudson.Launcher$ProcStarter.start(Launcher.java:454)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.durabletask.BourneShellScript.launchWithCookie(BourneS
hellScript.java:234)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.durabletask.FileMonitoringTask.launch(FileMonitoringTa
sk.java:103)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.steps.durable_task.DurableTaskStep$Execution.
start(DurableTaskStep.java:317)
```

```
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.DSL.invokeStep(DSL.java:286)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.DSL.invokeMethod(DSL.java:179)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsScript.invokeMethod(CpsScript.java:122
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1138)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:810)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.GroovyObjectSupport.invokeMethod(GroovyObjectSupport.java:46)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1172)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:810)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.GroovyObjectSupport.invokeMethod(GroovyObjectSupport.java:46)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1172)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:810)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.GroovyObjectSupport.invokeMethod(GroovyObjectSupport.java:46)
[2021-05-08T07:00:40.996Z]  at
```

```
groovy.lang.MetaClassImpl.invokeMethodOnGroovyObject(MetaClassImpl.java:1278
)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1172)
[2021-05-08T07:00:40.996Z]  at
groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1022)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.call(PogoMetaClassSit
e.java:42)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray
.java:48)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
java:113)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.sandbox.DefaultInvoker.methodCall(DefaultInvoker.ja
va:20)
[2021-05-08T07:00:40.996Z]  at WorkflowScript.run(WorkflowScript:63)
[2021-05-08T07:00:40.996Z]  at ___cps.transform___(Native Method)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.ContinuationGroup.methodCall(ContinuationGroup
.java:86)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.FunctionCallBlock$ContinuationImpl.dispatchOrA
rg(FunctionCallBlock.java:113)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.FunctionCallBlock$ContinuationImpl.fixArg(Func
tionCallBlock.java:83)
[2021-05-08T07:00:40.996Z]  at
sun.reflect.GeneratedMethodAccessor240.invoke(Unknown Source)
[2021-05-08T07:00:40.996Z]  at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl
.java:43)
[2021-05-08T07:00:40.996Z]  at
java.lang.reflect.Method.invoke(Method.java:498)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.ContinuationPtr$ContinuationImpl.receive(Conti
nuationPtr.java:72)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.CollectionLiteralBlock$ContinuationImpl.dispat
```

```
ch(CollectionLiteralBlock.java:55)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.CollectionLiteralBlock$ContinuationImpl.item(C
ollectionLiteralBlock.java:45)
[2021-05-08T07:00:40.996Z]  at
sun.reflect.GeneratedMethodAccessor298.invoke(Unknown Source)
[2021-05-08T07:00:40.996Z]  at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl
.java:43)
[2021-05-08T07:00:40.996Z]  at
java.lang.reflect.Method.invoke(Method.java:498)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.ContinuationPtr$ContinuationImpl.receive(Conti
nuationPtr.java:72)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.impl.ConstantBlock.eval(ConstantBlock.java:21)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.Next.step(Next.java:83)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.Continuable$1.call(Continuable.java:174)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.Continuable$1.call(Continuable.java:163)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.GroovyCategorySupport$ThreadCategoryInfo.use(Gro
ovyCategorySupport.java:129)
[2021-05-08T07:00:40.996Z]  at
org.codehaus.groovy.runtime.GroovyCategorySupport.use(GroovyCategorySupport.
java:268)
[2021-05-08T07:00:40.996Z]  at
com.cloudbees.groovy.cps.Continuable.run0(Continuable.java:163)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsThread.runNextChunk(CpsThread.java:185
)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsThreadGroup.run(CpsThreadGroup.java:40
0)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsThreadGroup.access$400(CpsThreadGroup.
java:96)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsThreadGroup$2.call(CpsThreadGroup.java
```

