DaoCloud Enterprise

DaoCloud Enterprise Product Documentation

DaoCloud

Copyright © 2016 - 2023 DaoCloud

Table of contents

1. 常见问题 3
1.1 执行流水线时报错 3
1.2 通过 Jenkins 构建镜像时，容器无法访问私有镜像仓库**** 3
1.3 在 GitOps 模块下添加 GitHub 仓库时报错 4
1.4 在某个工作空间下在 GitOps 模块中添加仓库时提示仓库已经存在 4
2. 离线升级 5
2.1 从安装包中加载镜像 5
2.2 升级 6
3. Intro 9
3.1 优势 9
3.2 基本概念 10
3.3 功能总览 11
3.4 应用工作台 Release Notes 12
3.5 适用场景 15
3.6 什么是应用工作台？ 16
4. Quickstart 20
4.1 基于 Argo Rollout 实现渐进式灰度发布 20
4.2 快速创建流水线 30
4.3 在指定的节点上运行流水线 42
4.4 集成 Harbor 实现镜像安全扫描 52
4.5 使用流水线实现代码扫描 55
4.6 Jenkins 场景配置 64
5. User guide 66
5.1 Gitops 66
5.2 Namespace 80
5.3 Pipeline 87
5.4 Release 134
5.5 Wizard 147

Copyright © 2016 - 2023 DaoCloud

1. 常见问题

本页列出使用应用工作台时可能遇到的一些问题并给出相应的解决方案。

1.1 执行流水线时报错

当 Jenkins 所在集群与应用部署集群跨数据中心时，网络通信延迟会变高，可能遇到如下的报错信息：

```
E0113 01:47:27.690555 50 request.go:1058] Unexpected error when reading response body: net/http: request canceled (Client.Timeout or context cancellation while reading body)
error: unexpected error when reading response body. Please retry. Original error: net/http: request canceled (Client.Timeout or context cancellation while reading body)
```

解决方案：

在该流水线的 Jenkinsfile 中将部署命令由 `kubectl apply -f` 修改为 `kubectl apply -f . --request-timeout=30m`

1.2 通过 Jenkins 构建镜像时，容器无法访问私有镜像仓库****

遇到此问题时可以在流水线的 Jenkinsfile 中增加如下命令：

```
cat > /etc/containers/registries.conf << EOF
[registries.insecure]
registries = ['temp-registry.daocloud.io']
EOF
```

```
        stage('build & push') {
          agent none
          steps {
            container('base') {
              withCredentials([usernamePassword(credentialsId:'harbor',passwordVariable:'PA
                sh '''cat > /etc/containers/registries.conf << EOF
[registries.insecure]
registries = [\'temp-registry.daocloud.io\']
EOF

docker login temp-registry.daocloud.io --tls-verify=false -u $USER -p $PASS'''
                sh '''
                  export IMG="temp-registry.daocloud.io/library/nginx:$IMAGE_TAG"
                  docker build -f Dockerfile -t $IMG  .
                  docker push $IMG --tls-verify=false'''
              }

            }

          }
        }

        stage('deploy') {
          agent none
```


Copyright © 2016 - 2023 DaoCloud

1.3 在 GitOps 模块下添加 GitHub 仓库时报错

由于 GitHub 移除了对 username/password 的支持，所以通过 HTTP 方式导入 GitHub 仓库时会导入失败，可能出现如下报错信息：

```
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on
currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/DaoCloud/dce-installer.git/'
```

解决方案：

使用 SSH 方式导入 GitHub 仓库。

1.4 在某个工作空间下在 GitOps 模块中添加仓库时提示仓库已经存在

目前一个仓库绑定只能到一个工作空间，不能绑定到不同工作空间。

Copyright © 2016 - 2023 DaoCloud

2. 离线升级

应用工作台支持离线升级。您需要先从安装包中加载镜像，然后执行相应命令进行升级。

2.1 从安装包中加载镜像

支持通过两种方式加载镜像。

通过 charts-syncer 同步镜像

如果环境中存在镜像仓库，建议通过 charts-syncer 将镜像同步到镜像仓库，更加高效便捷。

1. 创建 `load-image.yaml` 文件。

   注意：该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

   ```
   yaml
   source:
     intermediateBundlesPath: ghippo-offline # 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
   target:
     containerRegistry: 10.16.10.111 # 镜像仓库 url
     containerRepository: amamba # 镜像仓库的具体项目
     repo:
       kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别，比如 ChartMuseum
       url: http://10.16.10.111/chartrepo/amamba #  需更改为 chart repo url
       auth: # 用户名/密码
         username: "admin"
         password: "Harbor12345"
     containers:
       auth: # 用户名/密码
         username: "admin"
         password: "Harbor12345"
   ```

   !!! note "若当前环境未安装 chart repo，也可以通过 chart-syncer 将 chart 导出为 `tgz` 文件，并存放在指定路径。"

   ```
   ```yaml
   source:
       intermediateBundlesPath: amamba-offline #  到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
   target:
       containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
       containerRepository: release.daocloud.io/amamba # 需更改为你的镜像仓库
       repo:
         kind: LOCAL
         path: ./local-repo # chart 本地路径
       containers:
         auth:
           username: "admin" # 你的镜像仓库用户名
           password: "Harbor12345" # 你的镜像仓库密码
   ```
   ```

2. 执行如下命令同步镜像。

   ```
   bash
     charts-syncer sync --config load-image.yaml
   ```

通过 Docker 或 containerd 直接加载镜像

1. 执行下列命令解压镜像。

   `tar xvf amamba.bundle.tar`

   解压成功后会得到 3 个文件：`images.tar`、`hints.yaml`、`original-chart`。

Copyright © 2016 - 2023 DaoCloud

2. 执行如下命令从本地加载镜像到 Docker 或 containerd。

   注意：需要在集群中的 每个节点 上执行如下操作。加载完成后需要为镜像打标签，确保 Registry、Repository 与安装时一致。

   `shell`

   `docker load -i images.tar # for Docker`

   `ctr image import images.tar # for containerd`

2.2 升级

1. 检查应用工作台的 Helm 仓库是否存在。

   `helm repo list | grep amamba`

   若返回结果为空或出现 `Error: no repositories to show` 提示，则执行如下命令添加应用工作台的 Helm 仓库。

   `heml repo add amamba http://{harbor url}/chartrepo/{project}`

2. 更新应用工作台的 Helm 仓库。

   `helm repo update amamba # Helm 版本过低会导致失败。若失败，请尝试执行 helm update repo`

3. 备份 `--set` 参数。在升级全局管理版本之前，建议执行如下命令备份旧版本的 `--set` 参数。

   `bash`

   `helm get values ghippo -n ghippo-system -o yaml > amamba.bak.yaml`

4. 选择想安装的应用工作台版本（建议安装最新版本）。

   `helm search  repo amamba-release-ci --versions |head`

   `NAME  CHART VERSION  APP VERSION  DESCRIPTION`

   `amamba-release-ci/amamba  0.14.0  0.14.0  Amamba is the entrypoint to DCE 5.0, provides de...`

5. 修改 `amamba.bak.yaml` 文件里的 `registry` 和 `tag`。

   ??? note ″点击查看示例的 YAML 文件 “

```
amambaSyncer:
  resources:
    limits:
      cpu: 200m
      memory: 256Mi
    requests:
      cpu: 20m
      memory: 128Mi
apiServer:
  configMap:
    debug: true
  fromJar:
    image:
      registry: releas-ci.daocloud.io
      repository: docker/library/openjdk
      tag: 11.0-jre-slim
  image:
    registry: release-ci.daocloud.io
    repository: amamba/amamba-apiserver
  resources:
    limits:
      cpu: "2"
      memory: 2Gi
    requests:
      cpu: 20m
      memory: 150Mi
argo-cd:
  applicationSet:
    enabled: false
    image:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
  controller:
    image:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
    resources:
      limits:
        cpu: "2"
        memory: 2Gi
      requests:
        cpu: 100m
```


Copyright © 2016 - 2023 DaoCloud

```
        memory: 256Mi
  dex:
    enabeld: true
    image:
      repository: release-ci.daocloud.io/ghcr/dexidp/dex
      tag: v2.32.0
    initImage:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
    resources:
      limits:
        cpu: 50m
        memory: 64Mi
      requests:
        cpu: 10m
        memory: 16Mi
  enabled: true
  notifications:
    enabled: false
  redis:
    enabled: true
    image:
      repository: release-ci.daocloud.io/docker/library/redis
      tag: 7.0.4-alpine
    metrics:
      enabled: false
    resources:
      limits:
        cpu: 200m
        memory: 128Mi
      requests:
        cpu: 5m
        memory: 16Mi
  repoServer:
    image:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
    resources:
      limits:
        cpu: 200m
        memory: 256Mi
      requests:
        cpu: 5m
        memory: 8Mi
  server:
    image:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
    resources:
      limits:
        cpu: 250m
        memory: 256Mi
      requests:
        cpu: 5m
        memory: 8Mi
    service:
      nodePortHttp: 31886
      nodePortHttps: 31887
      type: NodePort
argo-rollouts:
  controller:
    image:
      registry: release-ci.daocloud.io
      repository: quay/argoproj/argo-rollouts
      tag: v1.2.0
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 16m
        memory: 128Mi
  dashboard:
    enabled: true
    image:
      registry: release-ci.daocloud.io
      repository: quay/argoproj/kubectl-argo-rollouts
      tag: v1.2.0
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 10m
        memory: 16Mi
    service:
      nodePort: 30070
      type: NodePort
  enabled: true
devopsServer:
  enabled: true
  image:
    registry: release-ci.daocloud.io
    repository: amamba/amamba-devops-server
```


Copyright © 2016 - 2023 DaoCloud

```
    resources:
      limits:
        cpu: "2"
        memory: 2Gi
      requests:
        cpu: 50m
        memory: 160Mi
  global:
    amamba:
      imageTag: v0.13-dev-a8ca782a
    imageRegistry: release-ci.daocloud.io
  mysql:
    busybox:
      repository: docker/busybox
      tag: 1.35.0
    image:
      repository: docker/mysql
      tag: 5.7.30
    persistence:
      size: 20Gi
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 20m
        memory: 256Mi
  ui:
    image:
      tag: v0.15.0-dev-fd64789e
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 5m
        memory: 4Mi

  ```
```

6. 执行如下命令进行升级

```
helm upgrade amamba . \
  -n amamba-system \
  -f ./amamba.bak.yaml
```


Copyright © 2016 - 2023 DaoCloud
```

3. Intro

3.1 优势

「DaoCloud 道客」凭借多年云原生领域的领先经验，全新开发的应用工作台具有以下产品优势：

- 企业级 CI/CD

  使用集中管理的容器化工作流，持续开发持续交付，支持横向扩展，可以在高可用性 (HA) 环境中支持成千上万个并发用户和流水线。

- 云原生为底座

  应用工作台实现了应用构建环境的容器化管理，自定义构建机器等资源，确保流水线级资源隔离，同时支持多种类型的应用部署，使得云原生应用程序的交付更加简单。

- 提升研发效能

  在可视化的流水线构建环境中，预先封装声明性步骤，无需脚本就能创建复杂的流水线，操作简单。可根据资源大小自动创建/销毁环境，避免资源浪费。

- 灰度发布

  帮助用户渐进式更新应用，从而保障新特性能平稳上线。支持金丝雀部署、蓝绿部署等应用发布策略。

  如果一个应用的发布策略配置为 ABTest，其 A 版本访问流量为 80%，B 版本访问流量为 20%，则灰度发布后将能够持续访问应用的两个版本，轻松统计出该应用不同版本的分流比例。

- GitOps

  提供基于 Kubernetes 的声明性 GitOps 持续交付能力，通过版本控制应用，从而实现自动化的应用部署与生命周期管理。

- 流水线即代码

  基于声明性（语法）的流水线易于学习，不同步骤的语法实现了标准化，其配置具有版本可控、模块化、可重用和声明性等特点。

- 综合集成

  集成社区流行的 DevOps 工具，单个流水线中的步骤可以在多操作系统、多体系结构节点上运行，支持在公有云、私有云或主机上独立部署，能很好地集成企业自有的系统和平台。 支持集成业界主流软件，如 Kubernetes、GitLab、SonarQube、Harbor 等。

- 多年行业沉淀

  在云原生领域耕耘多年，在金融、电商、制造等行业上百家企业有大规模实践经验，能够提供稳定可靠的 IT 交付生产线，帮助企业打造适合研发运行一体化平台。

Copyright © 2016 - 2023 DaoCloud

3.2 基本概念

应用工作台提供了 DCE 5.0 应用部署的统一入口，支持云原生应用的整个生命周期管理。 降低了企业使用云原生应用门槛，提高了企业软件研发到应用交付的效率。

- 工作空间 Workspace

  通过工作空间协调全局管理和子模块的权限关系，解决资源聚合和映射层级关系。 并且一个工作空间对应一个项目，可以为每个工作空间分配不同的资源，指派不同的用户和用户组。

- 命名空间 Namespace

  命名空间在平台上是工作空间下相互隔离的更小的资源空间，也是用户实现作业生产的工作区间。 一个工作空间下可以创建多个命名空间，可占用的资源配额总和不能超过工作空间配额。 命名空间更细粒度的划分了资源配额的同时，还限制了命名空间下容器的大小（CPU、内存），有效的提升了资源利用率。

- 流水线 Pipeline

  流水线提供可视化、可定制的自动交付流水线，帮助企业缩短交付周期，提升交付效率。目前流水线是基于 Jenkins 实现。

- 凭证 Credential

  流水线与第三方应用程序进行交互时，需要用户配置 Jenkins 凭证，流水线就可以与第三方应用程序交互。

- GitOps

  GitOps 的核心观点是使用包含当前期望的(生产环境基础设施的)声明式描述的 GIt 仓库，并通过自动化流程来确保生产环境与仓库中的期望状态保持一致。 如果您想要部署一个新的应用或更新一个现有的应用，只需要更新相应的仓库即可(自动化流程会处理后续的事情)。这就像在生产中使用巡航控制来管理应用程序一样。

- 灰度发布

  灰度发布是可以帮助用户渐进式更新应用的工具。它实现了多版本共存，发布暂停，流量百分比切换等功能，极大解放灰度发布过程中的手动操作，全自动化实现线上灰度流量切换。

Copyright © 2016 - 2023 DaoCloud

3.3 功能总览

应用工作台提供了以下功能。

- 应用管理
- 支持云原生场景下的“多形态”的云原生应用，包括 Kubernetes 原生应用、Helm 应用、OAM 应用等。
- 提供云原生应用的全生命周期管理，例如应用的扩缩容、日志、监控查看、更新应用等。
- 支持接入 SpingCloud、Dubbo、ServiceMesh 框架的微服务应用，实现微服务的治理，与 DCE 5.0 的微服务引擎、服务网格无缝集成。
- 流水线编排

  流水线是一个自定义的 CI/CD 模式。
- 支持自定义创建、基于代码仓库的 jenkinsfile 创建流水线两种模式。
- 支持流水线图形化编辑。
- 可通过源码、Jar 包的方式来构建应用。
- 凭证管理

  为流水线中的代码库、镜像仓库提供不同类型的凭证管理功能。
- 持续部署

  引入 GitOps 理念实现应用持续部署，用于把控代码构建之后的应用发布与部署交付流程。
- 基于 Argo CD，以自动化方式，频繁且持续性地将企业的应用部署到生产环境。
- 提供 Argo CD Application 的创建、同步、删除管理。
- 仓库管理

  支持导入 Git 的代码仓库，导入后您可以在持续部署中使用该仓库进行应用的持续部署。
- 灰度发布

  灰度发布可以保证整体系统的稳定，在初始灰度时就可以发现、调整问题，减少问题的影响范围。
- 支持金丝雀发布、蓝绿部署、A/B Testing 高级发布策略。
- 金丝雀发布支持自动化渐进式的发布流程。
- 支持通过监控指标分析来快速回滚应用。

Copyright © 2016 - 2023 DaoCloud

3.4 应用工作台 Release Notes

本页列出应用工作台的 Release Notes，便于您了解各版本的演进路径和特性变化。

2022-02-22

v0.14.0

新功能

- 新增 API 支持使用内置模板创建流水线
- 新增 API 支持自定义模板的创建、更新、删除
- 新增 apiserver、devopserver 支持访问 https 协议的 Jenkins
- 新增 event-proxy 组件支持 Jenkins 的事件持久化
- 新增 访问 Jenkins 时支持重试和限流

优化

- 优化 优化了重新运行某条流水线失败时的提示语

修复

- 修复 修复了凭证在数据库和 Jenkins 里不一致时的问题
- 修复 修复了创建灰度发布时流量比例大于 100 也能创建成功的问题

2022-01-30

v0.13.0

新功能

- 新增 支持级联删除，可以根据实际情况选择需要删除的资源
- 新增 灰度发布支持编辑 YAML 功能
- 新增 部署应用工作台的 Chart 包中移除了 Jenkins，支持使用单独的 Helm Chart 安装 Jenkins

优化

- 优化 制品报告下载，不在 apiserver 中全部缓存，采用 io 分块的模式部分缓存转发

修复

- 修复 修复了持续部署中添加私有仓库报错问题

2022-12-30

v0.12.0

新功能

- 新增 创建应用时支持同时开启传统微服务和服务网格
- 新增 查询灰度发布支持模糊查询，查询持续部署支持模糊查询和根据状态检索，查询 gitops 仓库支持模糊查询和状态检索
- 新增 添加根据 GVR 获取 jso 数据的接口
- 新增 添加获取 helm 信息的接口，可以通过 argocd 部署 helm chart
- 新增 Jenkins 配置文件的热加载以及 syncer 配置文件的热加载
- 新增 ListRegistry 接口返回值新增 Nacos 是否需要认证字段
- 新增 流水线制品报告列表接口和下载制品报告接口

Copyright © 2016 - 2023 DaoCloud

- 新增 流水线步骤日志支持获取全量日志
- 新增 查询插件信息接口添加了缓存
- 新增 直接复制已有的流水线来生成新的流水线
- 新增 开启微服务时接入指标监控提供差异化配置，可自定义路径
- 新增 Jenkins 的 trace 的对接
- 新增 ListWorkload 接口返回值添加应用关联资源

修复

- 修复 重新运行流水线，概率性的 panic 的修复
- 修复 创建微服务时链路追踪中用到的 instrumentation 镜像离线化问题
- 修复 灰度发布的监控分析失败
- 修复 支持设置 argo-rollouts controller replicas

2022-12-18

v0.11.0

新功能

- 新增 Ghippo 和 Skoala 的 trace 对接，Ghippo 自身只支持部分 api 的 trace
- 新增 helm 包离线情况下的镜像检查的流水线
- 新增 针对添加迁移 jenkins 后，credential 失效的问题,添加解决方案
- 新增 凭证列表支持按名称模糊搜索

优化

- 优化 通过调整 jvm 参数，降低 jenkins 在没有工作负载的情况下的内存占用
- 优化 gitops 模块创建仓库时增加对仓库可连接性的判断
- 优化 实时同步流水线运行状态等信息导致 jenkins 压力过大，改成懒加载的方式
- 优化 优化了 api 发版流程，支持单独发版

修复

- 修复 启用微服务的情况下如果创建失败则无限制进行回滚的问题
- 修复 当集群的 kubeconfig 变更导致的 argocd 应用部署/删除失败。以及 jenkins 中 credential 失效的问题

2022-11-30

v0.10.0

新功能

- 新增 新增了 gitops 中的仓库功能，支持导入、删除
- 新增 新增了 gitops 应用的同步功能

优化

- 优化 优化了应用接入服务网格流程

修复

- 修复 修复了 admin 用户未对部署目标（cluster/namespace）鉴权的问题
- 修复 修复了 gitops 应用创建时间、同步开始时间和同步结束时间为 `Invalida date` 的错误
- 修复 修复了获取 nacos 注册中心列表数据的错误
- 修复 修复了列出工作负载接口通过名称排序报错

Copyright © 2016 - 2023 DaoCloud

- 修复 修复了集群解绑并重新绑定后，在 ArgoCD 中 destionation 中的 cluster 和 namespace 丢失的问题
- 修复 修复了在更新 namespace 的 label 导致 namespace 和 workspace 绑定关系丢失的问题
- 修复 修复了在完成流水线后，同步 jenkins 的 config 到数据库时，trigger 转换的错误
- 修复 修复了因为集群的 kubeconfig 变更导致的 ArgoCD cluster 和 jenkins 中 kubeconfig 类型的 credential 不同步的问题
- 修复 修复了仓库列表出现的无序和分页问题
- 修复 修复了 from-jar 上传超过 32M 文件失败的问题

2022-11-18

v0.9.0

新功能

- 新增 jenkins-agent 镜像持续发布
- 新增 添加了使用中间件的数据库的选项
- 升级 jenkins 从 2.319.1 升级到 2.346.2,kubernetes plugin 升级到 3734.v562b_b_a_627ea_c，相关的 plugins 也作了升级

优化

- 优化 获取 rollout 镜像列表，应用组列表，原生应用列表等性能优化
- 优化 from-jar 用到的 image 不再写死在源代码中，通过 env 的方式传递，并保证安装器能正确获取

修复

- 修复 rollout 在不同 workspace 下无法区分的问题
- 修复 gitops 模块未鉴权的问题
- 修复 偶现的 pipeline step 状态不正确的问题
- 修复 获取 helmchart 的 description 为空的问题
- 修复 创建 namespace 没有校验 storage 的问题
- 修复 list argocd repository 出现的无序和分页问题的修复
- 修复 from-jar 上传超过 32M 文件失败的问题的修复
- 修复 获取 pipeline 日志的时候如果日志量过大则无法获取全量日志的问题

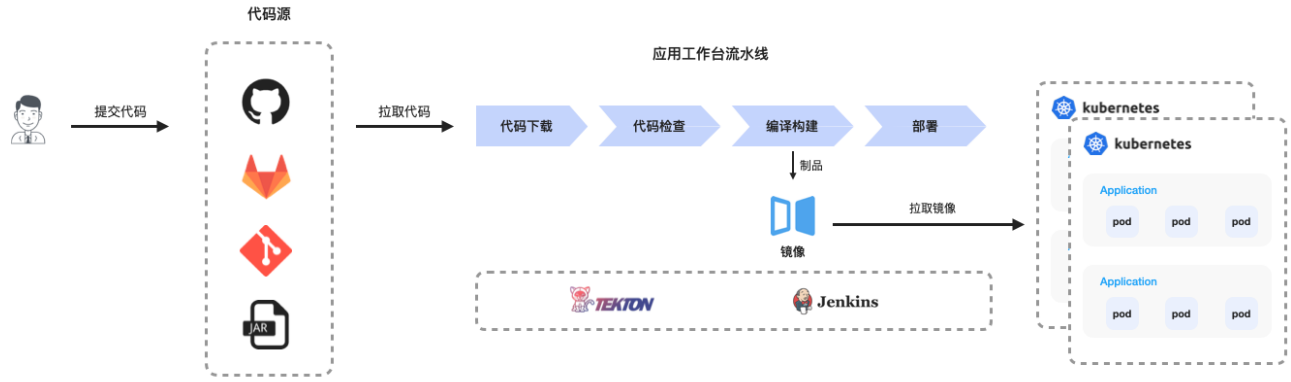
Copyright © 2016 - 2023 DaoCloud

3.5 适用场景

- 持续交付流水线

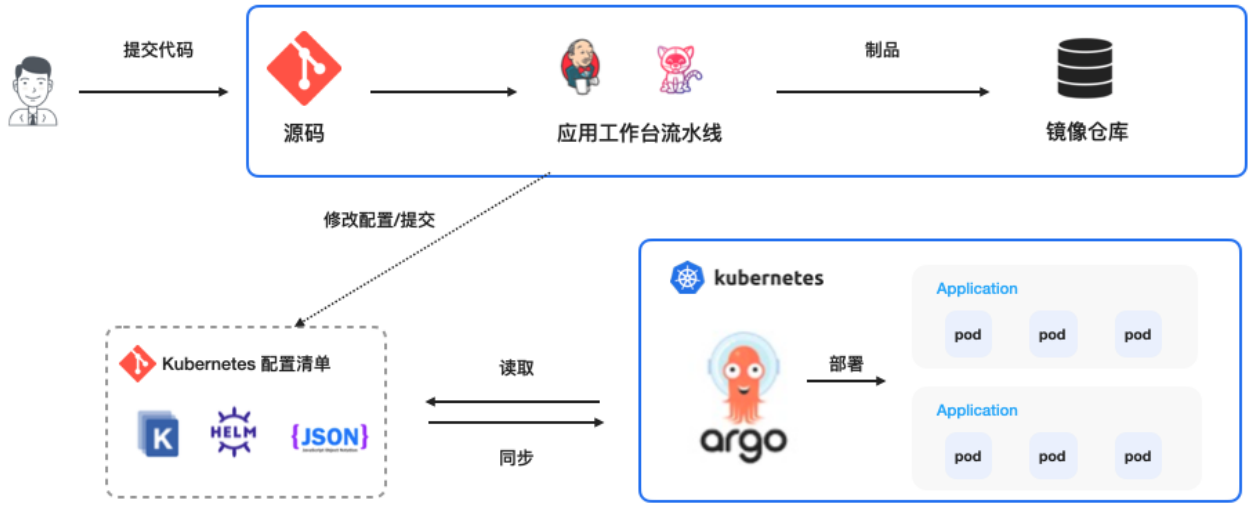
  对于复杂的业务系统，从项目创建、编译、构建、自验、集成验证、类生产验证、上线的各个阶段都需要耗费大量的人力和时间，并且容易受到人为因素影响而出错。 持续集成和持续交付由于具有标准化和自动化特点，以代码变更为流动单元，基于发布流水线，拉通开发、测试、运维所有职能，持续、快速、高可靠地发布软件，可以很好的解决该问题。

  自动化 CI/CD 流水线



- 基于流水线 + GitOps 实现云原生下的应用持续交付

  得益于 GitOps 理念，因此采用开源软件 Argo CD 来实践 Kubernetes 的应用发布。 用户仅需将 Kubernetes 的 YAML 文件提交至代码仓库中，GitOps 会自动感知到 YAML 文件的变化，配合代码仓库的合并请求功能， 将 YAML 文件的变更自动推送至集群中，整个过程无需学习 Kubernetes 的发布命令，也无需直接操作集群。



Copyright © 2016 - 2023 DaoCloud

3.6 什么是应用工作台?

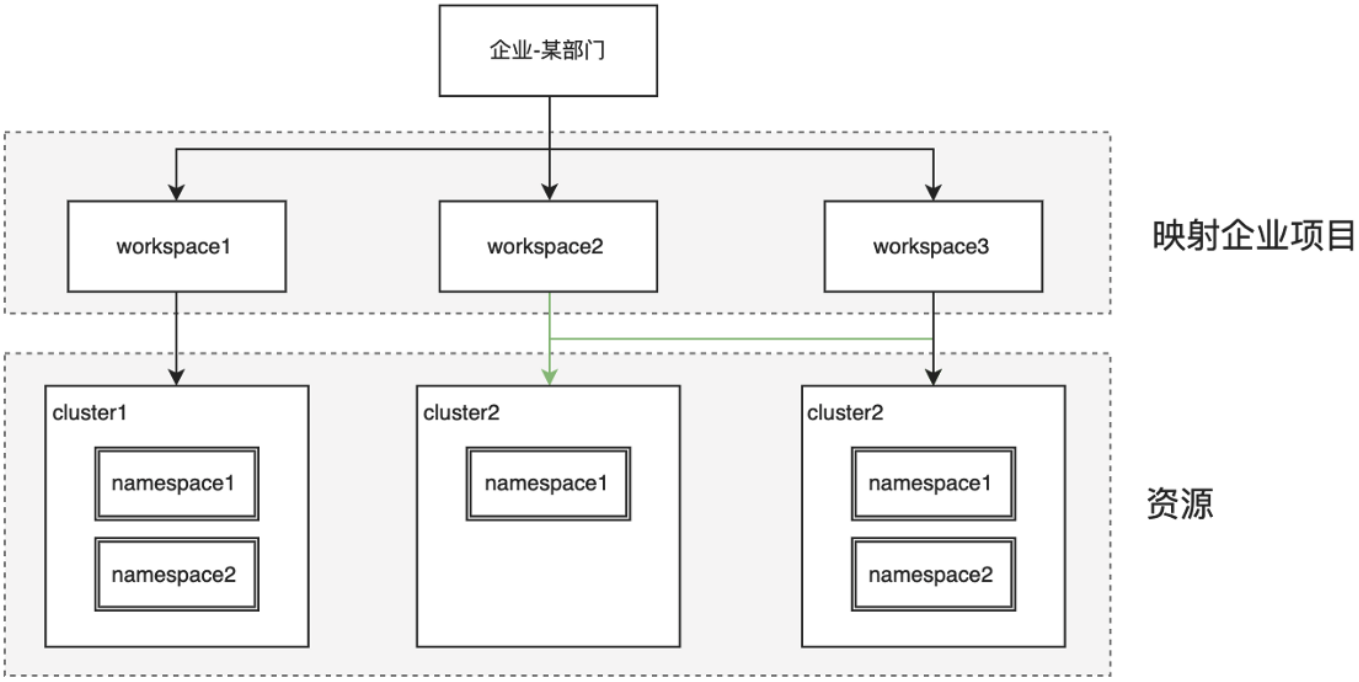
应用工作台是基于容器的 DevOps 云原生应用平台，提供了 DCE 5.0 应用创建的统一入口，完善的层级多租户方案，并且支持跨集群的资源部署。

应用工作台着重企业应用自动化交付和基础设施变更的过程，提供业务应用从“开发 -> 测试 -> 部署 -> 运维”的全生命周期管理，能有效帮助企业实现数字化转型，提升企业的 IT 交付能力和竞争力。

=== "层级多租户资源管理"

以工作空间作为最小的租户单元，不仅支持单集群的独享资源的能力，还支持跨集群共享资源的能力：

- 支持弱绑定集群，从而获得跨集群、跨命名空间共享资源的能力
- 支持强绑定集群，从而获得独享资源的能力
- 工作空间管理成员可以在关联的集群中创建命名空间资源
- 自助式资源创建的模式，用户可以在工作空间内自助创建命名空间来进行资源隔离



=== "以云原生应用为中心"

支持云原生场景下的“多形态”的云原生应用，包括 Kubernetes 原生应用、Helm 应用、OAM 应用等。 能够接入 SpingCloud、Dubbo、ServiceMesh 框架的微服务应用，实现微服务的治理，与 DCE 5.0 的微服务引擎、服务网格无缝集成。 支持云原生应用的全生命周期管理，例如应用的扩缩容、日志、监控查看、更新应用等等。

Copyright © 2016 - 2023 DaoCloud

Kubernetes 原生应用

Helm **应用**

OAM(基于KubeVela)

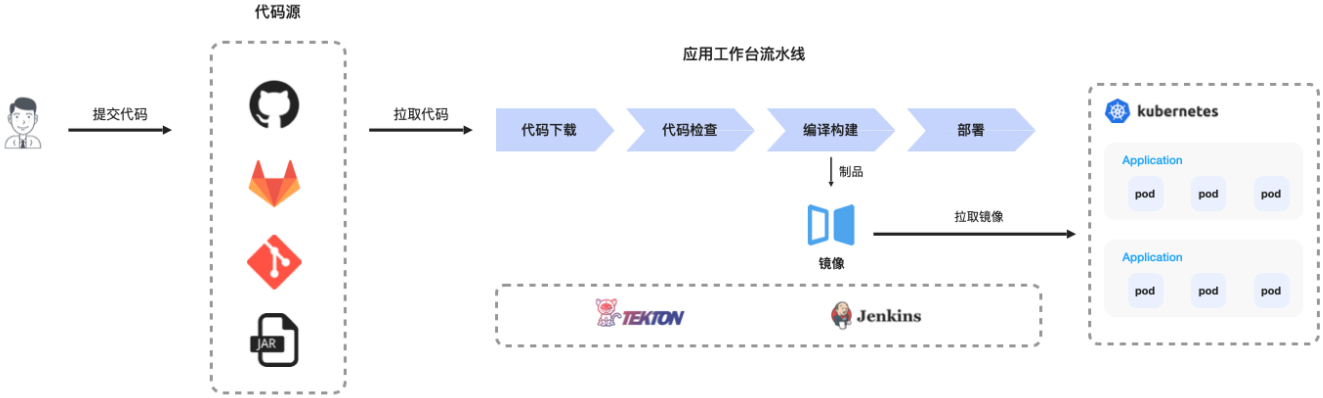
OLM(Operator Lifecycle Manager)

Argo CD Application

多云应用（Karmada）

=== "高效的持续集成"

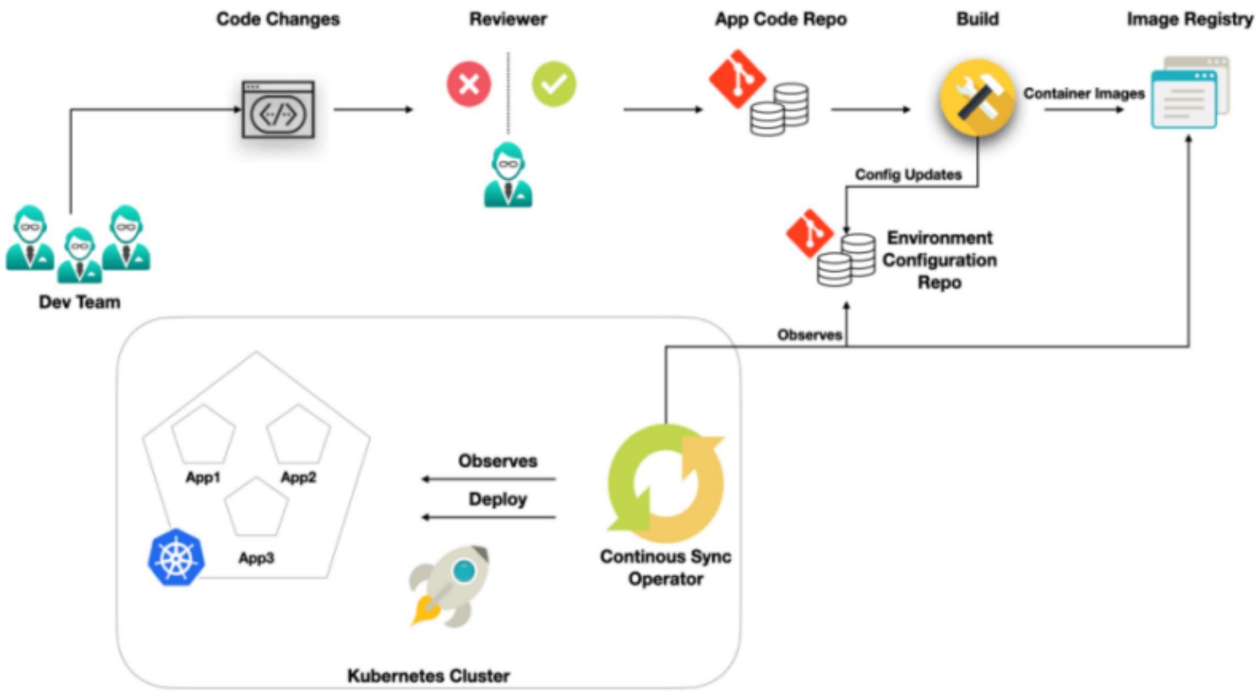
支持 Jenkins 和 Tekton 双流水线引擎系统。采用图形化方式编辑流水线，做到所见即所得的效果。可以用不同来源的代码构建应用。



=== "安全自动化渐进式交付"

应用工作台引入一种为云原生应用实现持续部署的理念 - GitOps，全面拥抱 GitOps，以集成渐进式交付组件 Argo Rollout，支持灰度发布，从而提高应用交付的稳定性和效率。

Copyright © 2016 - 2023 DaoCloud



!!! info

渐进式交付是一种将应用程序的新版本逐步暴露给最初的一小部分用户，然后逐渐变得越来越大的子集的做法，以减轻负面影响（例如错误）的风险。

Argo-Rollout Kubernetes Progressive Delivery Controller，提供更强大的部署能力。包括灰度发布、蓝绿部署、更新测试 (experimentation)、渐进式交付 (progressive delivery) 等特性。

在 DCE 5.0 中的地位

应用工作台在 DCE 5.0 中的地位如下图所示。

以容器管理为底座，借助全局管理实现层级资源管理，以 CI/CD 流水线和 GitOps 流程增删改查云原生应用，实现渐进式交付。

Copyright © 2016 - 2023 DaoCloud

部署方法

依次执行以下命令进行部署。

```
export VERSION=**** # 修改为实际部署的版本。
helm repo add mspider-release https://release.daocloud.io/chartrepo/amamba
helm repo update amamba
helm upgrade --install --create-namespace --cleanup-on-fail amamba amamba-release/amamba -n amamba-system --version=${VERSION}
```

下载 DCE 5.0{ .md-button .md-button--primary } 安装 DCE 5.0{ .md-button .md-button--primary } 申请社区免费体验{ .md-button .md-button--primary }

Copyright © 2016 - 2023 DaoCloud

4. Quickstart

4.1 基于 Argo Rollout 实现渐进式灰度发布

本文介绍如何基于开源的 Argo Rollout 实现渐进式灰度发布。

前提条件

- 示例中的镜像需要访问公网：`argoproj/rollouts-demo:yellow` 和 `argoproj/rollouts-demo:blue`。
- 仅适用于通过安装器以 metallb 方式部署出来的 DCE 5.0 平台。
- 使用灰度发布能力需要所在集群中安装 Istio 和 Argo Rollout 组件。

操作步骤

整个过程分为四个步骤：首先基于容器镜像构建应用，然后配置 Istio 相关资源，创建灰度发布任务，最后验证效果。

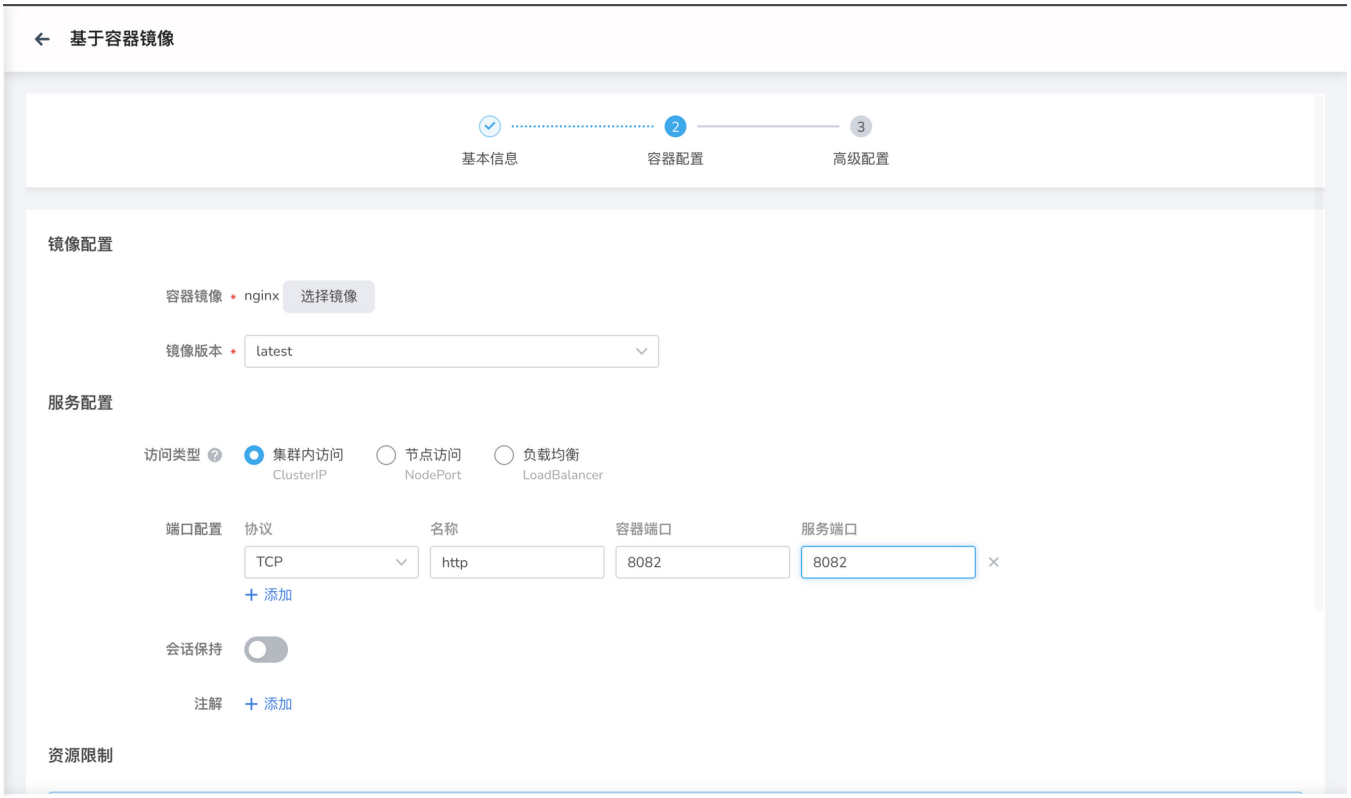
基于容器镜像构建应用

具体操作步骤可参考基于 Git 仓构建微服务应用、基于 Jar 包部署 Java 应用。

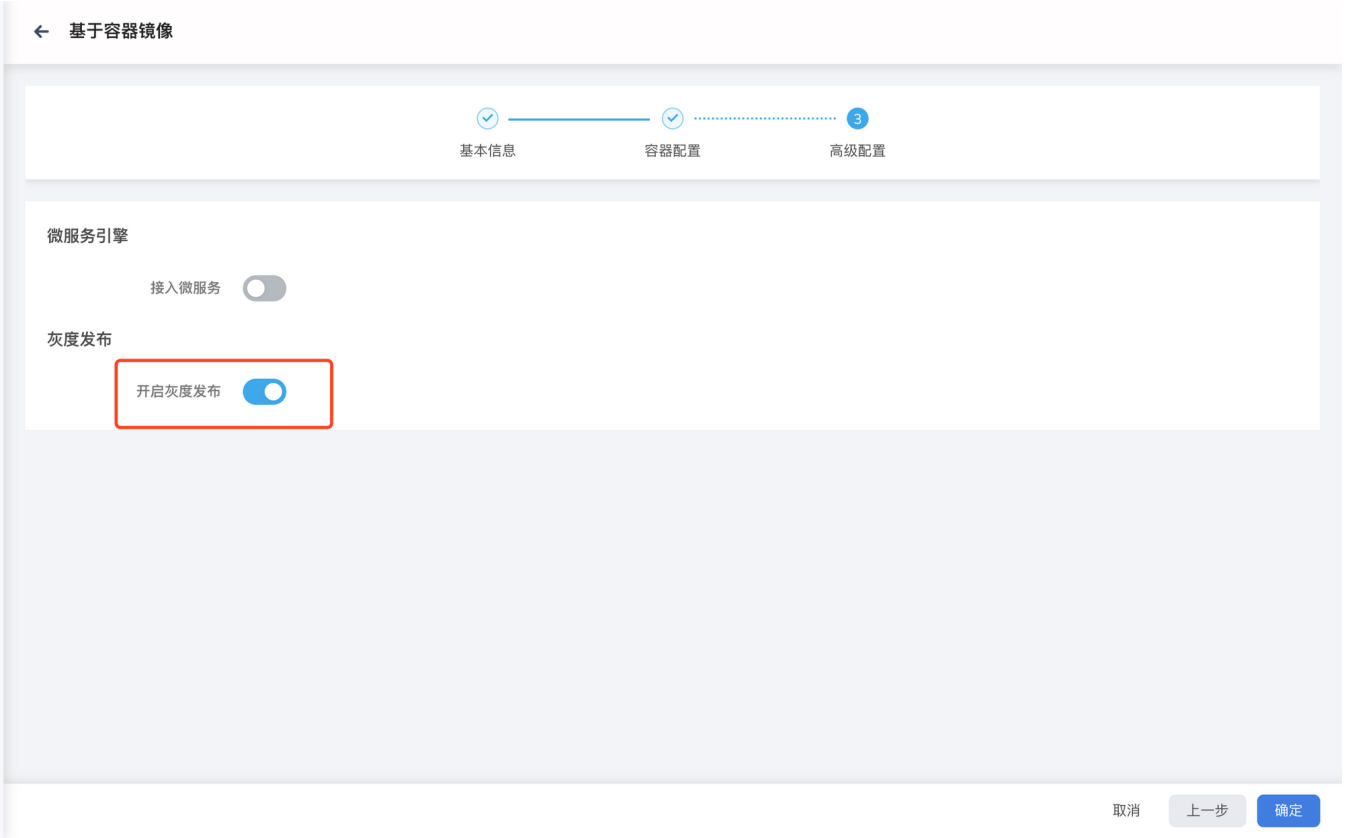
Copyright © 2016 - 2023 DaoCloud

需要注意：

- 容器镜像为：argoproj/rollouts-demo:blue
- 服务端口：名称为 `http`、容器端口为 `8082`、服务端口为 `8082`。



- 填写高级配置时需要开启灰度发布。



Copyright © 2016 - 2023 DaoCloud

Istio 相关资源配置

在服务网格模块或者控制台创建以下资源。

Copyright © 2016 - 2023 DaoCloud

1. 创建 Gateway

```yaml title="gateway.yaml"
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: rollout-demo
  namespace: rollout-demo # (1)
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*'
    port:
      name: http
      number: 8082
      protocol: HTTP
```

a. 部署应用的命名空间

2. 部署 Gateway

```shell
vi gateway.yaml
kubectl apply -f gateway.yaml
```

3. 修改 VirtualService

```shell
kubectl edit vs {#创建应用的名字} -n {#创建应用的命名空间}
```

仅需修改提示的字段，其余字段无需修改。

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  creationTimestamp: "2022-11-07T10:46:23Z"
  generation: 84
  name: demo01
  namespace: default
  resourceVersion: "5741370"
  uid: 8109f754-aa9d-49f1-b8a9-d4daf5108032
spec:
  gateways:
  - rollout-demo # (1)
  hosts:
  - '*' # (2)
  http:
  - name: primary
    route:
    - destination:
        host: demo01
        subset: stable
      weight: 100
    - destination:
        host: demo01
```


Copyright © 2016 - 2023 DaoCloud

```
        subset: canary
      weight: 0
```

a. 修改此处，需要新增 gateway，指向上一步创建的 gateway 名称

b. 修改此处，原来 host 为 vs 的名称，需要删除后更改为 `'*'`

4. 配置 istio-ingressgateway 网关

```shell
kubectl edit svc istio-ingressgateway -n istio-system
```

仅需修改提示的字段，其余字段无需修改。

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: istio-ingressgateway
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: istio-ingressgateway
    app.kubernetes.io/version: 1.15.0
    helm.sh/chart: gateway-1.15.0
    istio: ingressgateway
  name: istio-ingressgateway
  namespace: istio-system
  resourceVersion: "5775680"
  uid: 53bd7344-ba45-4547-b695-aca2c4dd713d
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 100.66.222.131
  clusterIPs:
  - 100.66.222.131
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: status-port
    nodePort: 32384
    port: 15021
    protocol: TCP
    targetPort: 15021
  # 新增以下内容
  - name: rollout-demo
    port: 8082
    protocol: TCP
    targetPort: 8082
  # --------------
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  sessionAffinity: None
  type: LoadBalancer
```