```
:312)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsThreadGroup$2.call(CpsThreadGroup.java
:276)
[2021-05-08T07:00:40.996Z]  at
org.jenkinsci.plugins.workflow.cps.CpsVmExecutorService$2.call(CpsVmExecutor
Service.java:67)
[2021-05-08T07:00:40.996Z]  at
java.util.concurrent.FutureTask.run(FutureTask.java:266)
[2021-05-08T07:00:40.996Z]  at
hudson.remoting.SingleLaneExecutorService$1.run(SingleLaneExecutorService.ja
va:131)
[2021-05-08T07:00:40.996Z]  at
jenkins.util.ContextResettingExecutorService$1.run(ContextResettingExecutorS
ervice.java:28)
[2021-05-08T07:00:40.996Z]  at
jenkins.security.ImpersonatingExecutorService$1.run(ImpersonatingExecutorSer
vice.java:59)
[2021-05-08T07:00:40.996Z]  at
java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
[2021-05-08T07:00:40.996Z]  at
java.util.concurrent.FutureTask.run(FutureTask.java:266)
[2021-05-08T07:00:40.996Z]  at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:11
49)
[2021-05-08T07:00:40.996Z]  at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:6
24)
[2021-05-08T07:00:40.996Z]  at java.lang.Thread.run(Thread.java:748)
[2021-05-08T07:00:40.996Z] Finished: FAILURE
```

此时，在集群配置页面中，可能会同时看到集群名称后有异常提示，提示信息为“业务集群已经被删除，请配置新的业务集群”，或者也可能会看到集群配置页面中未配置集群，集群已被自动移除。

问题原因

当前承载流水线运行的Kubernetes集群已被删除，无法正常连接。

解决方案

方案一：直接更改Kubernetes集群

1. （可选）移除旧集群。

   当已删除集群未在集群配置页面被自动移除时，请执行以下操作移除该旧集群。反之，可跳过本步骤。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[集群配置]，进入“集群配置”页面。在该页面中，单击列表上方的 `移除` ，弹出“移除集群”提示框。在该提示框中，单击 `移除` ，移除该旧集群，并关闭提示框。

2. 添加新集群。

   在云平台的顶部导航栏中，依次选择[产品与服务]-[DevOps]-[集群配置]，进入“集群配置”页面。在该页面中，单击页面上方的 `添加集群` ，弹出“添加集群”对话框。在该对话框中，选择用于承载流水线运行的新集群后，单击 `添加` ，开始添加Kubernetes集群，并关闭对话框。

方案二：重新配置该Kubernetes集群

配置Kubernetes集群的具体操作步骤，请参考 配置集群。

当上述Kubernetes集群配置成功且状态为“健康”状态时，DevOps功能会自动恢复正常，可以直接按需重新执行上述流水线。

6.5 当执行流水线时提示 java.lang.OutOfMemoryError，如何排查解决?

问题描述

当执行流水线时，执行状态显示失败。之后，在查看此次运行记录的详情时，发现提示 **java.lang.OutOfMemoryError: unable to create new native thread** 报错。

```
15:10:47  java.lang.OutOfMemoryError: unable to create new native thread
15:10:47    at java.lang.Thread.start0(Native Method)
15:10:47    at java.lang.Thread.start(Thread.java:717)
15:10:47    at
hudson.remoting.AtmostOneThreadExecutor.execute(AtmostOneThreadExecutor.java
:95)
15:10:47    at
java.util.concurrent.AbstractExecutorService.submit(AbstractExecutorService.
java:112)
15:10:47    at
org.jenkinsci.remoting.util.ExecutorServiceUtils.submitAsync(ExecutorService
Utils.java:59)
15:10:47    at
hudson.remoting.JarCacheSupport.resolve(JarCacheSupport.java:63)
15:10:47    at
hudson.remoting.ResourceImageInJar._resolveJarURL(ResourceImageInJar.java:93
)
15:10:47    at
hudson.remoting.ResourceImageInJar.resolve(ResourceImageInJar.java:45)
15:10:47    at
hudson.remoting.RemoteClassLoader.findClass(RemoteClassLoader.java:306)
15:10:47    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
15:10:47    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
15:10:47    at java.lang.ClassLoader.defineClass1(Native Method)
15:10:47    at java.lang.ClassLoader.defineClass(ClassLoader.java:763)
15:10:47    at java.lang.ClassLoader.defineClass(ClassLoader.java:642)
15:10:47    at
hudson.remoting.RemoteClassLoader.loadClassFile(RemoteClassLoader.java:385)
```