5. 创建 AuthorizationPolicy 资源

此步骤的目的是在浏览器访问是可以正常访问到 js 相关的资源。

Copyright © 2016 - 2023 DaoCloud

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: demo01
  namespace: istio-system
spec:
  rules:
  - to:
    - operation:
        paths:
        - /*
  - from:
    - source:
        requestPrincipals:
        - '*'
  selector:
    matchLabels:
      app: istio-ingressgateway
status:
  loadBalancer:
    ingress:
    - ip: 10.29.135.48
```

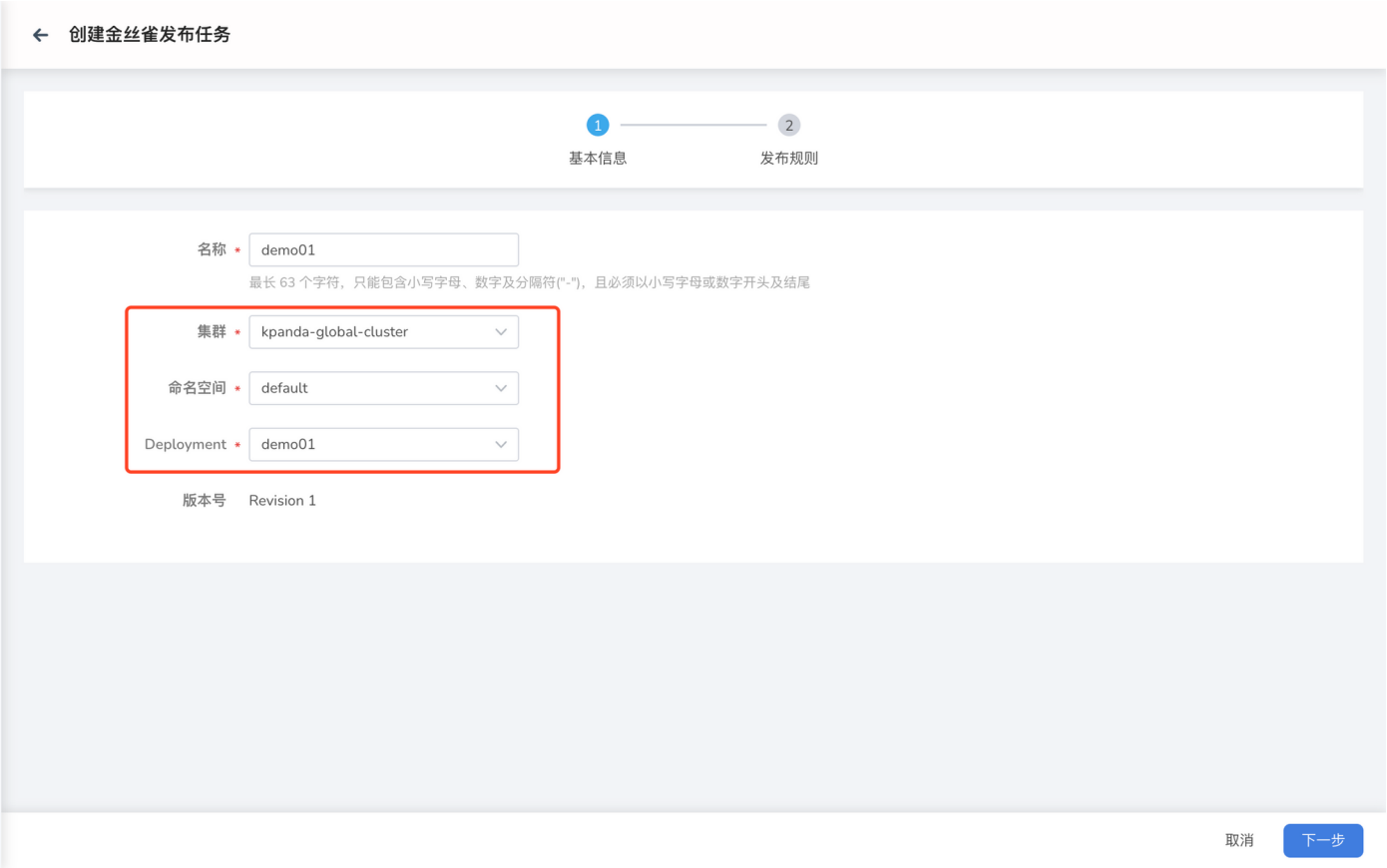

Copyright © 2016 - 2023 DaoCloud

创建灰度发布任务

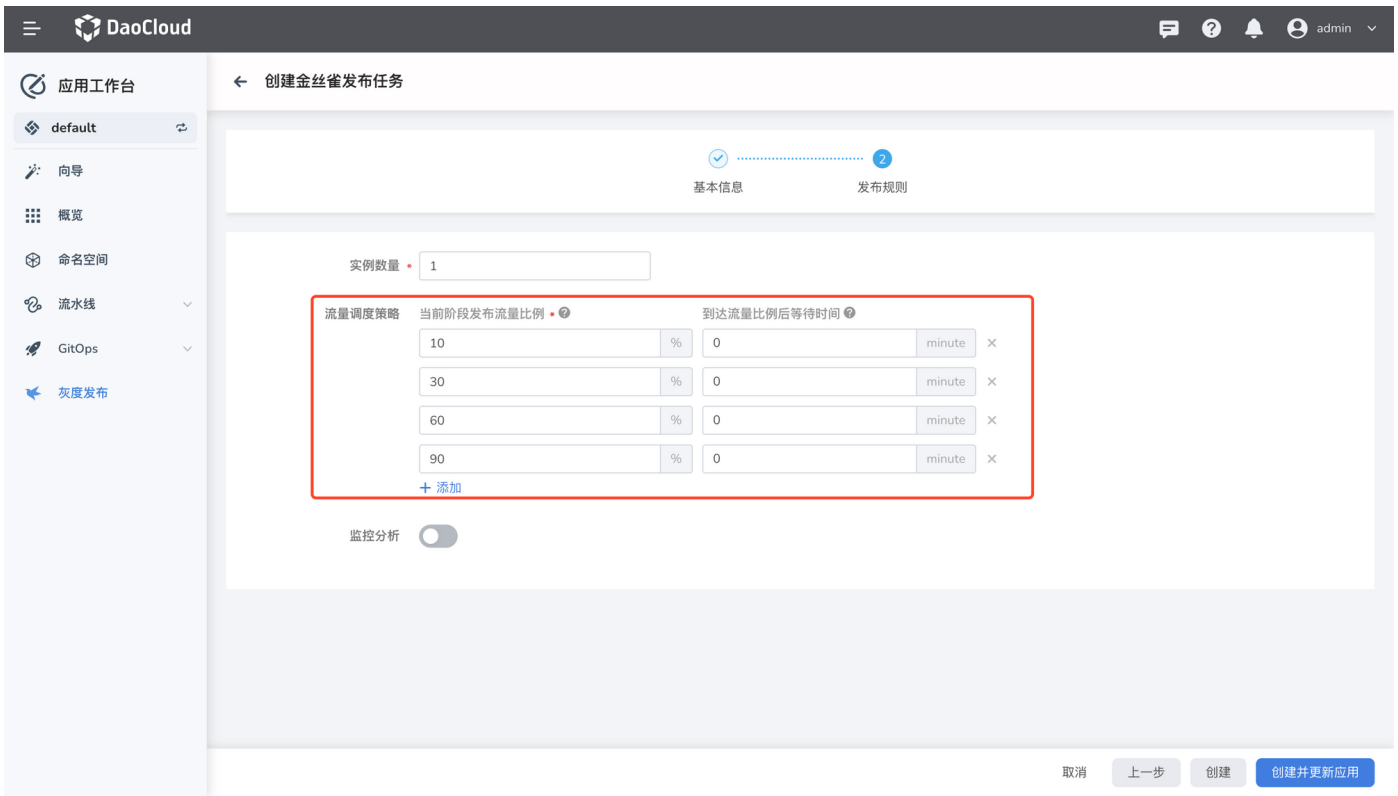
在应用工作台创建灰度发布任务。如需了解更详细的创建说明，可参考创建金丝雀发布任务。

Copyright © 2016 - 2023 DaoCloud

1. 选择开启灰度发布的应用

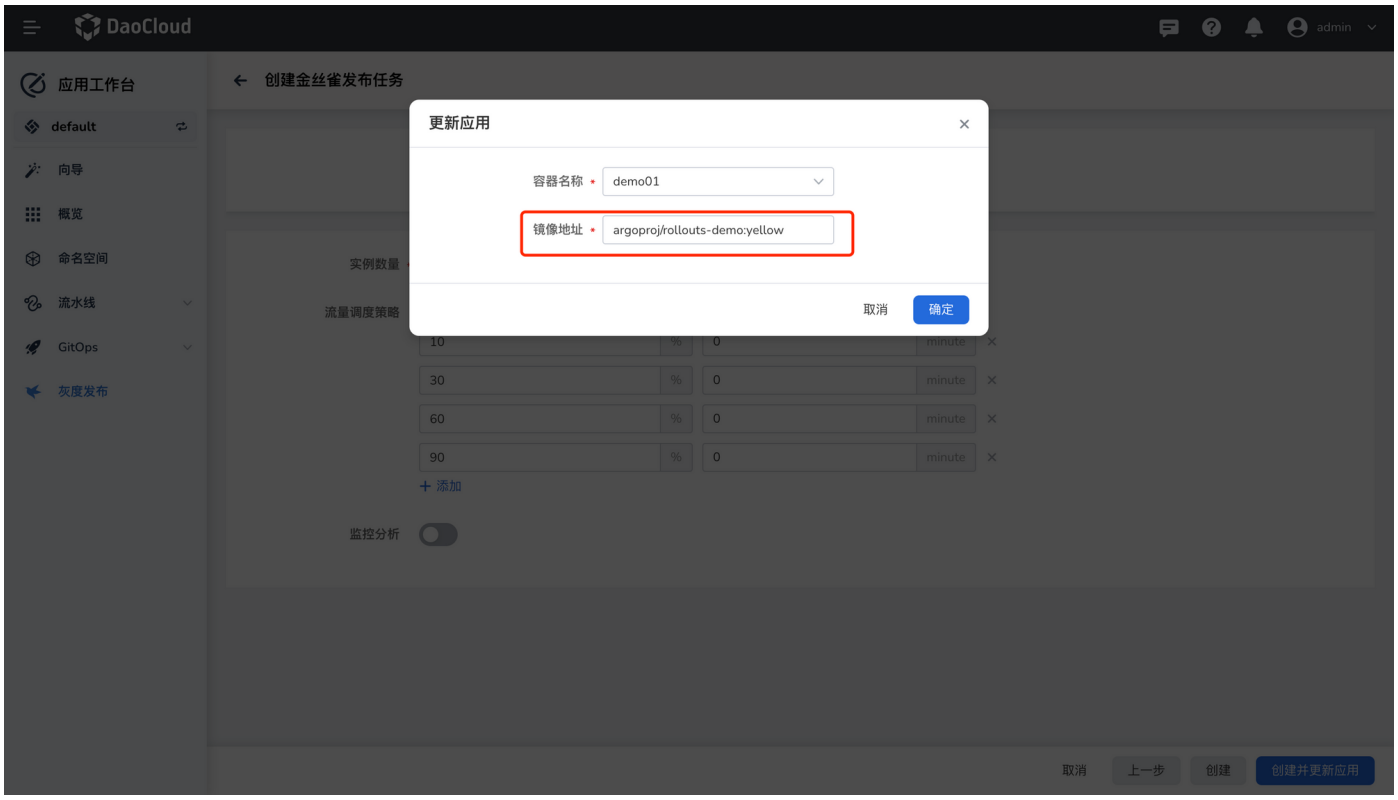


2. 修改发布规则（为了更明显的演示效果）



3. 点击创建并更新应用，在弹出的对话框中填写镜像地址：`argoproj/rollouts-demo:yellow`

Copyright © 2016 - 2023 DaoCloud



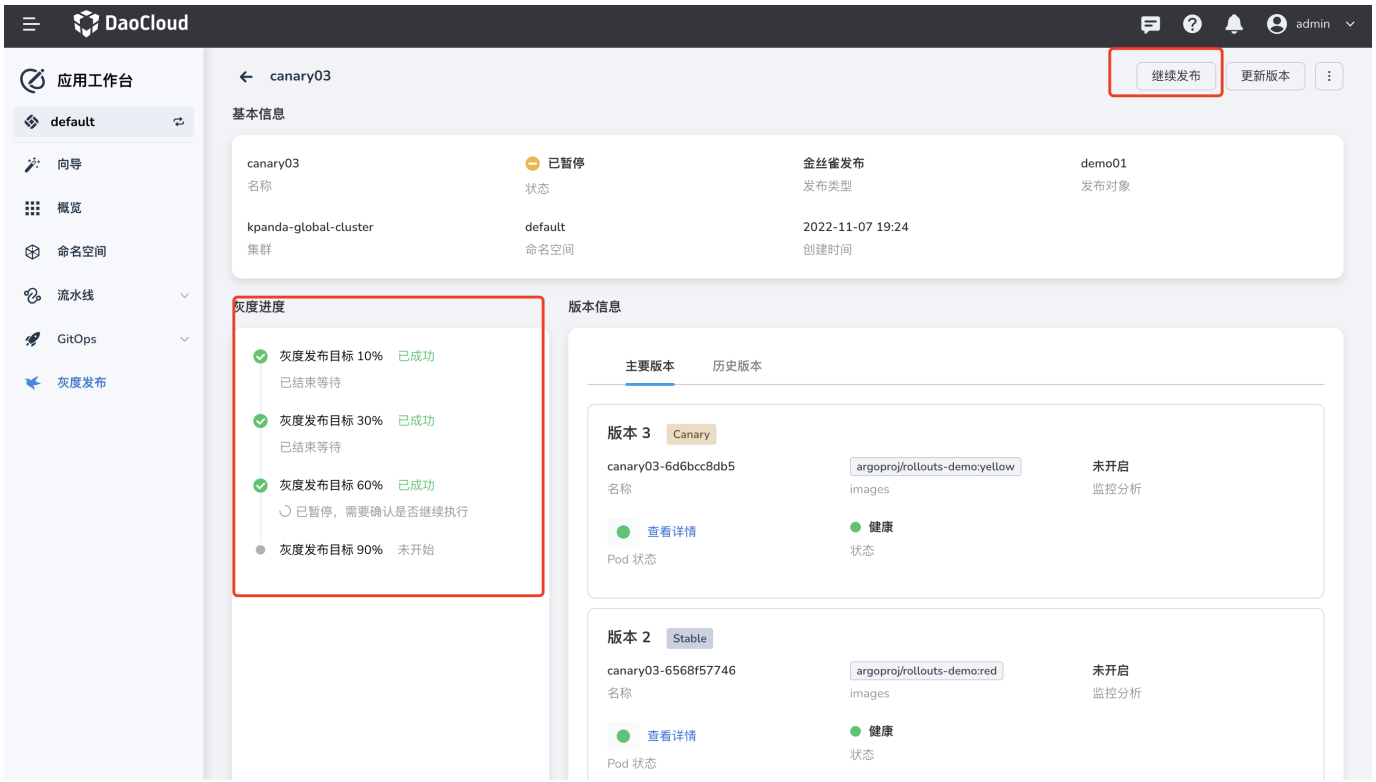
验证效果

访问地址：`http://{istio-ingressgateway LB IP}:8082`，得到如下所示的访问效果。

此界面会并发调用 `http://{istio-ingressgateway LB IP}:8082/color`，将获取到颜色信息填充到方格中。 在灰度发布对象中，指定的颜色为 blue、yellow，会按照定义规则 1:9 的流量比进行展示。

此时可以在应用工作台灰度发布模块选择继续发布应用来调整流量比例，直到最终成功发布。

Copyright © 2016 - 2023 DaoCloud



Copyright © 2016 - 2023 DaoCloud

4.2 快速创建流水线

本节将通过编译、构建、部署来创建流水线，通过详细的步骤操作描述，帮助您快速创建一条流水线。

前提条件

- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 `workspace edit` 角色。参考创建工作空间、用户和角色。
- 创建可以访问镜像仓库、集群的两个凭证，分别命名为：`registry`、`kubeconfig`。创建凭证的更多信息，请参考凭证管理。
- 准备一个 GitHub 仓库、DockerHub 仓库。

创建凭证

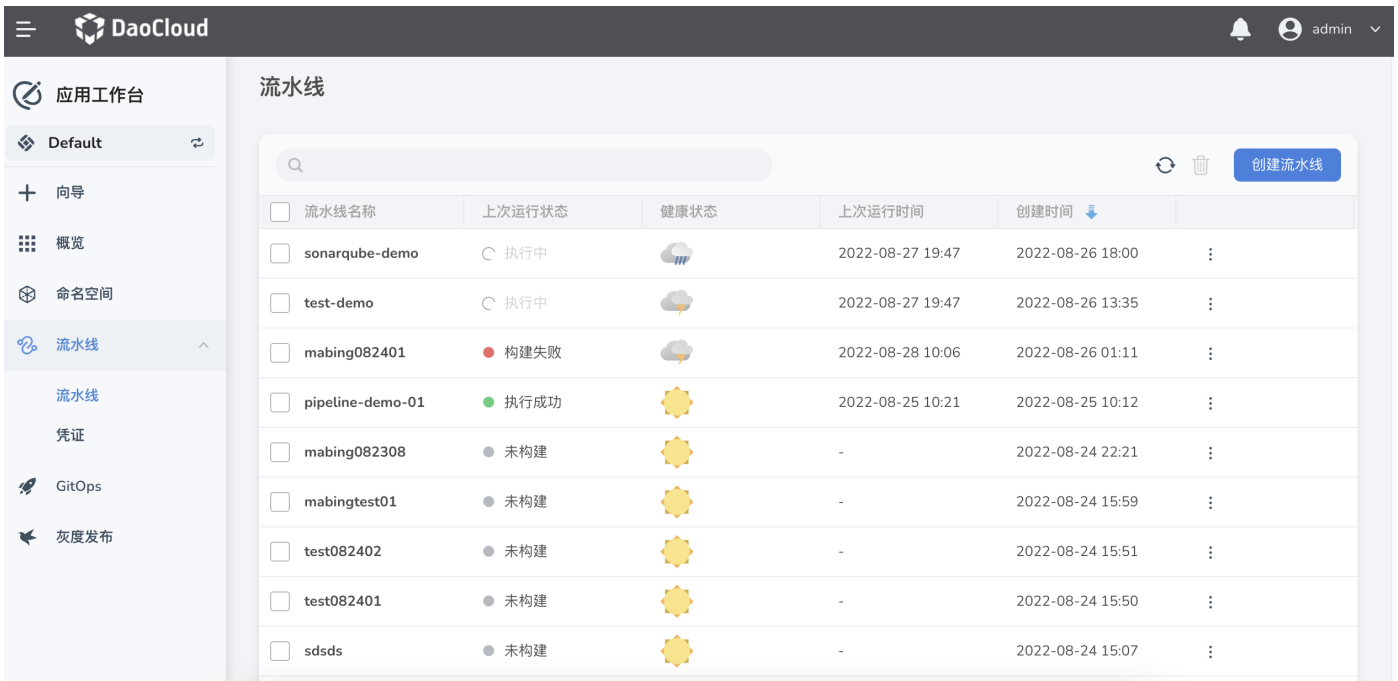
1. 在 凭证 页面创建两个凭证：

- docker-credential：用户名和密码，用于访问镜像仓库。
- demo-dev-kubeconfig：用于使用这个 kubeconfig 访问 Kubernetes 集群。

2. 创建完成后，可以在 凭证列表 页面看到凭证信息。

创建自定义流水线

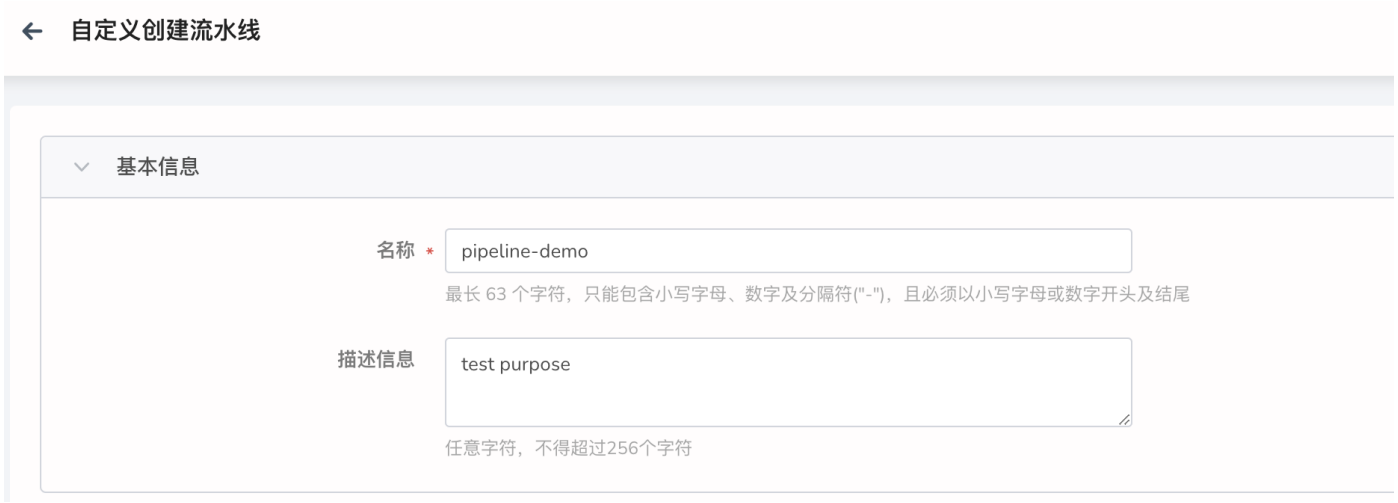
1. 在流水线列表页点击 创建流水线 。



2. 在弹出的对话框中，选择 自定义创建流水线 ，点击 确定 。

Copyright © 2016 - 2023 DaoCloud

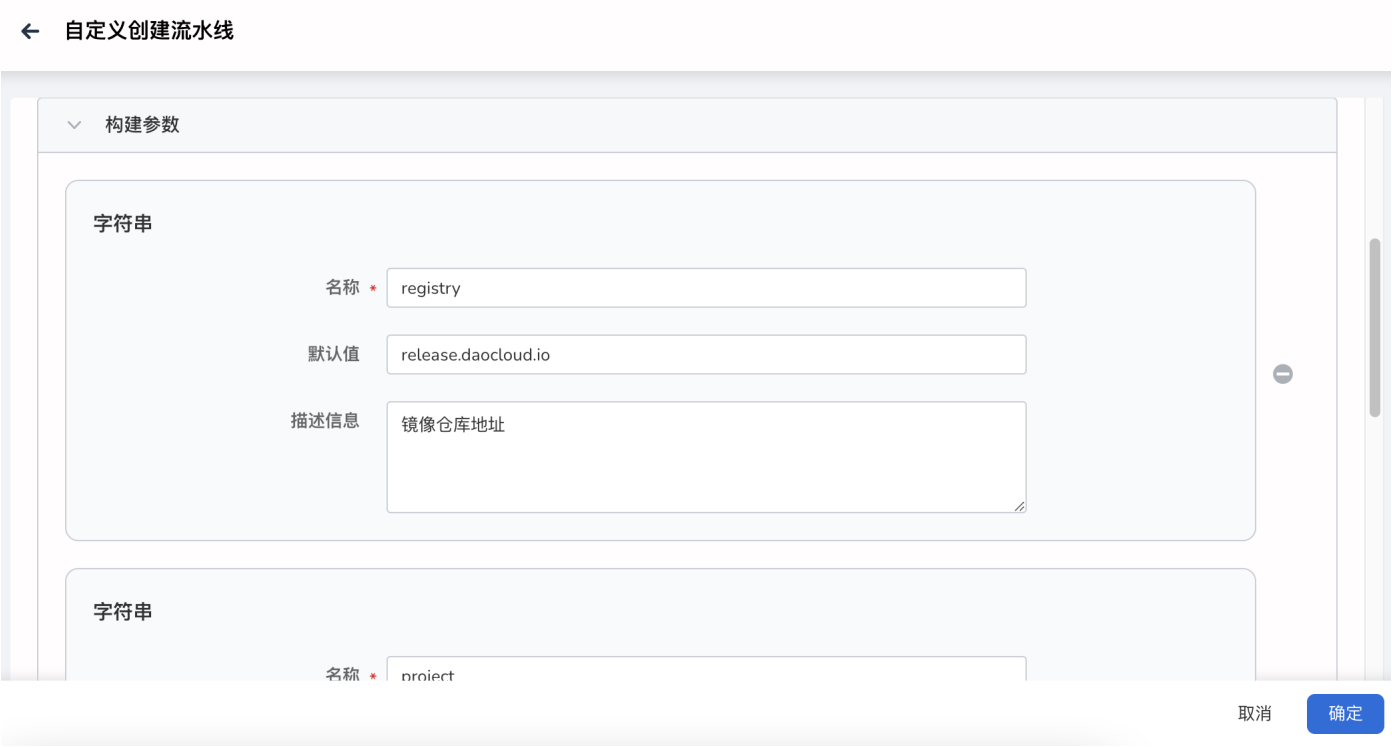
3. 进入 `自定义创建流水页面` ，输入流水线名称 `pipeline-demo` 。



4. 在 `构建参数` 中添加三个字符串参数，这些参数将用于镜像构建的命令中。

- registry：镜像仓库地址。示例值： `release.daocloud.io` 。
- project：镜像仓库中的项目名称。示例值： `demo` 。
- name：镜像的名称。示例值： `http-hello` 。

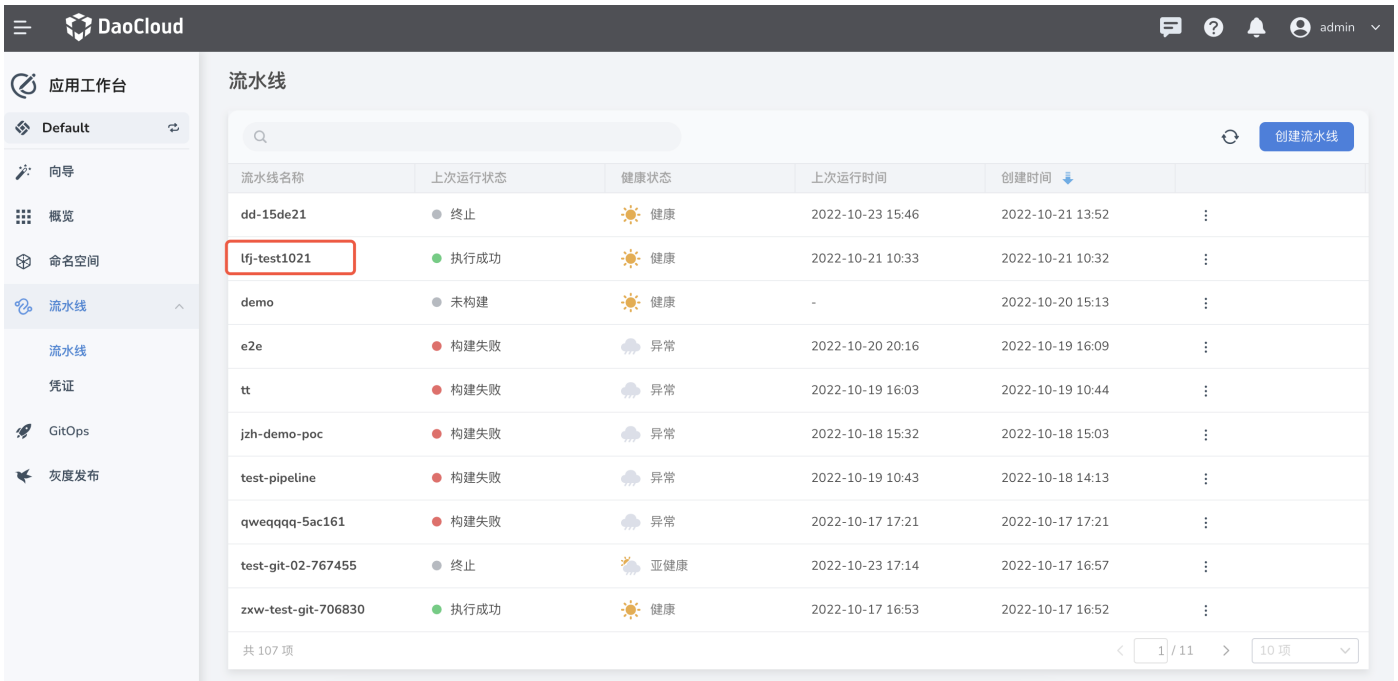
Copyright © 2016 - 2023 DaoCloud



5. 添加完成后，点击 确定 。

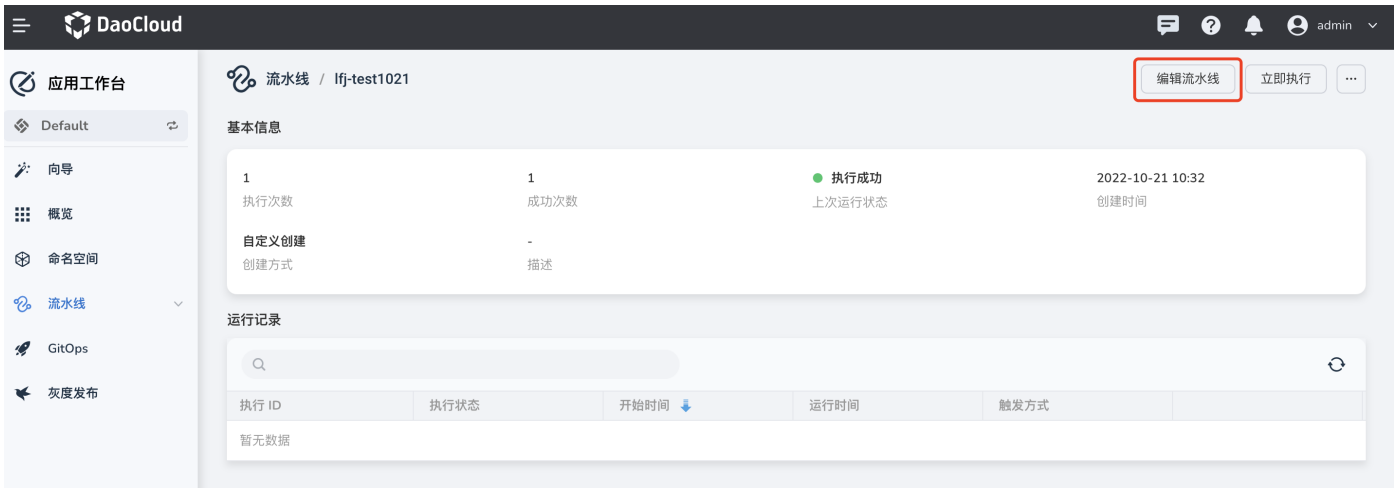
编辑流水线

1. 在流水线列表页面点击一个流水线的名称。

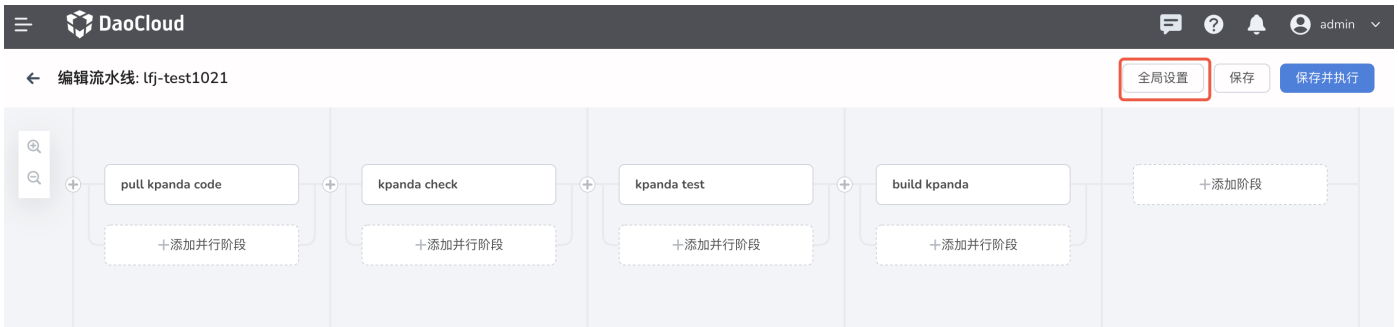


2. 在右上角点击 编辑流水线 ，

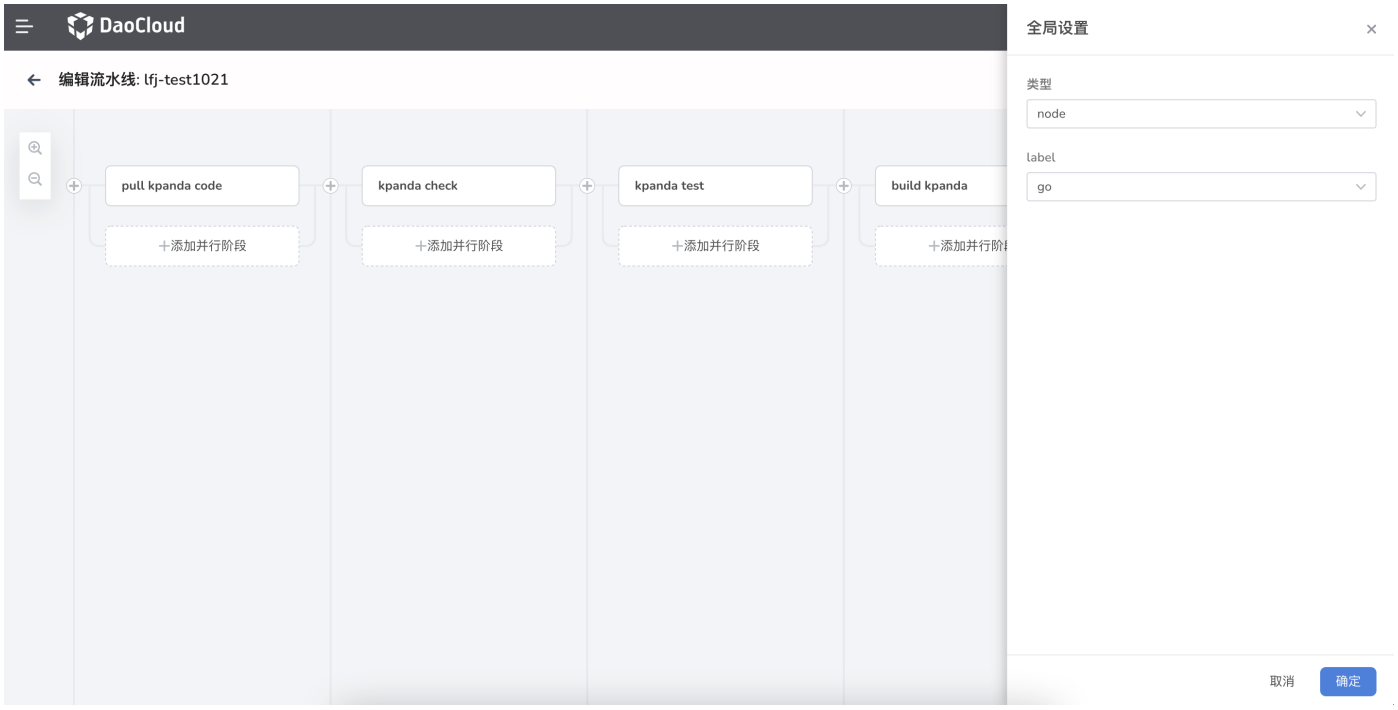
Copyright © 2016 - 2023 DaoCloud



3. 在右上角点击 全局设置 。



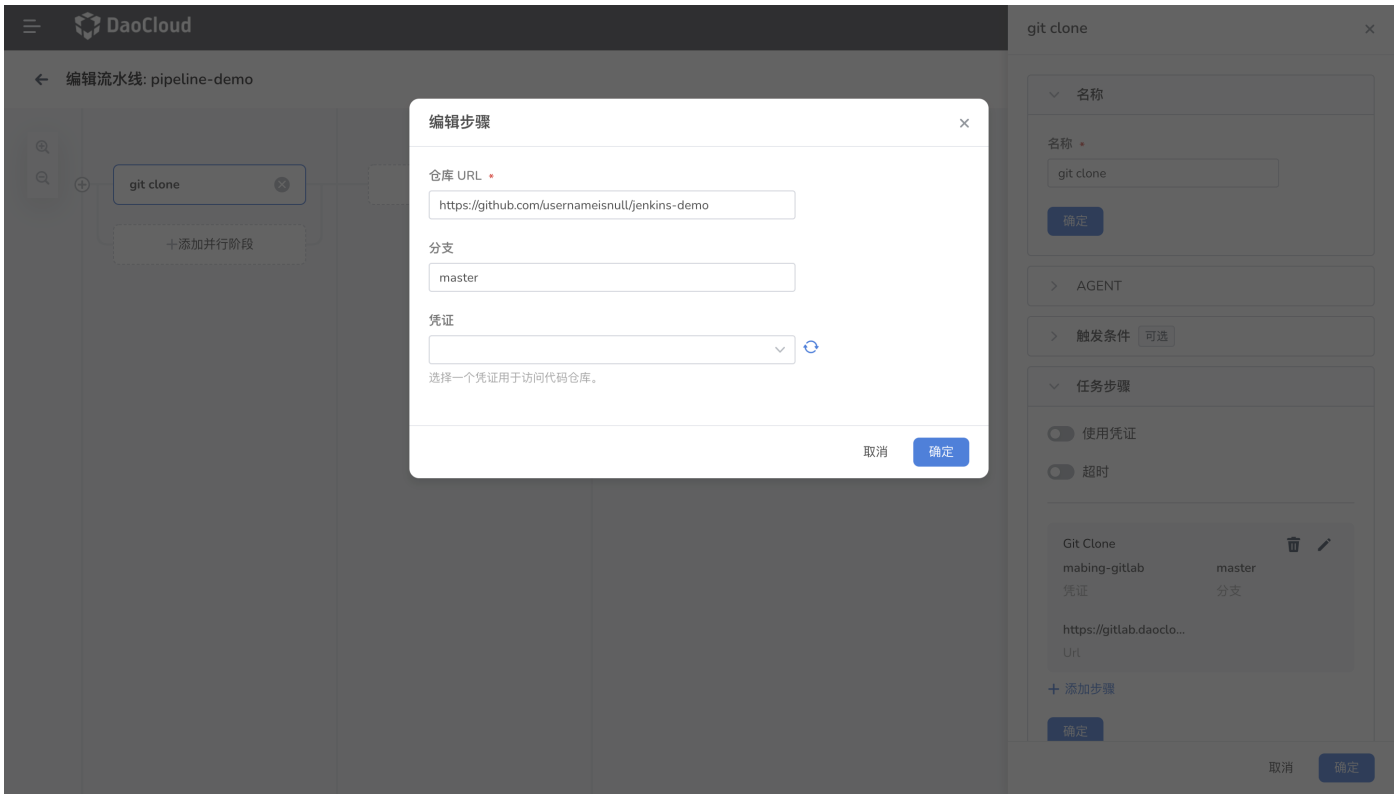
4. 类型设为 node，且 label 设为 go，点击 确定 。



Copyright © 2016 - 2023 DaoCloud

5. 添加阶段 - 拉取源代码。

- 点击画布中的 添加阶段 。在右侧的阶段设置中设置名称：git clone。
- 点击 添加步骤 ，在弹出对话框中步骤类型下选择 git clone，配置相关参数：
- 仓库 URL：输入 GitLab 仓库地址。
- 分支：不填写默认为 master 分支。
- 凭证：如果属于私有仓库，则需要提供一个凭证。

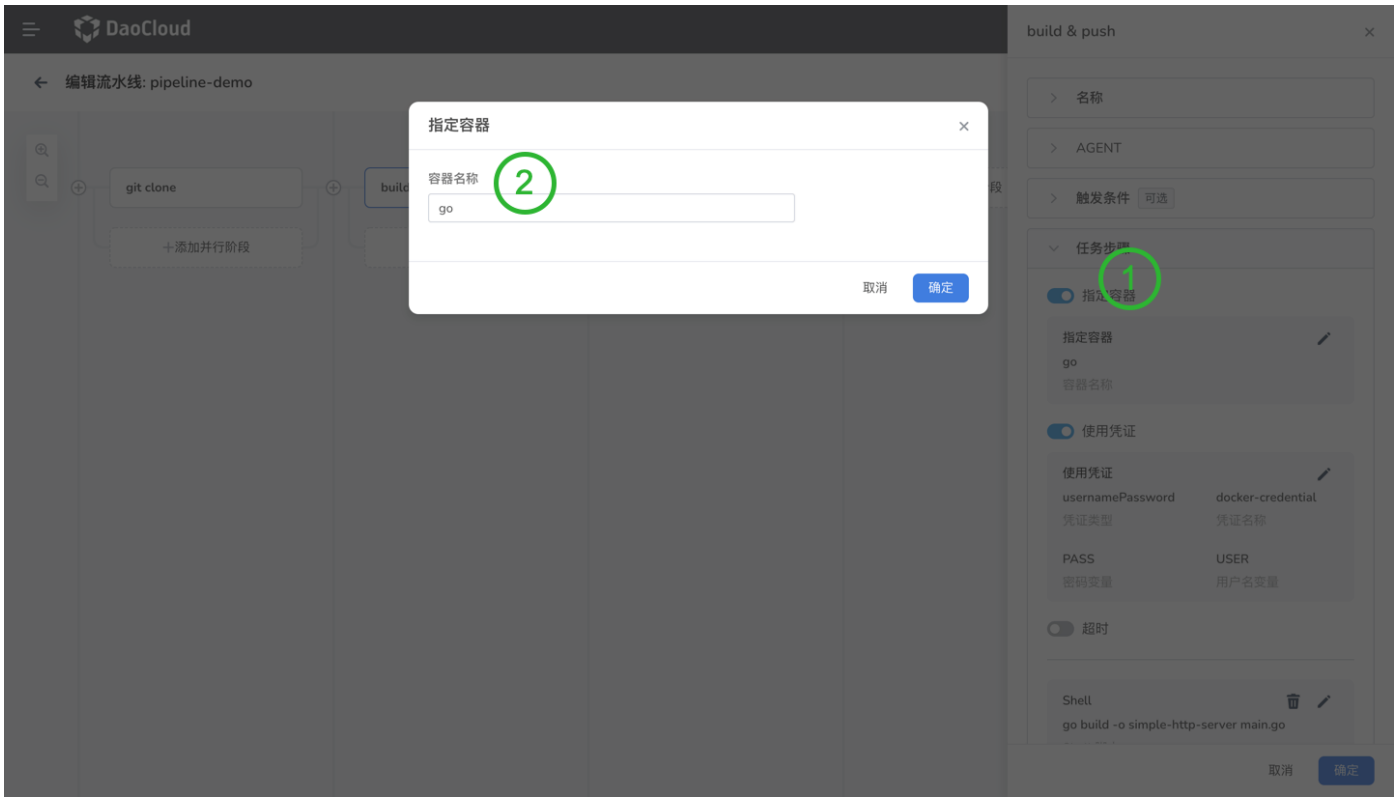


Copyright © 2016 - 2023 DaoCloud

6. 添加阶段 - 构建并推送镜像。

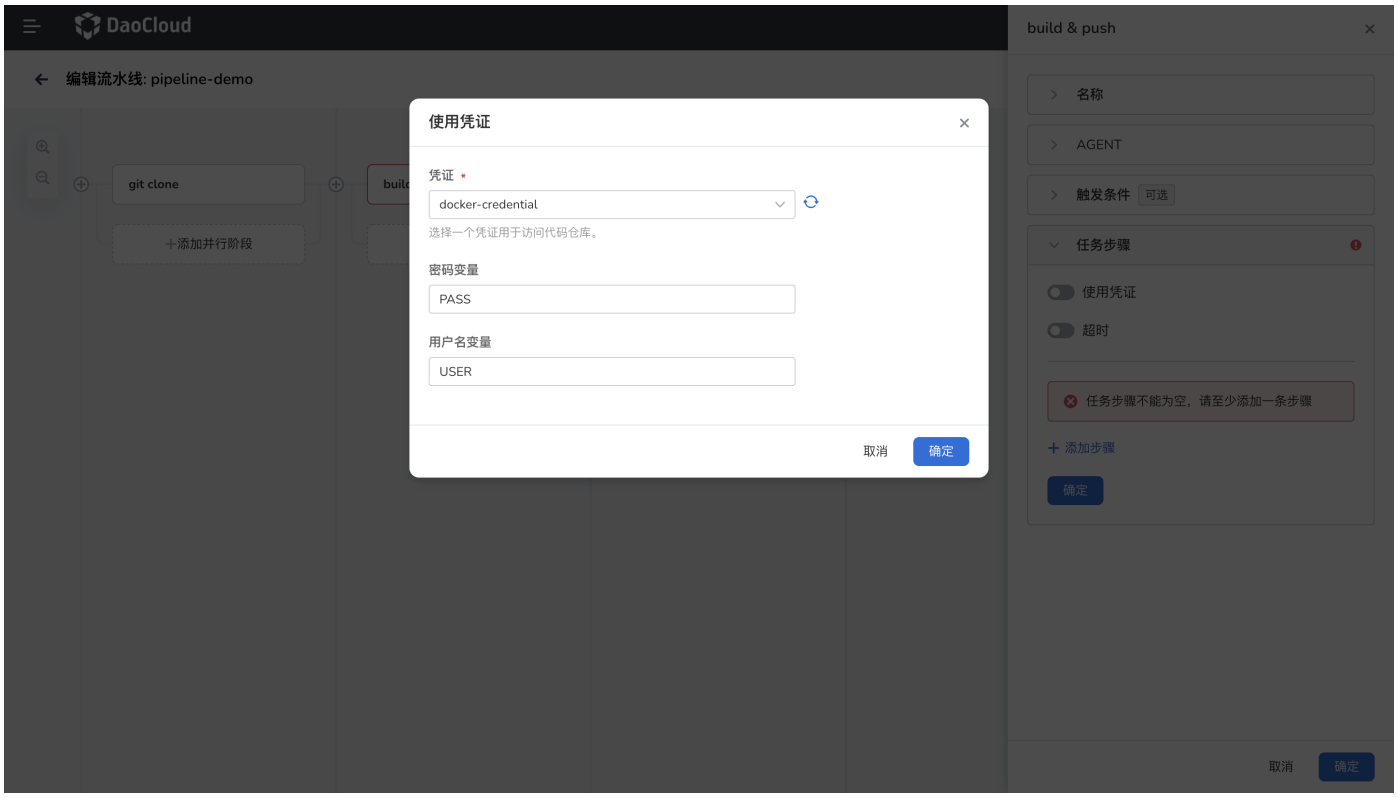
Copyright © 2016 - 2023 DaoCloud

- 点击画布中的 添加阶段 。在右侧的阶段设置中设置名称：build & push。
- 在步骤模块中选择开启 指定容器 ，在弹出的对话框中填写容器名称：go，然后点击 确定 。



- 在步骤模块中选择开启 使用凭证 ，在弹出的对话框中填写相关参数，然后点击 确定 。
- 凭证：选择创建的 Docker hub 凭证，用户访问镜像仓库。
- 密码变量：PASS
- 用户名变量：USER

Copyright © 2016 - 2023 DaoCloud



- 点击 `添加步骤` 进行代码构建，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

  ```
  go
  go build -o simple-http-server main.go
  ```

- 点击 `添加步骤` 以根据源码中的 Dockerfile 构建 Docker 镜像，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

  ```
  docker
  docker build -f Dockerfile . -t $registry/$project/$name:latest
  ```

- 点击 `添加步骤` 以登录镜像仓库，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

  ```
  docker
  docker login $registry -u $USER -p $PASS
  ```

- 点击 `添加步骤` 将镜像推送至镜像仓库中，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

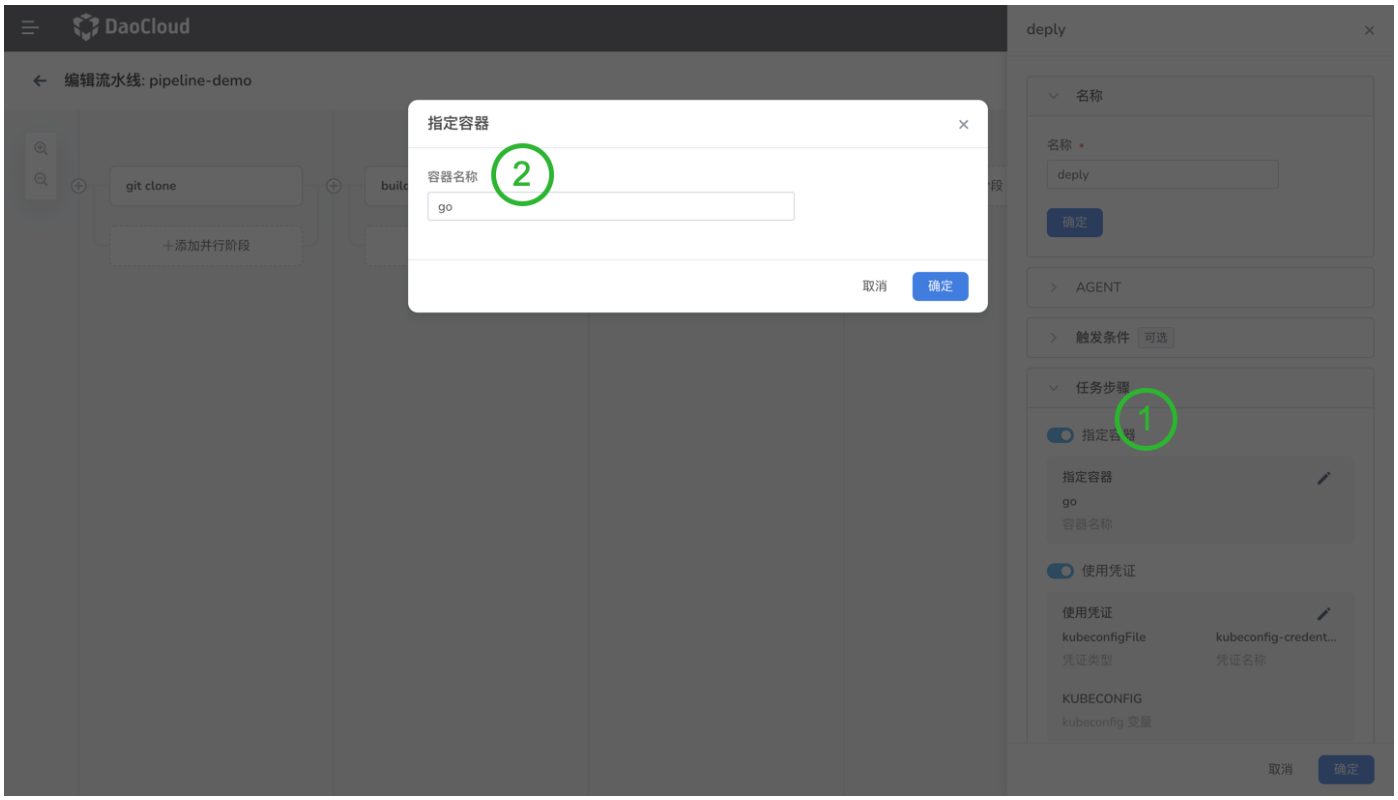
  ```
  docker
  docker push $registry/$project/$name:latest
  ```


Copyright © 2016 - 2023 DaoCloud

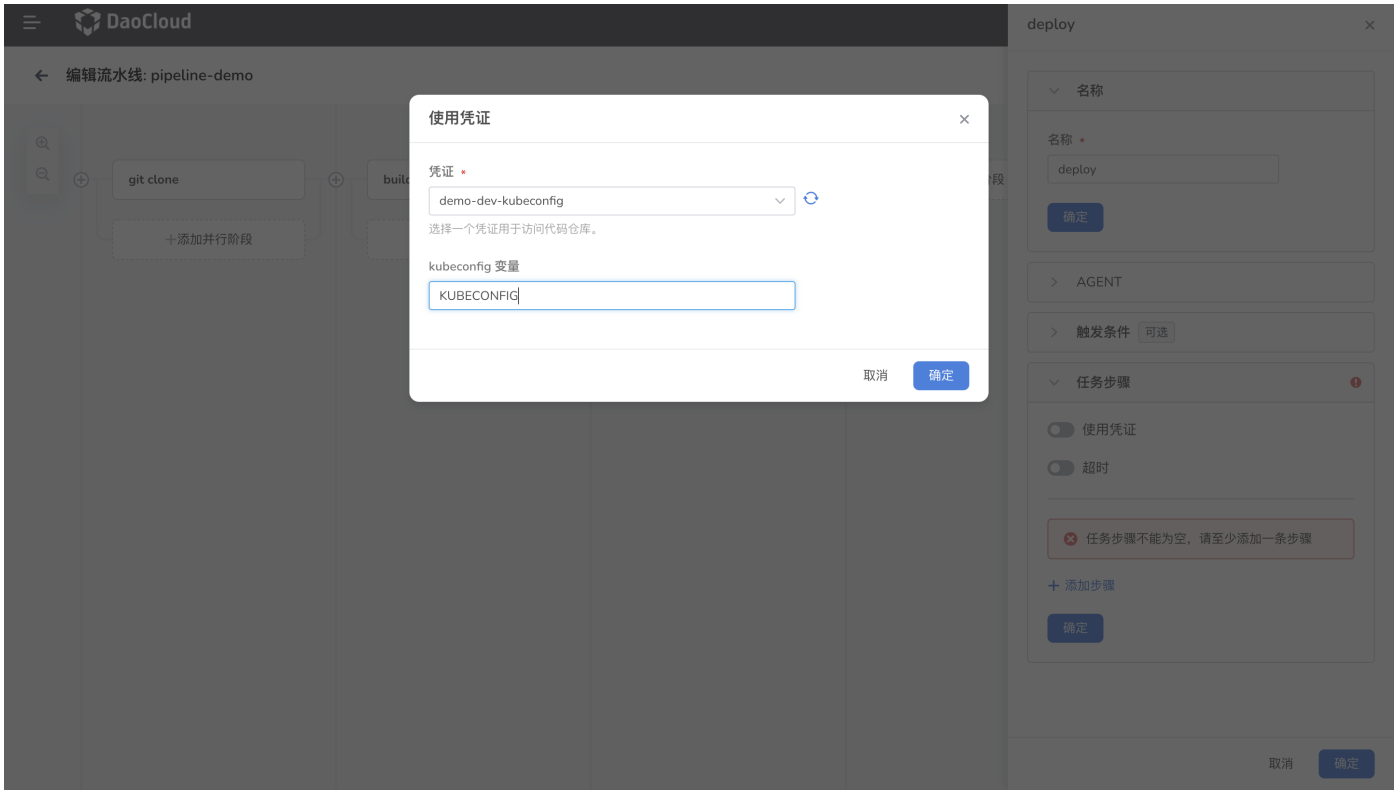
7. 添加阶段 - 部署至集群

Copyright © 2016 - 2023 DaoCloud

- 点击画布中的 添加阶段 。在右侧的阶段设置中设置名称：deploy。
- 在步骤模块中选择开启 指定容器 ，在弹出的对话框中填写容器名称：go，然后点击 确定 。



- 在步骤模块中选择开启 使用凭证 ，在弹出的对话框中填写相关参数，然后点击 确定 。
- 凭证：选择 kubeconfig 类型的凭证。
- kubeconfig 变量：如果您使用的是 kubectl apply 的部署方式，变量值必须为 KUBECONFIG。



Copyright © 2016 - 2023 DaoCloud

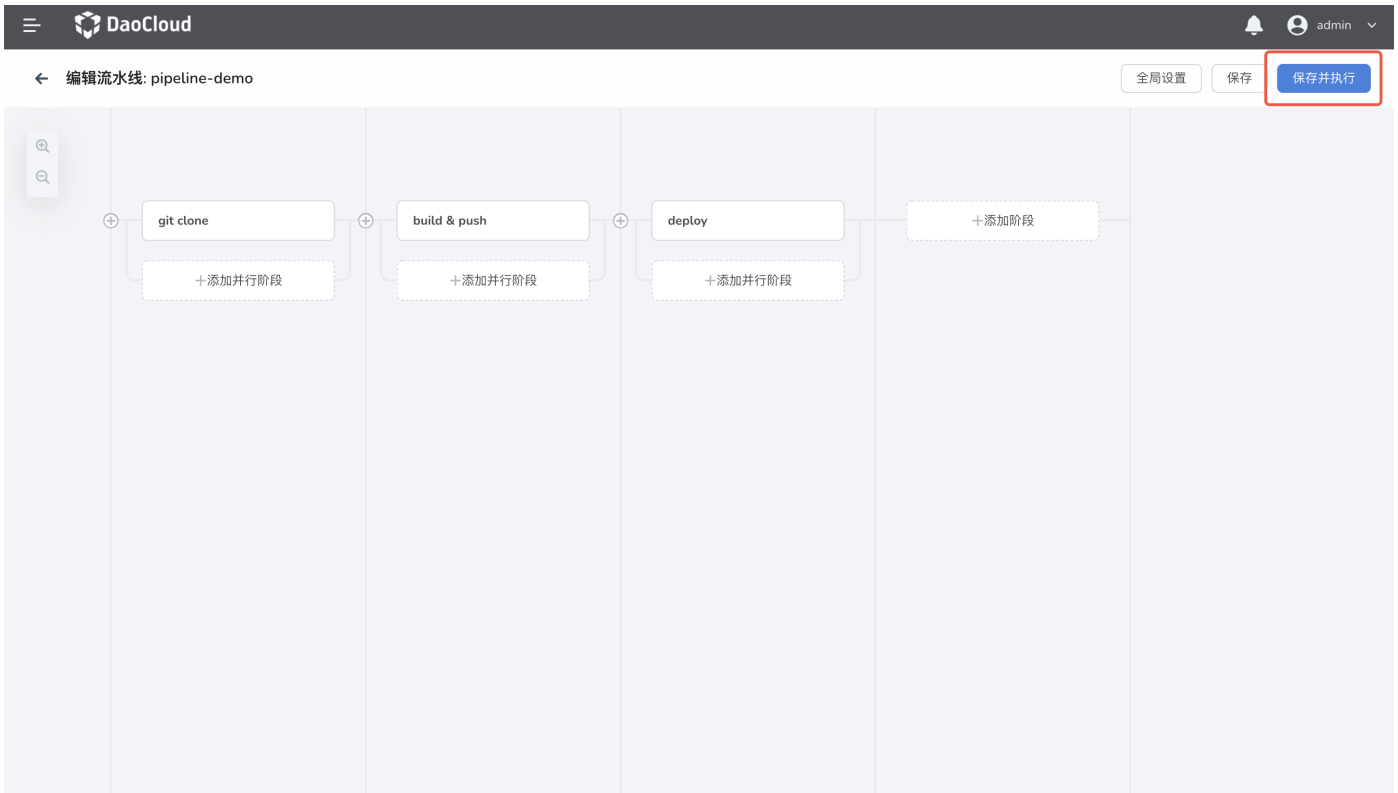
- 点击 `添加步骤` 以进行集群部署操作，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

  `shell`

  `kubectl apply -f deploy.yaml`