```
15:10:47    at
hudson.remoting.RemoteClassLoader.findClass(RemoteClassLoader.java:309)
15:10:47    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
15:10:47    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
15:10:47    at java.lang.Class.forName0(Native Method)
15:10:47    at java.lang.Class.forName(Class.java:264)
15:10:47    at com.sun.proxy.$Proxy9.(Unknown Source)
15:10:47    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native
Method)
15:10:47    at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAcces
sorImpl.java:62)
15:10:47    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstruc
torAccessorImpl.java:45)
15:10:47    at
java.lang.reflect.Constructor.newInstance(Constructor.java:423)
15:10:47    at java.lang.reflect.Proxy.newProxyInstance(Proxy.java:739)
15:10:47    at
hudson.remoting.RemoteInvocationHandler.wrap(RemoteInvocationHandler.java:16
9)
15:10:47    at hudson.remoting.Channel.export(Channel.java:811)
15:10:47    at hudson.remoting.Channel.export(Channel.java:774)
15:10:47    at
org.jenkinsci.plugins.gitclient.LegacyCompatibleGitAPIImpl.writeReplace(Lega
cyCompatibleGitAPIImpl.java:204)
15:10:47    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
15:10:47    at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62
)
15:10:47    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl
.java:43)
15:10:47    at java.lang.reflect.Method.invoke(Method.java:498)
15:10:47    at
java.io.ObjectStreamClass.invokeWriteReplace(ObjectStreamClass.java:1230)
15:10:47    at
java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1136)
15:10:47    at
java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:348)
15:10:47    at hudson.remoting.UserRequest._serialize(UserRequest.java:263)
```

```
15:10:47    at hudson.remoting.UserRequest.serialize(UserRequest.java:272)
15:10:47    at hudson.remoting.UserRequest.perform(UserRequest.java:222)
15:10:47    at hudson.remoting.UserRequest.perform(UserRequest.java:54)
15:10:47    at hudson.remoting.Request$2.run(Request.java:369)
15:10:47    at
hudson.remoting.InterceptingExecutorService$1.call(InterceptingExecutorServi
ce.java:72)
15:10:47    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
15:10:47    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:11
49)
15:10:47    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:6
24)
15:10:47    at hudson.remoting.Engine$1.lambda$newThread$0(Engine.java:117)
15:10:47  Caused: java.io.IOException: Remote call on a37cc38f-9376-42cb-
a959-8813ab1b64d3-6-9hbvx-9tdk8-brg6j failed
15:10:47    at hudson.remoting.Channel.call(Channel.java:1004)
15:10:47    at hudson.FilePath.act(FilePath.java:1069)
15:10:47    at hudson.FilePath.act(FilePath.java:1058)
15:10:47    at org.jenkinsci.plugins.gitclient.Git.getClient(Git.java:121)
15:10:47    at hudson.plugins.git.GitSCM.createClient(GitSCM.java:861)
15:10:47    at hudson.plugins.git.GitSCM.createClient(GitSCM.java:833)
15:10:47    at hudson.plugins.git.GitSCM.checkout(GitSCM.java:1240)
15:10:47    at
org.jenkinsci.plugins.workflow.steps.scm.SCMStep.checkout(SCMStep.java:125)
15:10:47    at
org.jenkinsci.plugins.workflow.steps.scm.SCMStep$StepExecutionImpl.run(SCMSt
ep.java:93)
15:10:47    at
org.jenkinsci.plugins.workflow.steps.scm.SCMStep$StepExecutionImpl.run(SCMSt
ep.java:80)
15:10:47    at
org.jenkinsci.plugins.workflow.steps.SynchronousNonBlockingStepExecution.lam
bda$start$0(SynchronousNonBlockingStepExecution.java:47)
15:10:47    at
java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
15:10:47    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
15:10:47    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:11
49)
```

```
15:10:47    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:6
24)
15:10:47    at java.lang.Thread.run(Thread.java:748)
15:10:47  Finished: FAILURE
```

问题原因

在当前承载流水线运行的Kubernetes集群内，节点正在运行的线程数量已达到最大线程数上限，因此，无法创建新的线程，导致流水线执行失败。

解决方案

由于节点支持的最大线程数主要由物理内存决定，所以，可通过使用更高节点配置的Kubernetes集群，达到增加物理内存的目的，从而提升最大线程数上限；或通过扩容当前Kubernetes集群，增加工作节点，分担流水线执行任务。

其中，扩容Kubernetes集群的具体操作步骤，请参考“Kubernetes容器服务”帮助中“扩容集群”的相关内容。

咨询热线：400-100-3070

北京易捷思达科技发展有限公司：
北京市海淀区西北旺东路10号院东区23号楼华胜天成科研大楼一层东侧120-123
南京分公司：
江苏省南京市雨花台区软件大道168号润和创智中心B栋一楼西101
上海office：
上海黄浦区西藏中路336号华旭大厦22楼2204
成都分公司：
成都市高新区天府五街168号德必天府五街WE602

邮箱：
contact@easystack.cn (业务咨询)
partners@easystack.cn(合作伙伴咨询)
marketing@easystack.cn (市场合作)
training@easystack.cn (培训咨询)
hr@easystack.cn (招聘咨询)