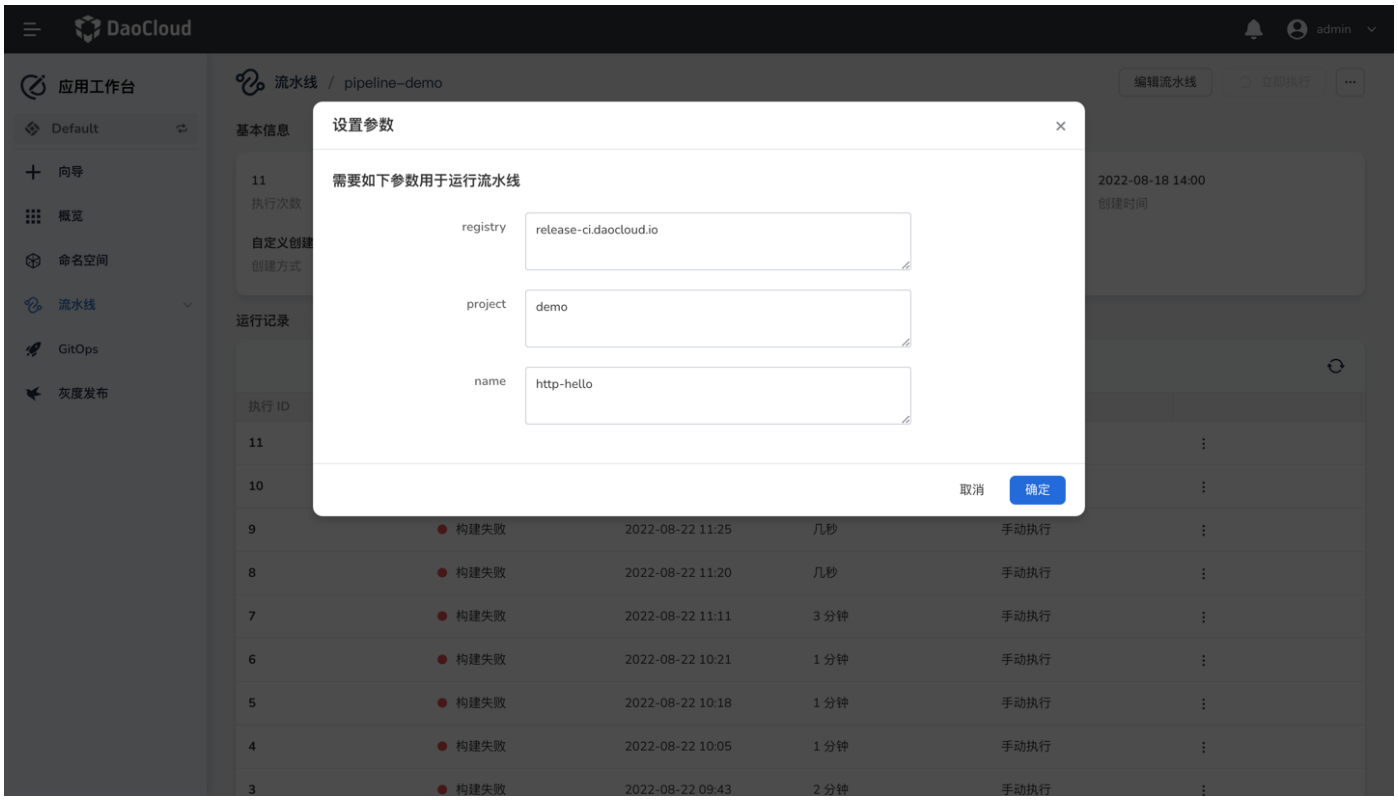
保存并执行流水线流水线

1. 完成上一步后点击 `保存并执行` 。



2. 在弹出的对话框中输入步骤二中的示例参数。点击 `确定` 即可成功运行该流水线。

Copyright © 2016 - 2023 DaoCloud



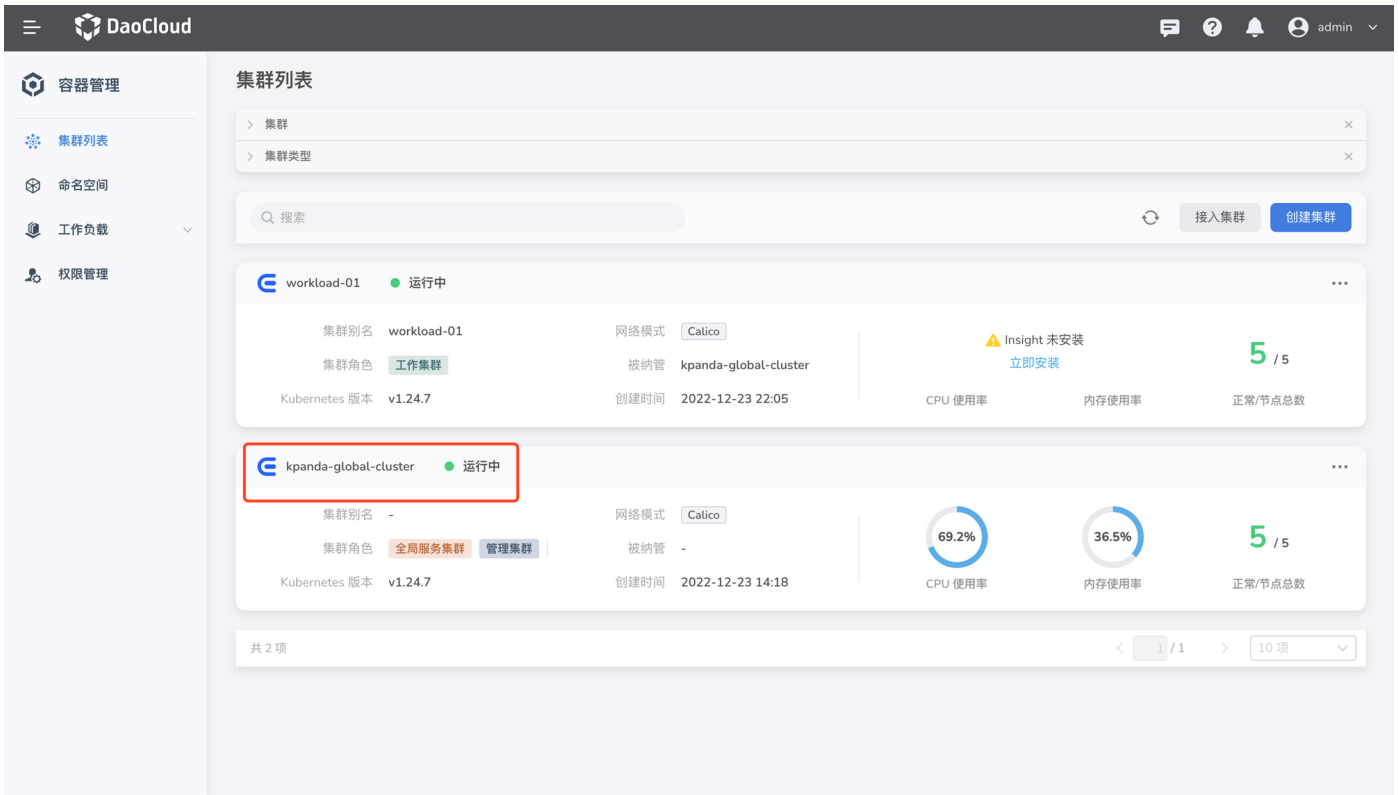
Copyright © 2016 - 2023 DaoCloud

4.3 在指定的节点上运行流水线

本文介绍如何在应用工作台中客户的流水线任务在指定的节点上运行。

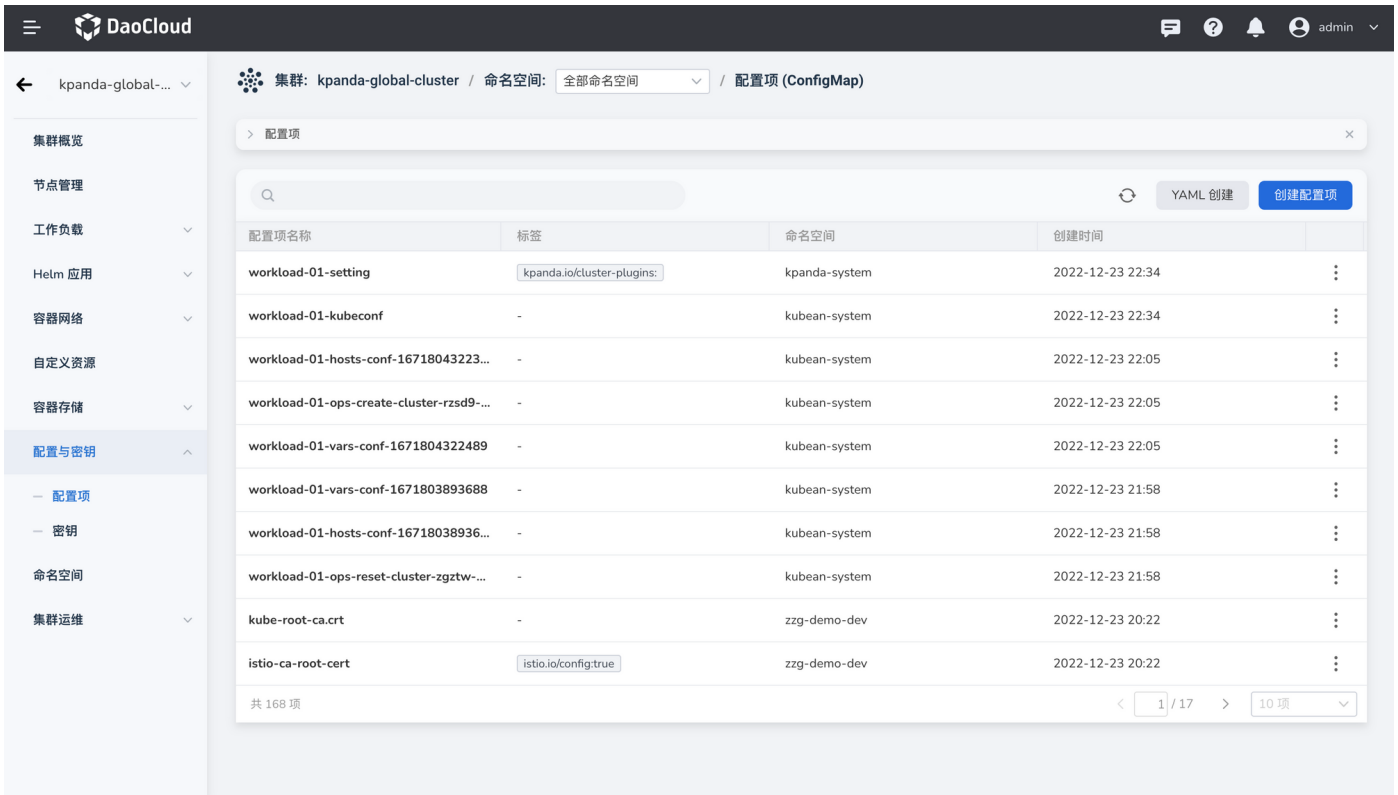
修改配置文件 jenkins-casc-config

1. 前往 容器管理 模块，进入目标集群的详情页面，例如 `kpanda-global-cluster` 集群。

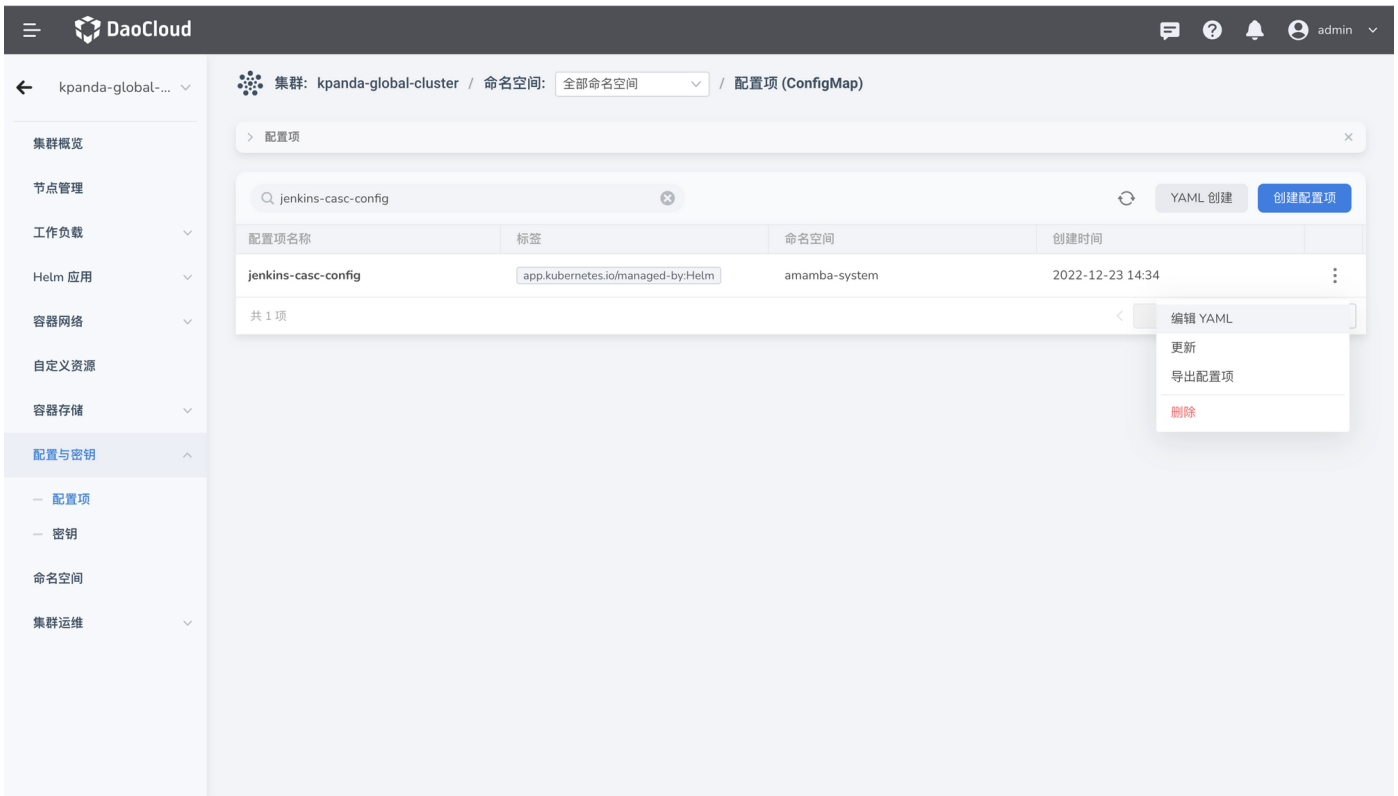


2. 在左侧导航点击 配置与密钥 -> 配置项 。

Copyright © 2016 - 2023 DaoCloud

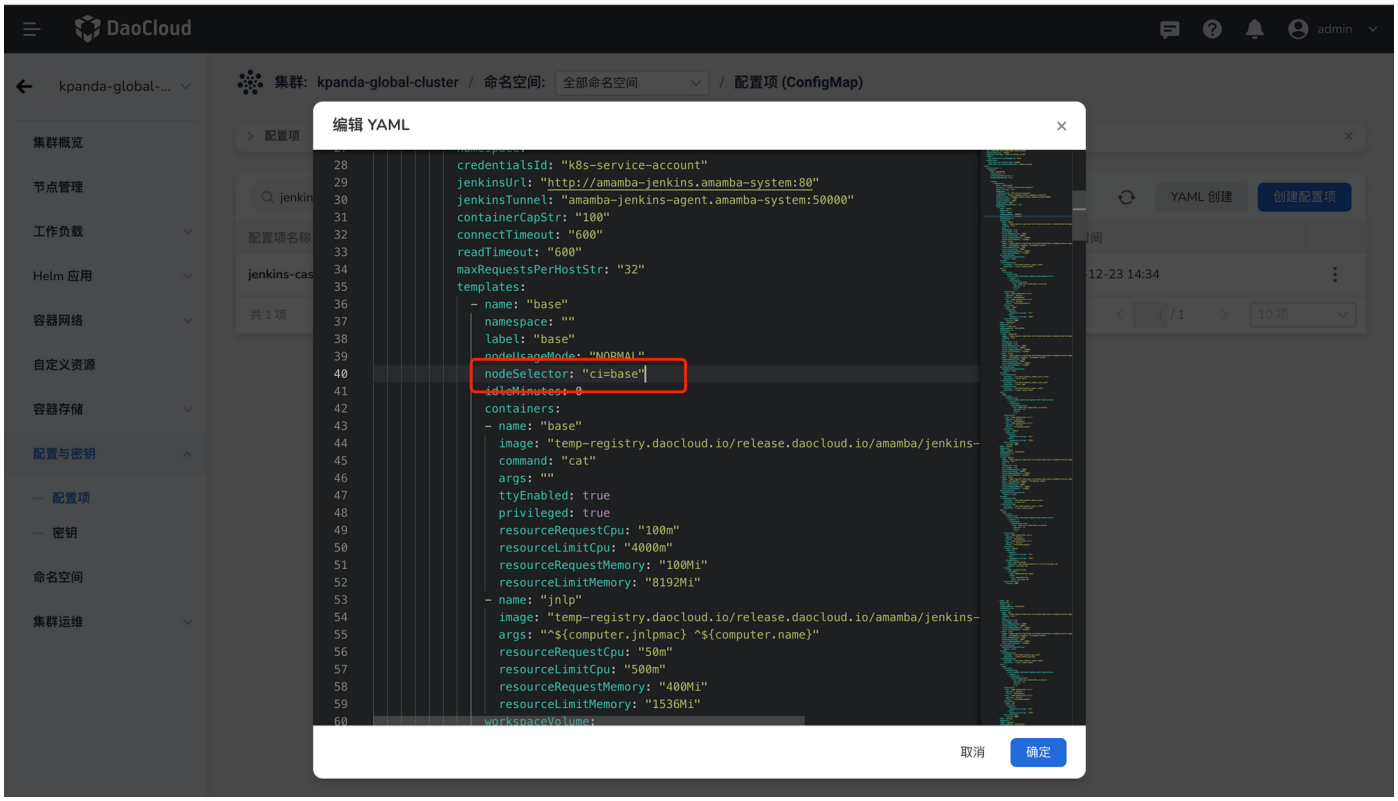


3. 搜索 `jenkins-casc-config`，在列表选择 编辑 `YAML` 。



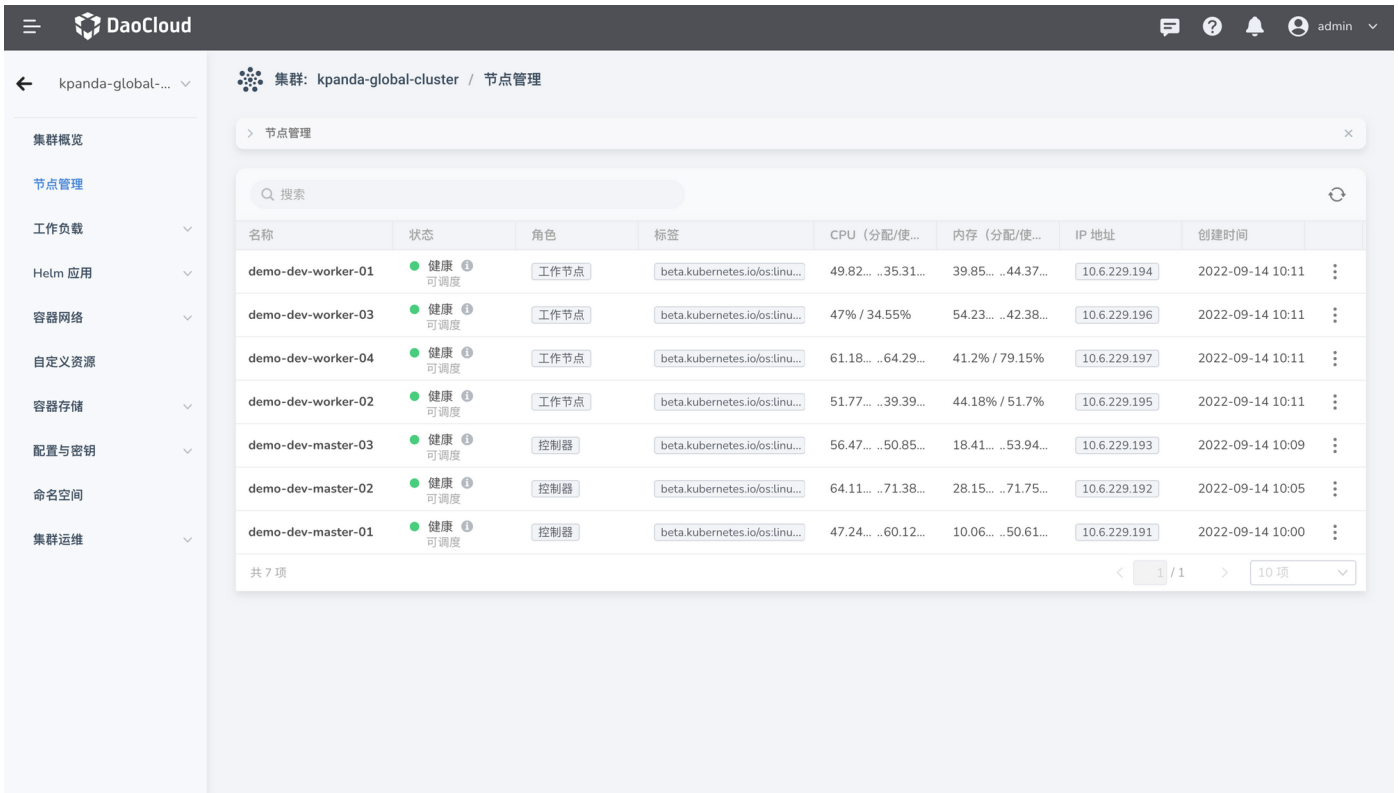
4. 在 YAML 配置项 `jenkins.yaml` 中的 `jenkins.cloud.kubernetes.templates` 位置下为某个具体的 Agent 添加 `nodeSelector: "ci=base"` ，点击 `确定` 保存更改。

Copyright © 2016 - 2023 DaoCloud



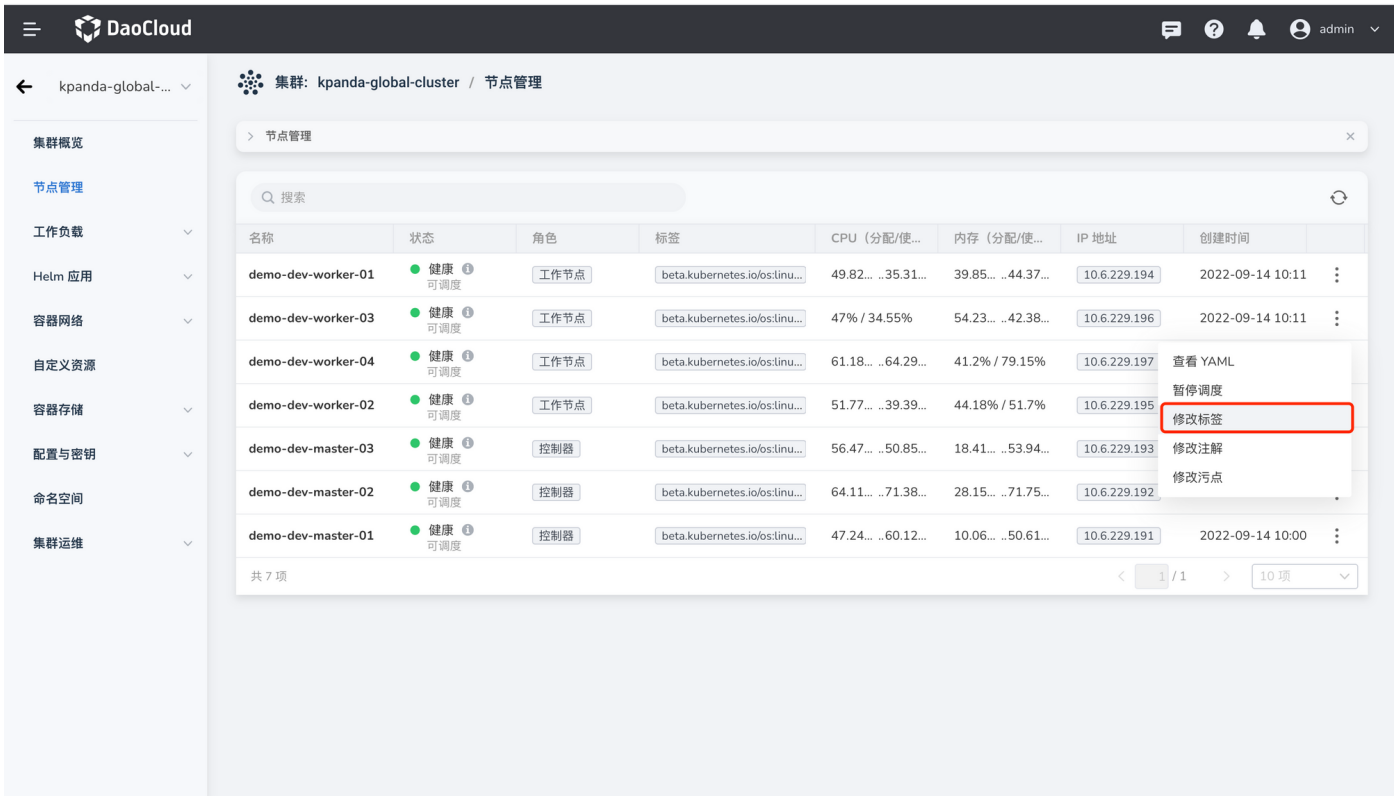
选择指定的节点添加标签

1. 进入 容器管理 模块，在 `kpanda-global-cluster` 集群的详情页面，在左侧导航点击 节点管理 。

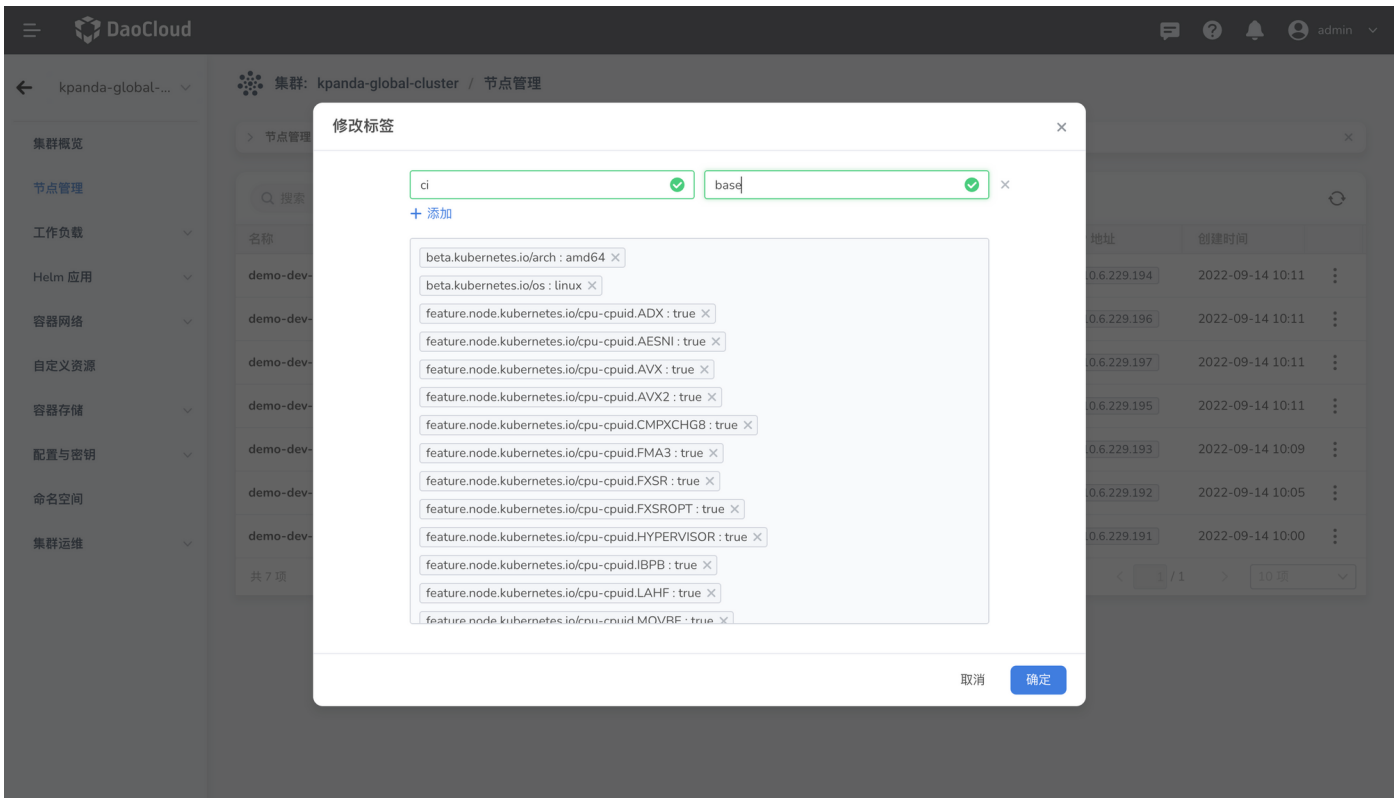


2. 选择目标的工作节点（例如 demo-dev-worker-03），点击 修改标签 。

Copyright © 2016 - 2023 DaoCloud



3. 添加 `ci=base` 标签，点击 确定 保存更改。



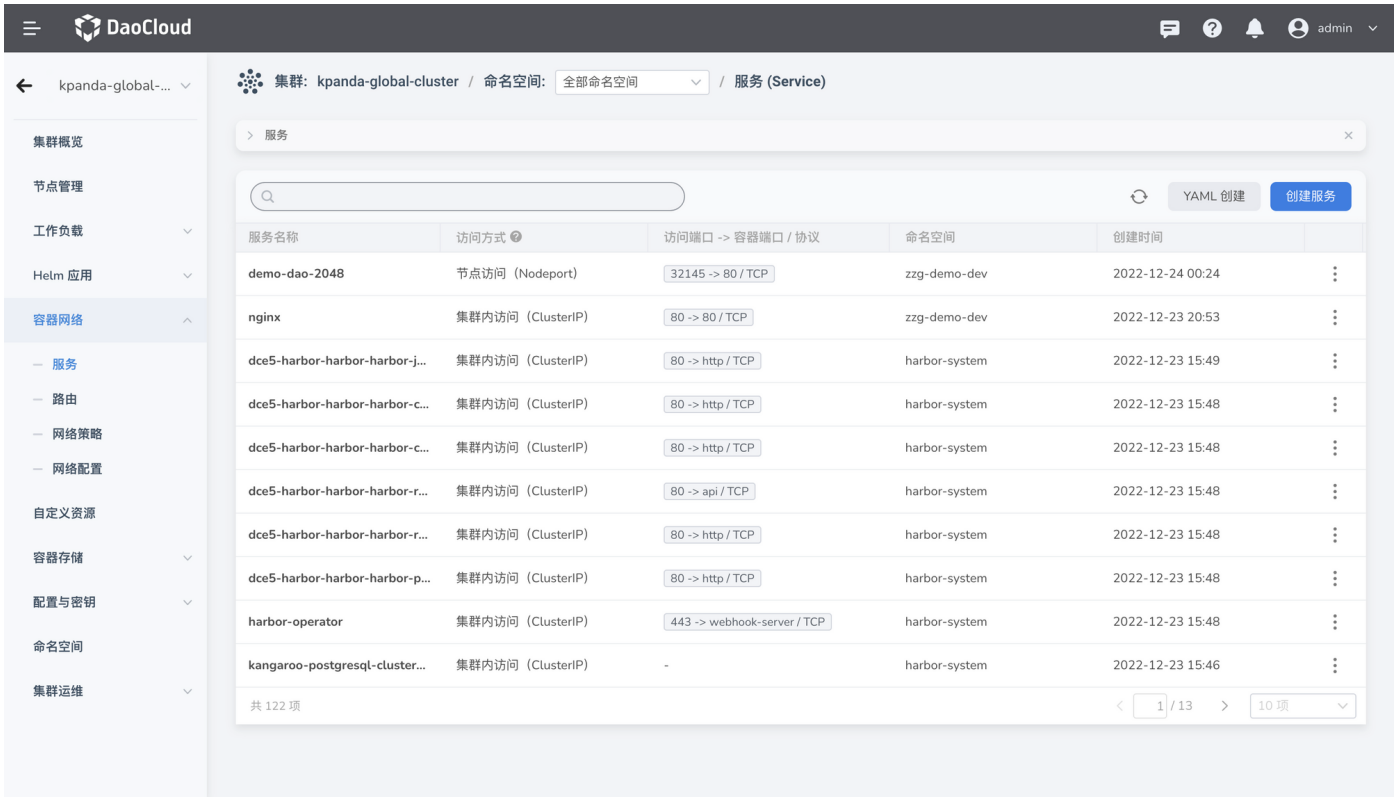
Copyright © 2016 - 2023 DaoCloud

访问 Jenkins Dashbord，重新加载配置

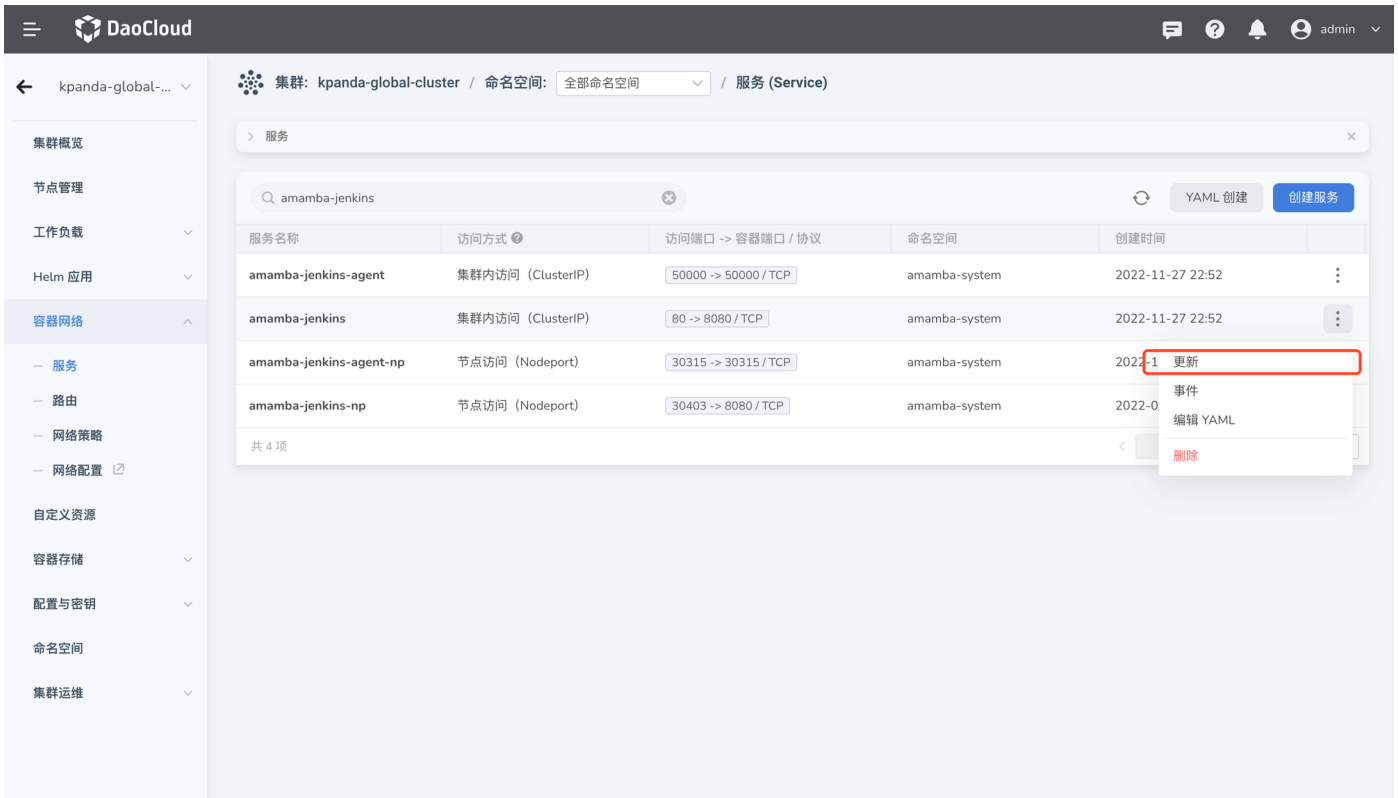
首先需要讲通过 NodePort 方式暴露 Jenkins Dashbord 的访问地址（其他暴露方式根据业务实际情况进行暴露）。

Copyright © 2016 - 2023 DaoCloud

1. 进入 容器管理 模块，在 `kpanda-global-cluster` 集群页面，在左侧导航栏点击 容器网络 -> 服务 。

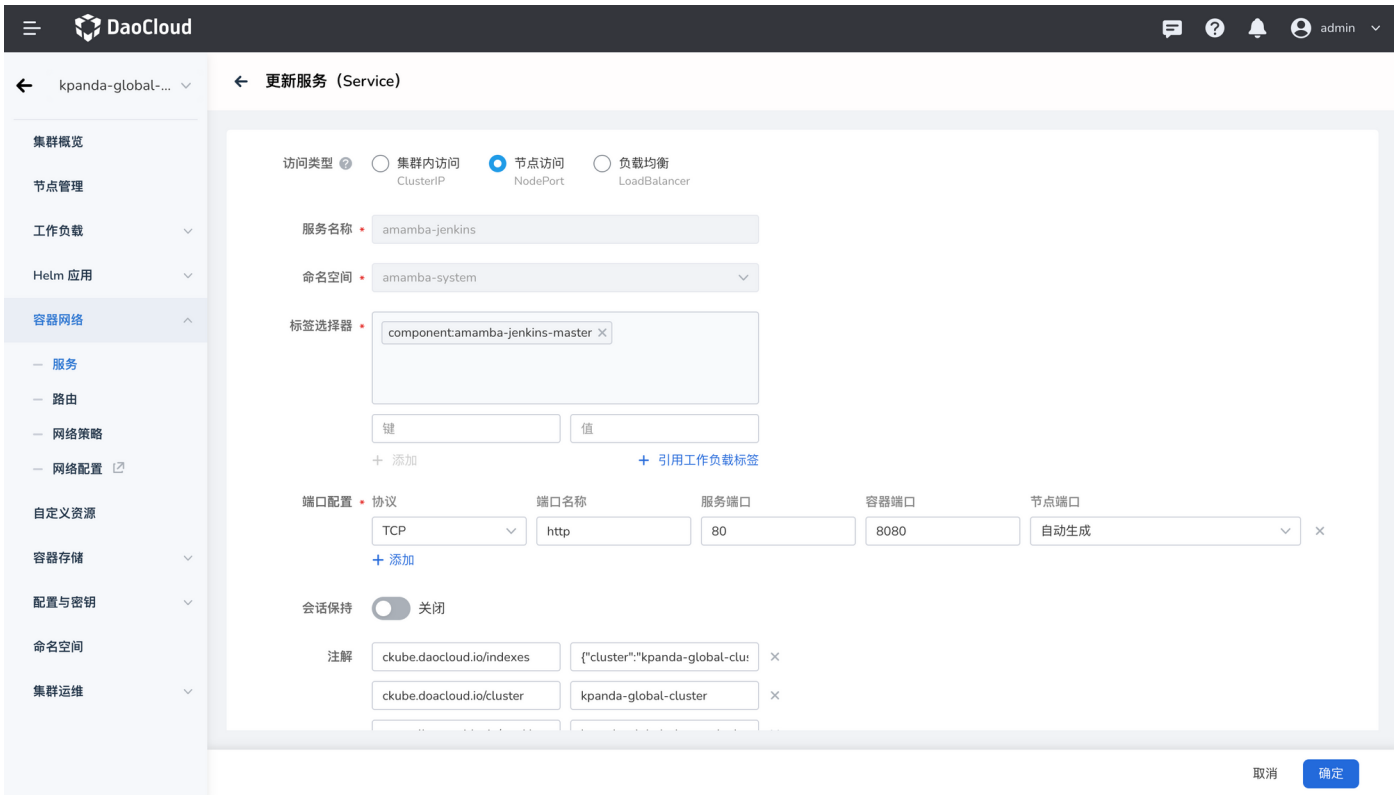


2. 搜索 `amamba-jenkins`，在列表选择 更新 。

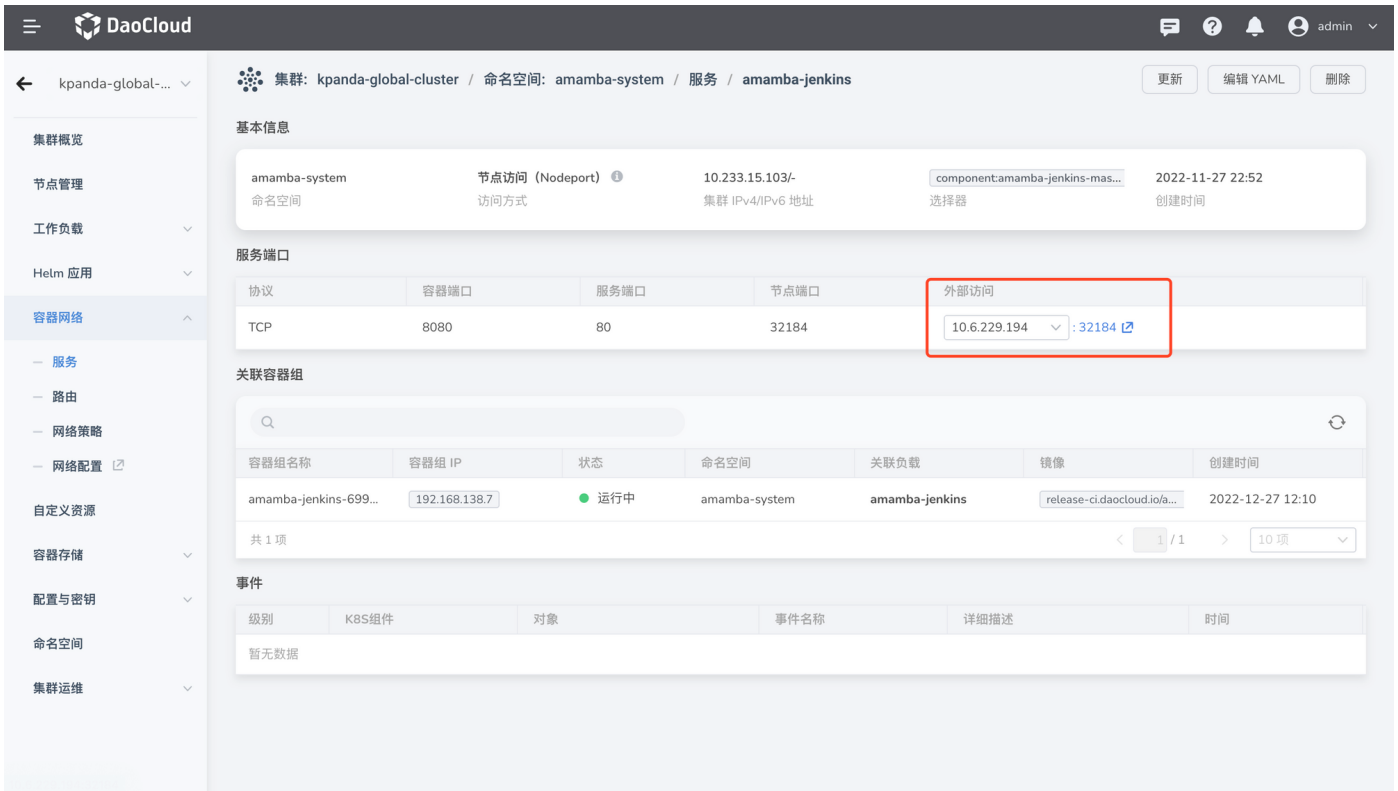


3. 将访问类型更改为 NodePort，节点端口选择自动生成即可。

Copyright © 2016 - 2023 DaoCloud

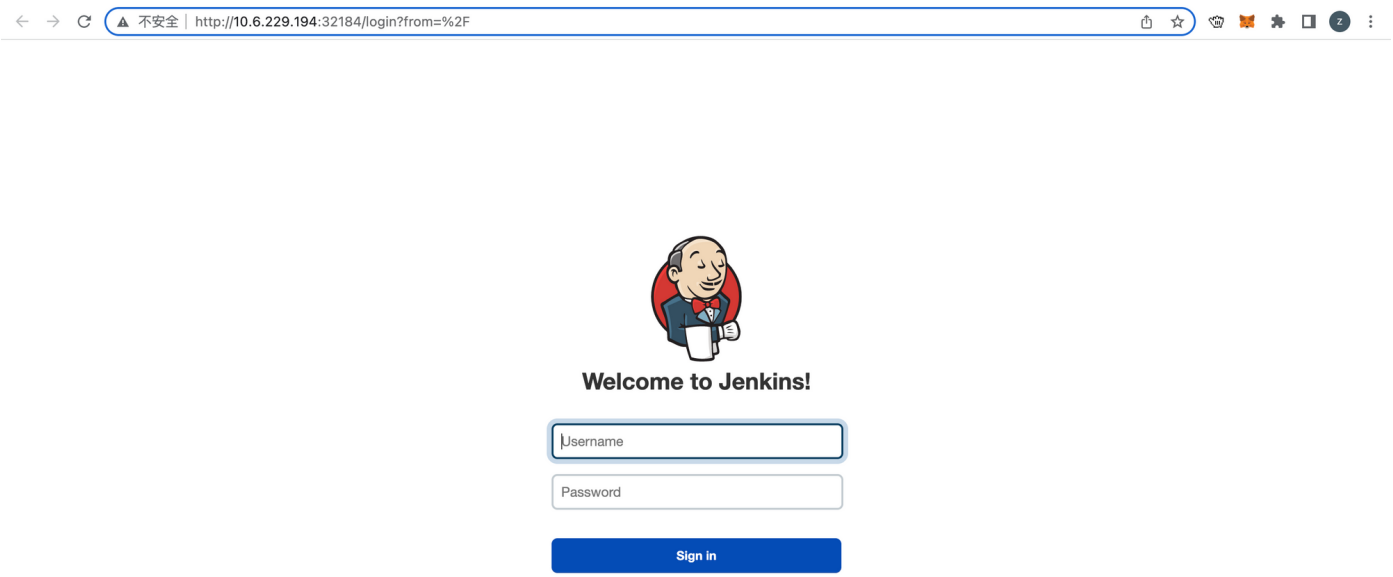


4. 点击确定，然后返回到详情页面点击链接访问 Jenkins Dashboard。

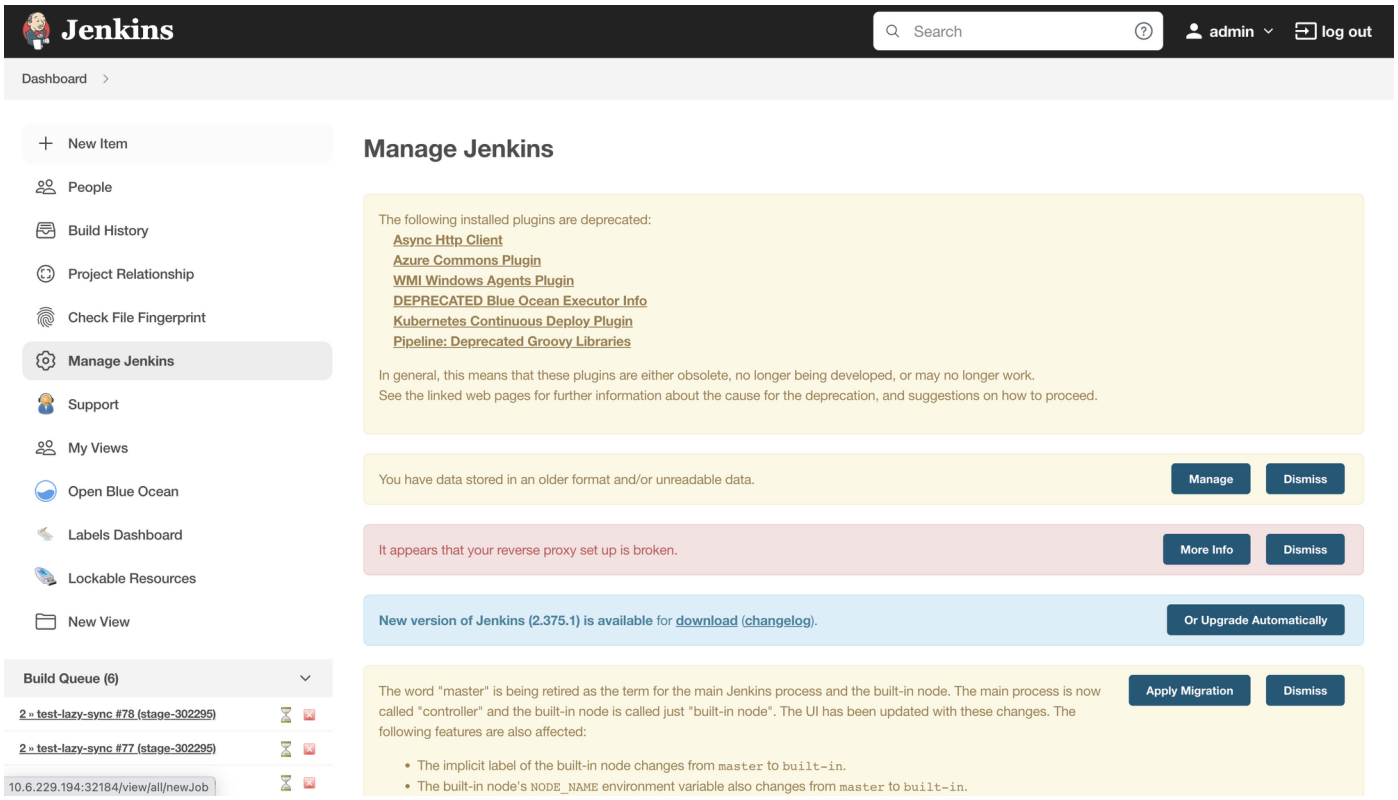


5. 输入账号/密码（默认为 `admin/Admin01` ），进入到 Jenkins Dashboard 页面。

Copyright © 2016 - 2023 DaoCloud

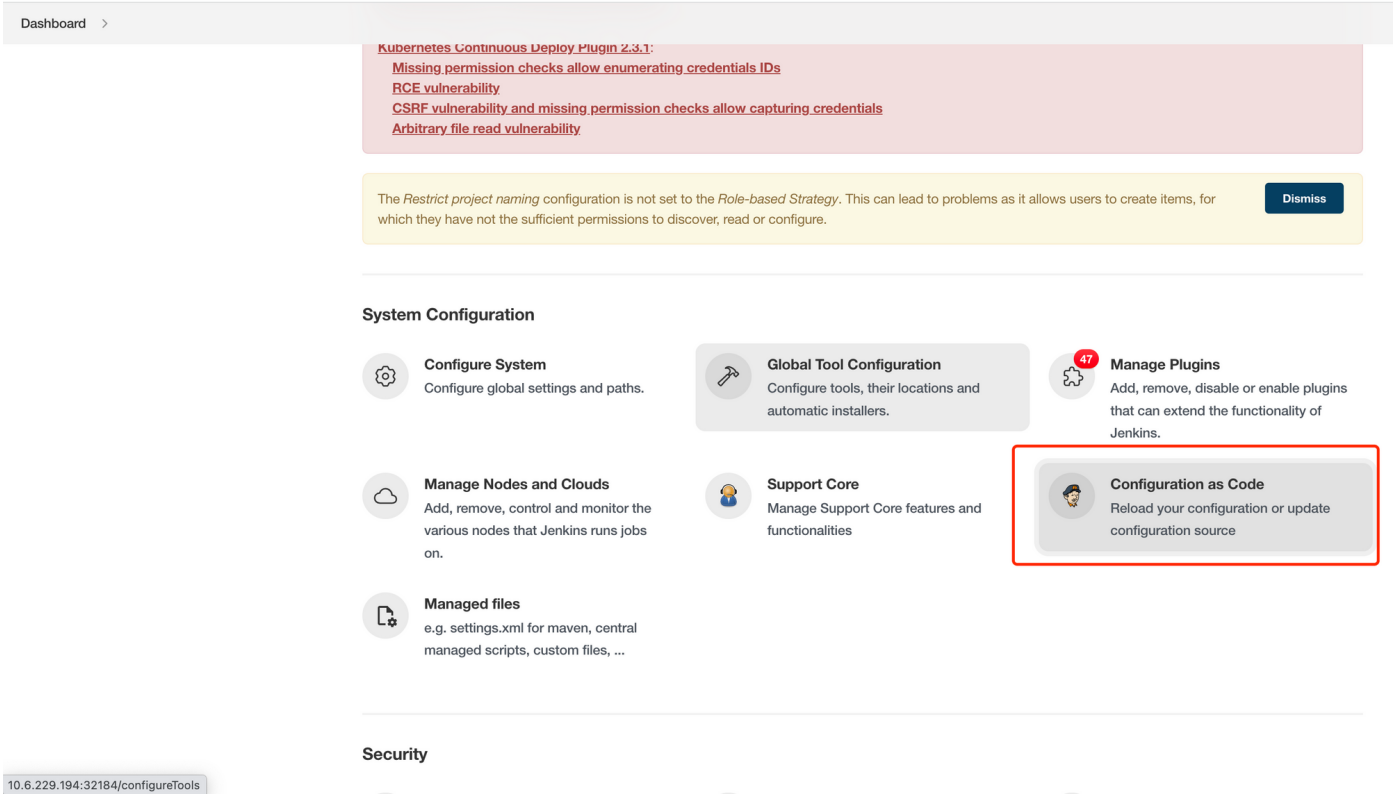


6. 在左侧导航栏选择 `Manage Jenkins` 。

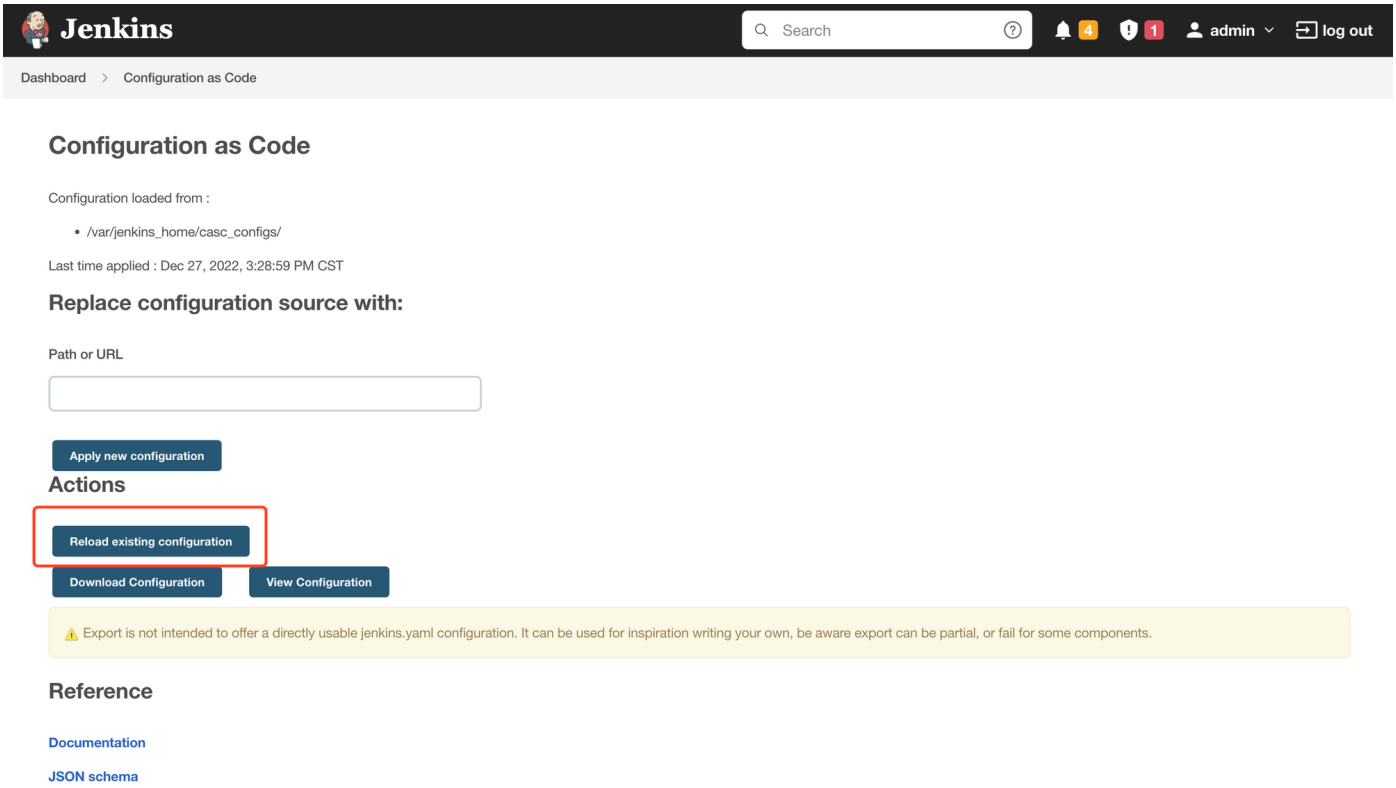


7. 点击 `Configuration as Code` 。

Copyright © 2016 - 2023 DaoCloud



8. 在 `Configuration as Code` 点击 `Reload existing configuration` 。如果点击后在当前页面没有任何提示，说明配置加载生效。



运行流水线，检查是否在指定的节点上

1. 在 `应用工作台` 创建一个流水线任务，并编辑 `Jenkinsfile` 如下：

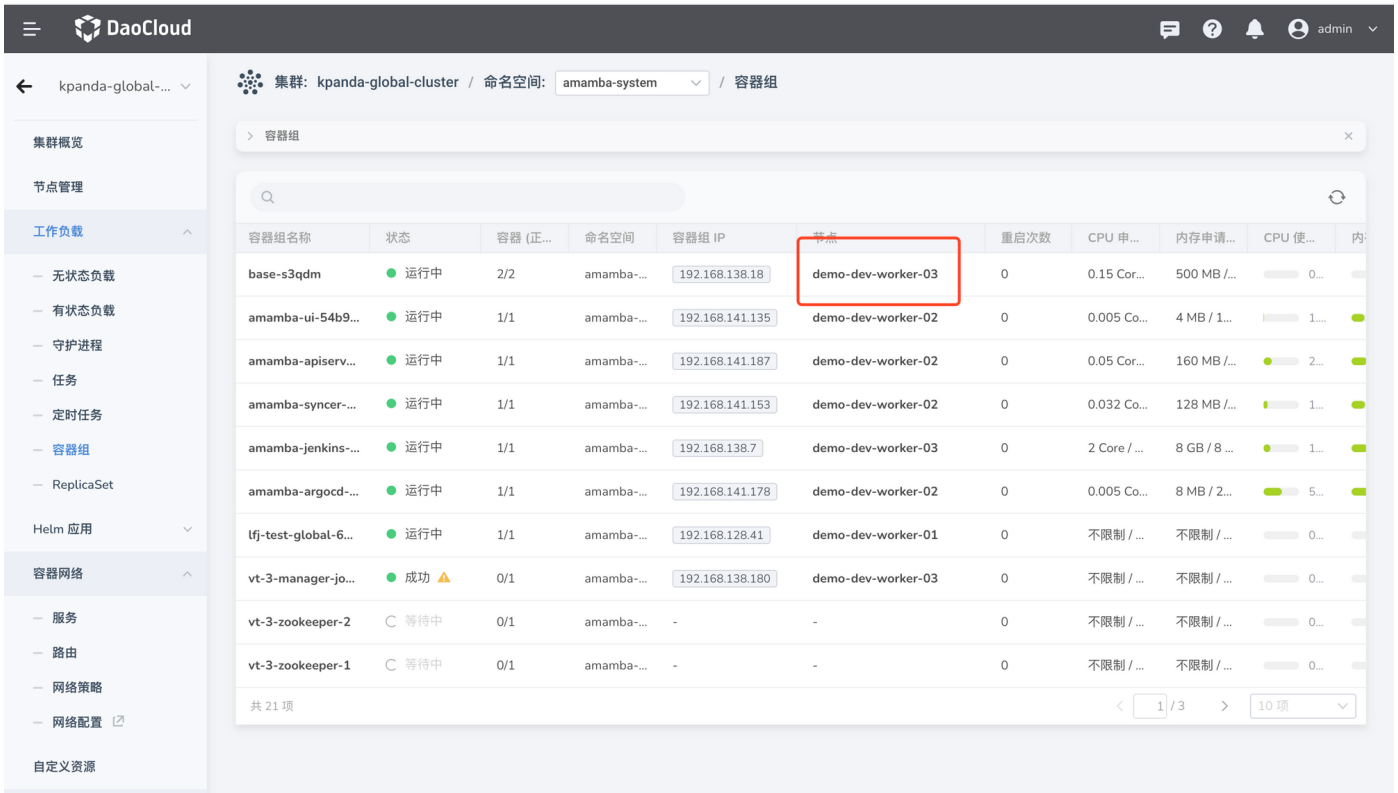
Copyright © 2016 - 2023 DaoCloud

```bash
pipeline { agent { node { label 'base' }
} stages { stage('Hello') { agent none steps { container('base') { echo 'Hello World' sh 'sleep 300' }
  }
}
} } ```
```

!!! note

```
需要注意 agent 部分需要选择 label 为 base。因为在配置文件中只为 base 设置了指定节点，如果需要为其他的 agent 设置。重复上述操作即可。
```

2. 点击 立即执行 该流水线，前往 容器管理 查看执行该任务的 Pod 运行的节点。



3. 可以看到执行该流水任务的 Pod 运行在了预期的 `demo-dev-worker-03` 节点上。

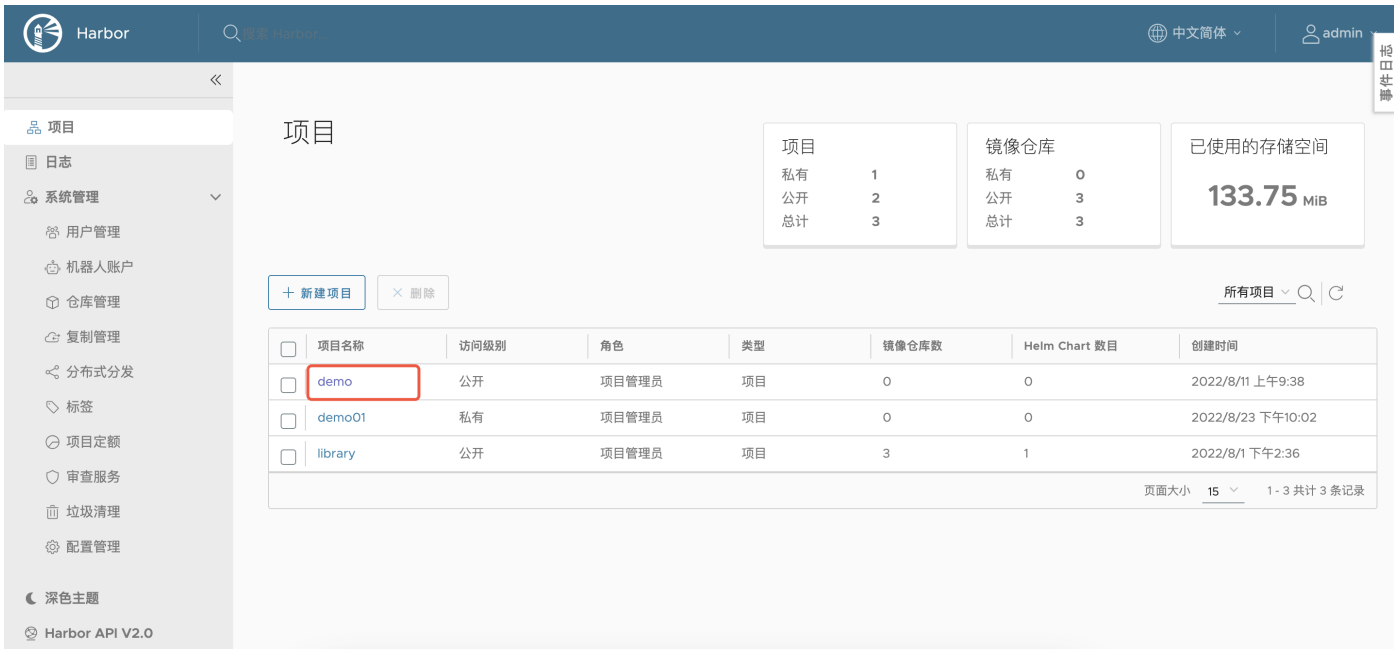
Copyright © 2016 - 2023 DaoCloud

4.4 集成 Harbor 实现镜像安全扫描

本文将介绍如何在流水线中集成 Harbor 并实现镜像安全扫描。

在 Harbor 中开启自动扫描镜像

1. 登录 Harbor，点击一个具体的项目。

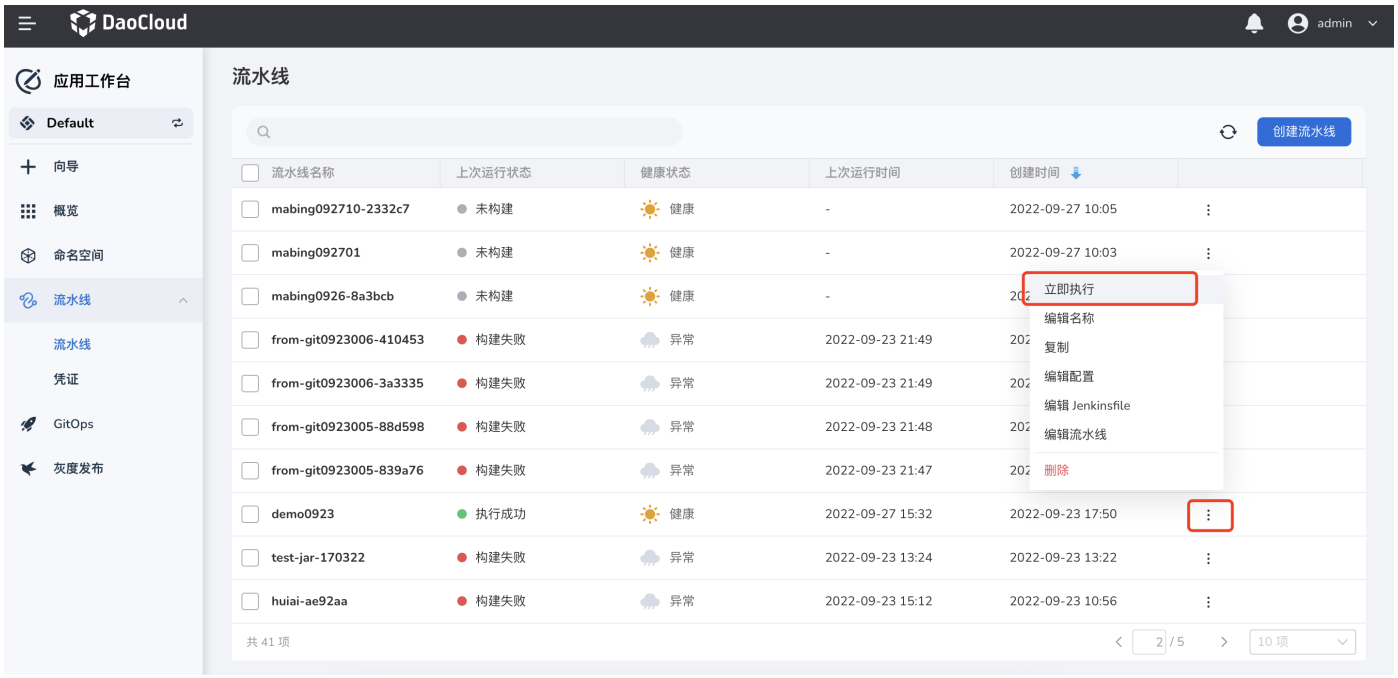


2. 选择 `配置管理` 页签，勾选 `自动扫描镜像`。

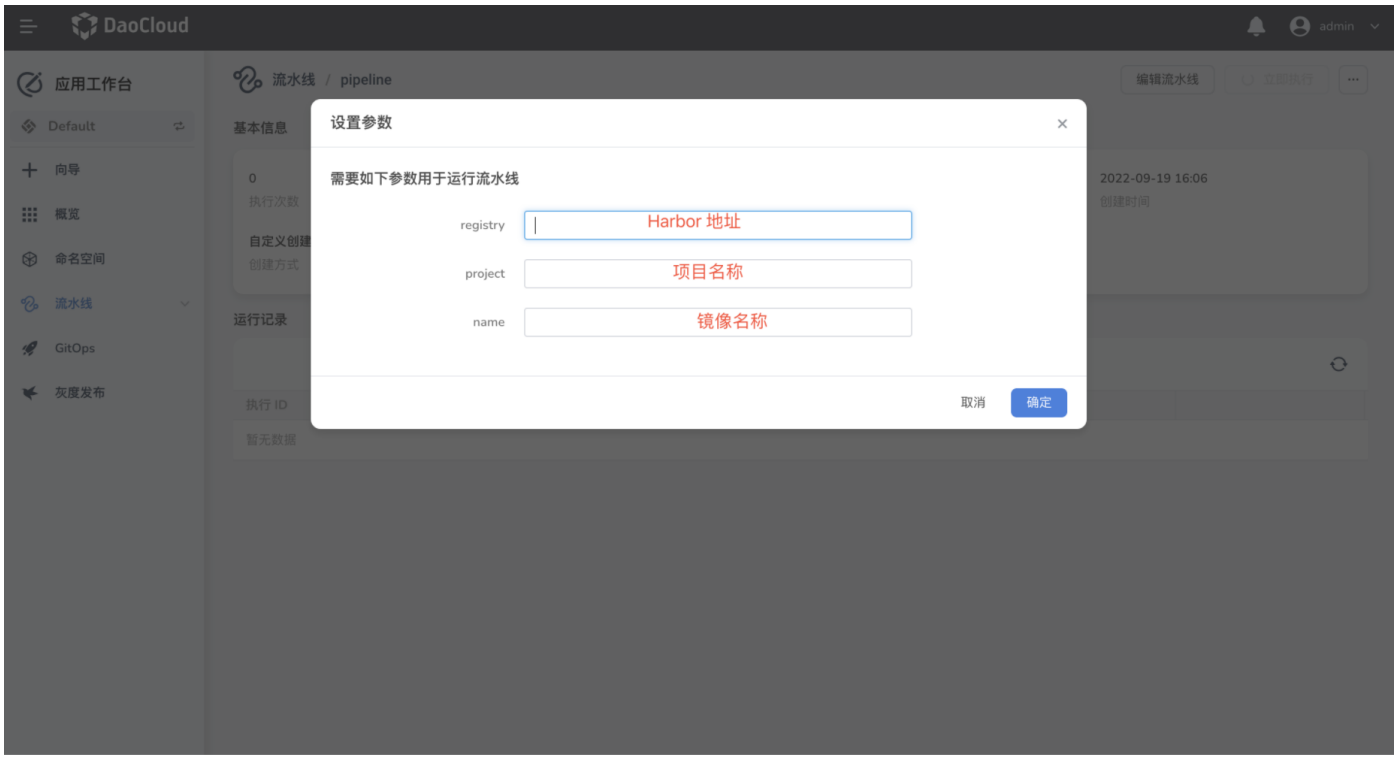
在应用工作台配置流水线

1. 在应用工作台，创建一条流水线，参考快速创建流水线，配置完毕后点击 `立即执行`。

Copyright © 2016 - 2023 DaoCloud



2. 在弹出的对话框中输入上述 Harbor 配置的项目中的镜像仓库地址。

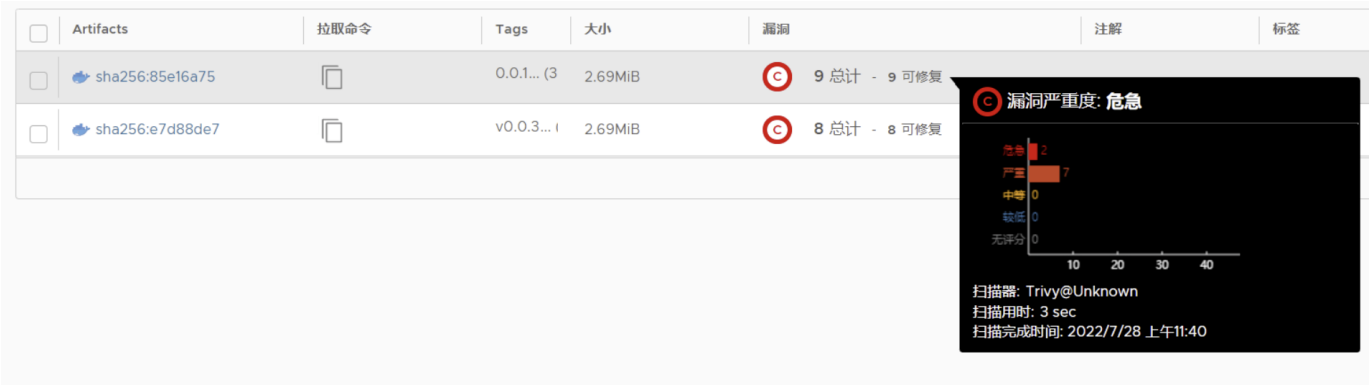


3. 等待流水线执行成功。

在 Harbor 中查看镜像安全扫描信息

在 Harbor 中依次访问 项目 → 镜像仓库，查看镜像的漏洞信息。

Copyright © 2016 - 2023 DaoCloud



Copyright © 2016 - 2023 DaoCloud

4.5 使用流水线实现代码扫描

代码仓库中的源代码作为软件的最初原始形态，其安全缺陷是导致软件漏洞的直接根源。 因此，通过代码扫描分析发现源代码中的安全缺陷是降低软件潜在漏洞的一种重要方法。

例如，SonarQube 是一款代码自动审查工具，用于检测项目代码中的 bug 错误，提高测试覆盖率等。 它可以与项目中现有的工作流程集成，以便在项目分支和拉取请求之间进行连续的代码检查。

本文将介绍如何在流水线中集成 SonarQube 来实现代码扫描能力。

Copyright © 2016 - 2023 DaoCloud

部署 SonarQube

如果您已有 SonarQube 环境，可以跳过此步骤，但需要保证当前集群可以访问到 SonarQube 服务器。

Copyright © 2016 - 2023 DaoCloud

1. 通过 Helm 安装 SonarQube。

```bash
helm repo add sonarqube <https://SonarSource.github.io/helm-chart-sonarqube>
helm repo update
kubectl create namespace sonarqube
helm upgrade --install -n sonarqube sonarqube sonarqube/sonarqube -n amamba-system  --create-namespace --set service.type=NodePort
```

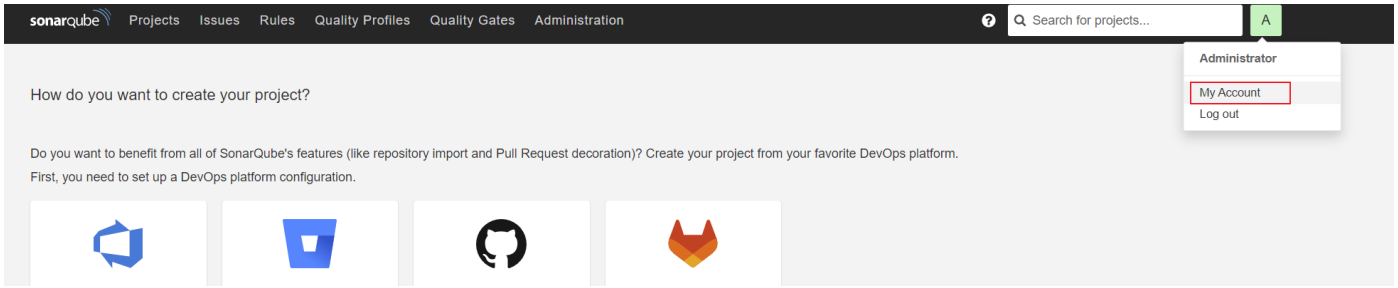
2. 查看对应 namespace 下的 Pod STATUS 都为 Running，表明 SonarQube 安装成功。

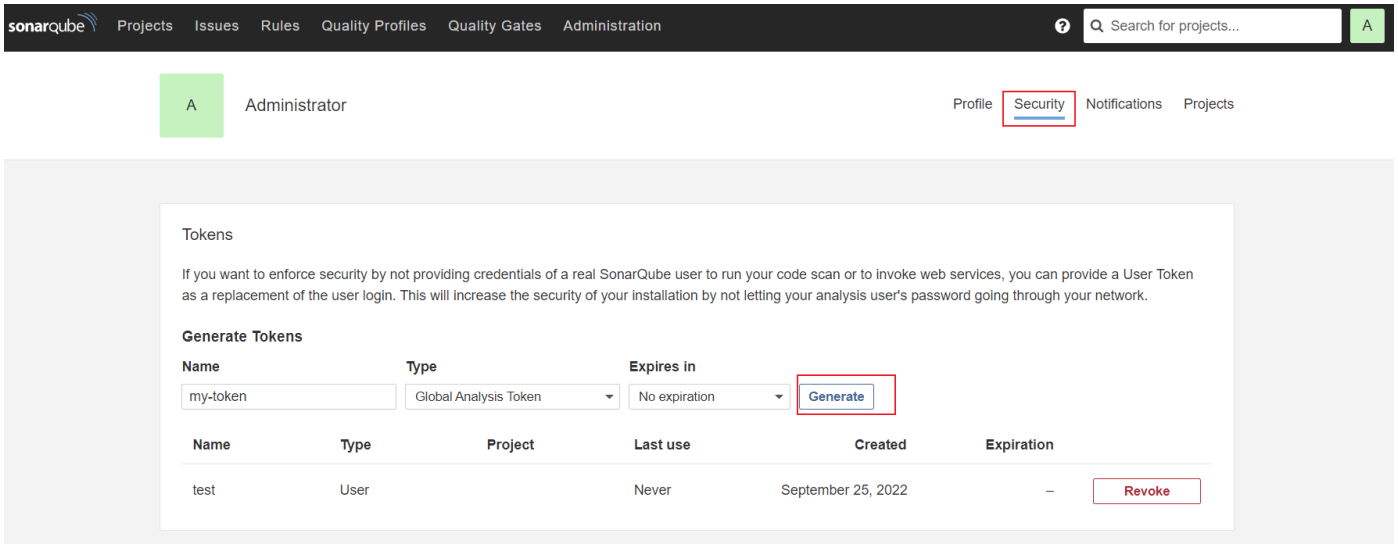
```none
kubectl get po
NAME                    READY   STATUS    RESTARTS   AGE
sonarqube-postgresql-0  1/1     Running   0          3h59m
sonarqube-sonarqube-0   1/1     Running   0          3h59m
```

3. 查看 SonarQube 控制台的访问地址。通常访问地址为 `http://<Node IP>:<NodePort>`，账户和密码为 `admin/admin`。

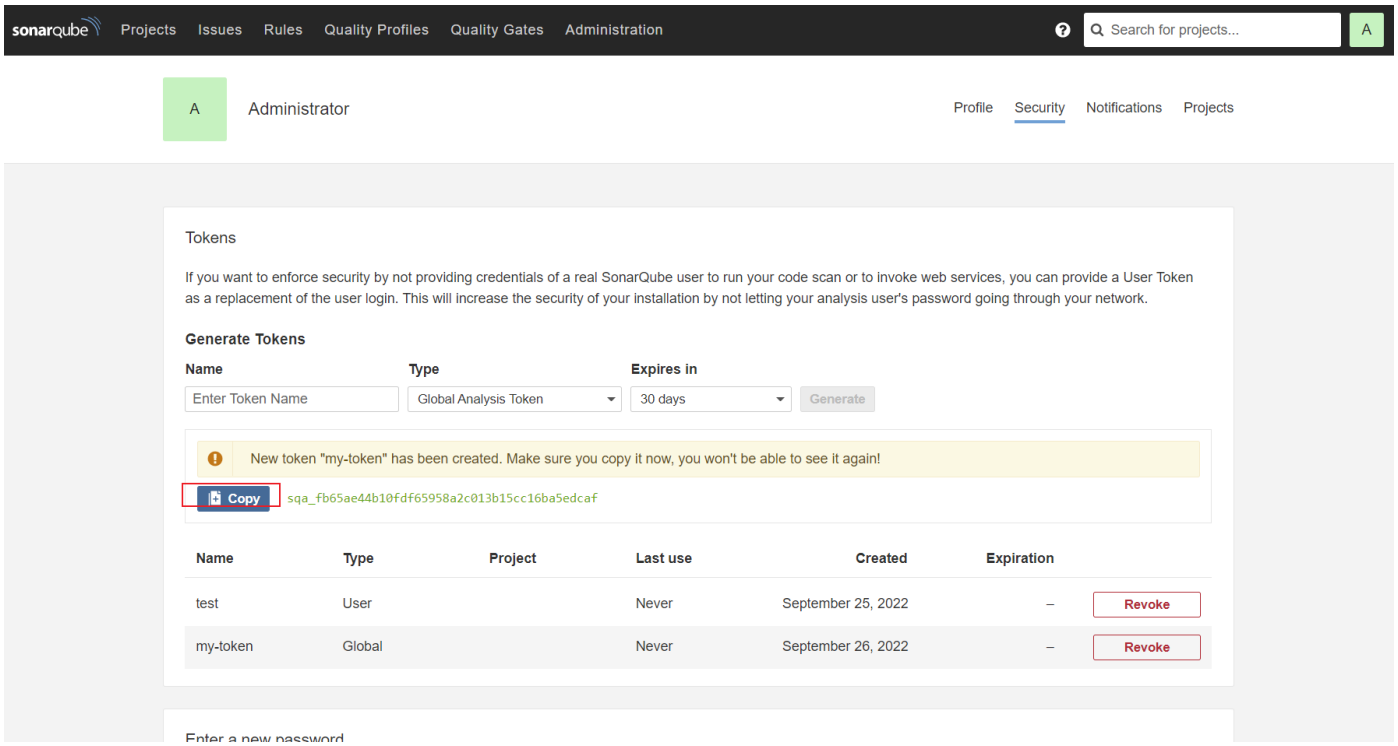
```bash
export NODE_PORT=$(kubectl get --namespace amamba-system -o jsonpath="{.spec.ports[0].nodePort}" services sonarqube-sonarqube)
export NODE_IP=$(kubectl get nodes --namespace amamba-system -o jsonpath="{.items[0].status.addresses[0].address}")
echo http://$NODE_IP:$NODE_PORT
```

4. 在 SonarQube 生成管理员令牌（Token），操作路径为：`My Account` -> `Profile` -> `Security` -> `Generate` -> `Copy`





Copyright © 2016 - 2023 DaoCloud



5. 在 SonarQube 创建 Webhook 服务器，操作路径：

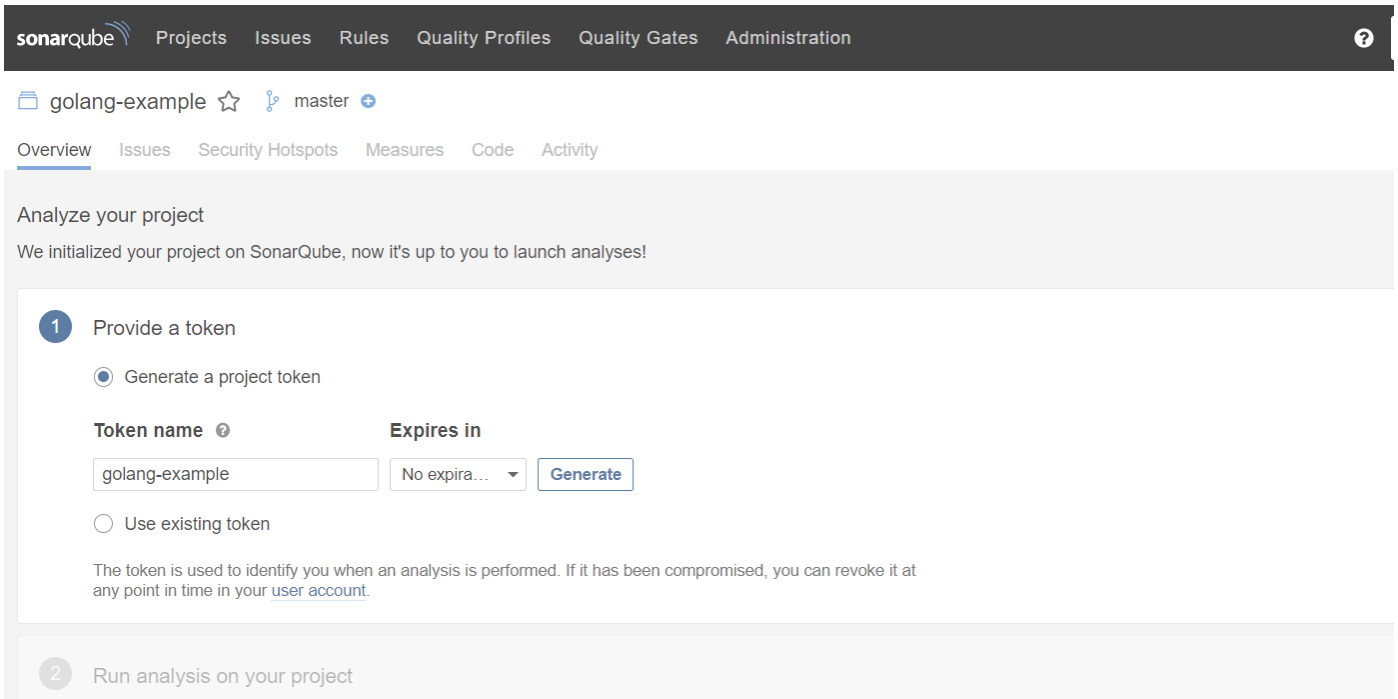
a. 操作路径为 `Administration` -> `Configuration` -> `Webhooks` -> `Create`。

b. 在弹出的对话框中输入 `Name` 和 `Jenkins Console URL`（即 SonarQube Webhook 地址，这个地址是前面获取的 SonarQube 地址 + sonarqube-webhook）。

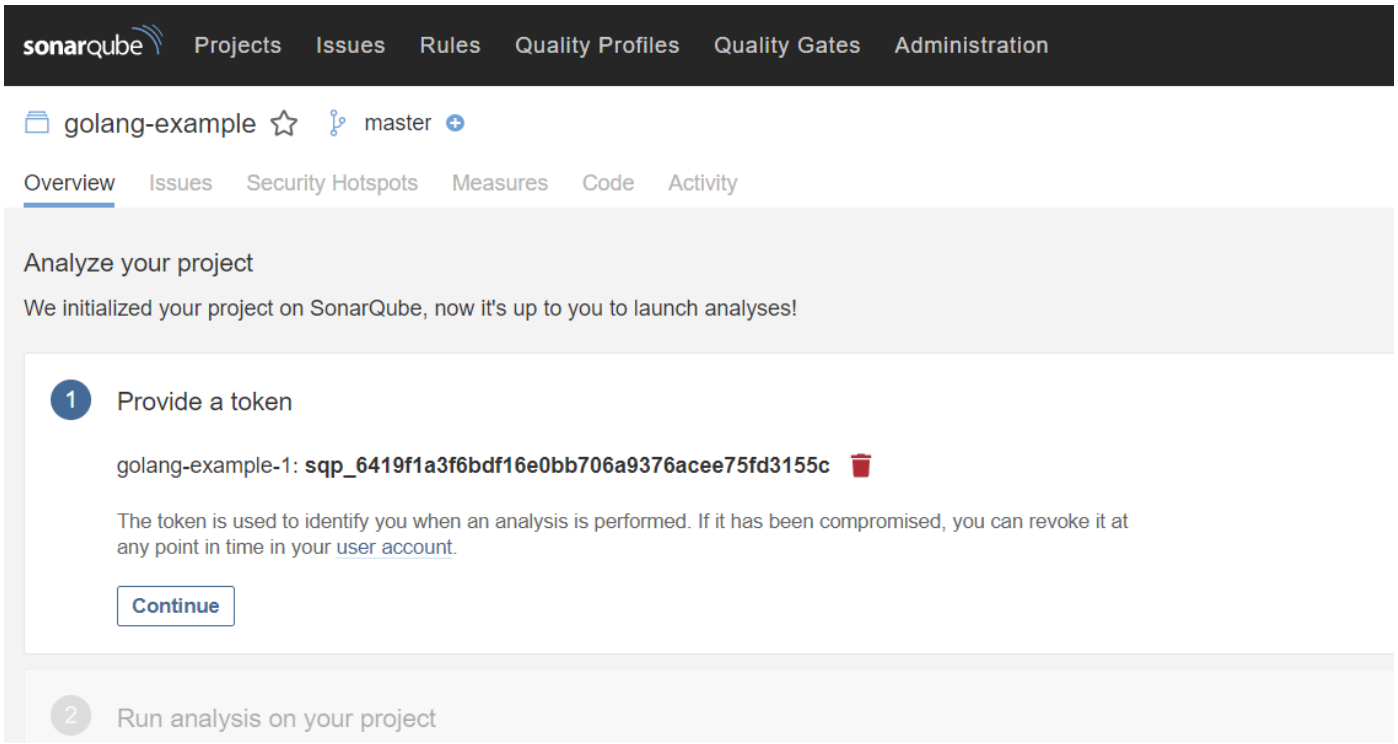
c. 点击 `Create` 完成操作。

6. 提供一个暴露 Jenkins 地址，这是为了让用户将 SonarQube 服务器添加至 Jenkins。

7. 为新项目创建 SonarQube Token，操作路径为 `Create new project` -> `Set Up` -> `Generate` -> `Continue`。

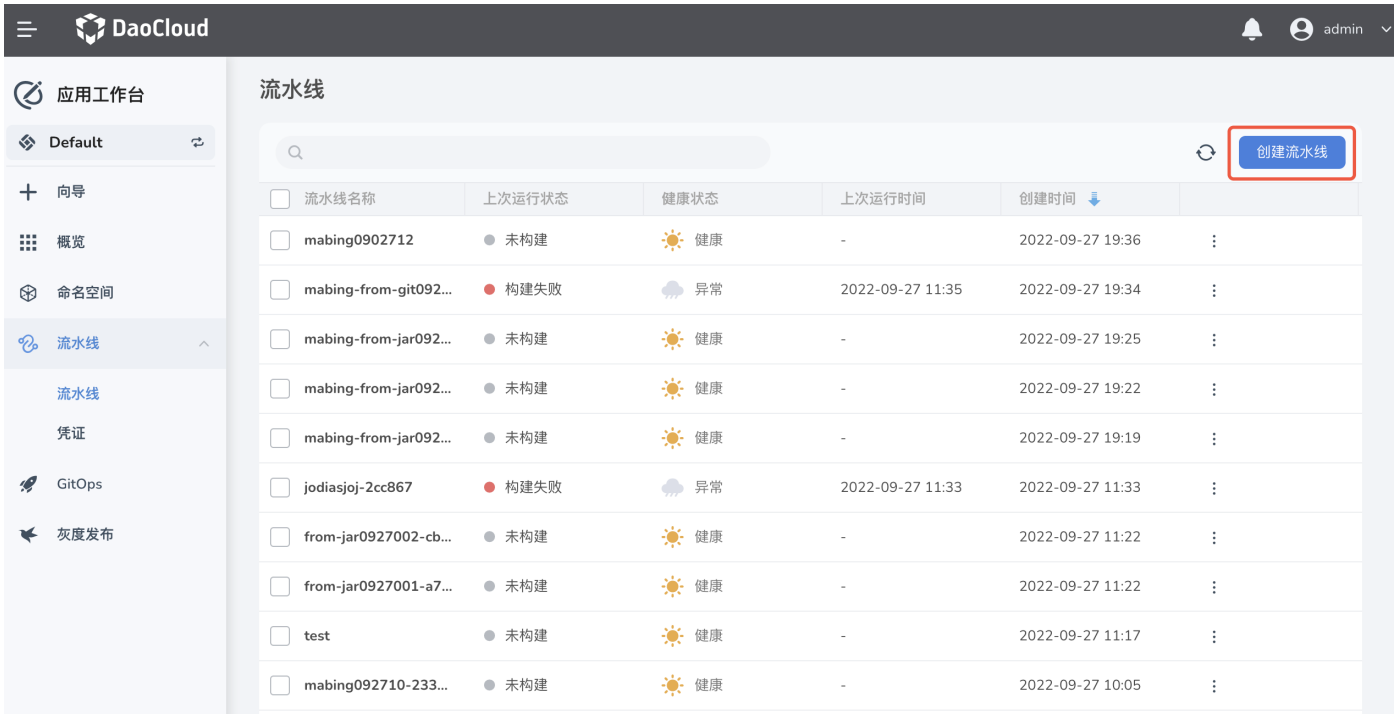


Copyright © 2016 - 2023 DaoCloud



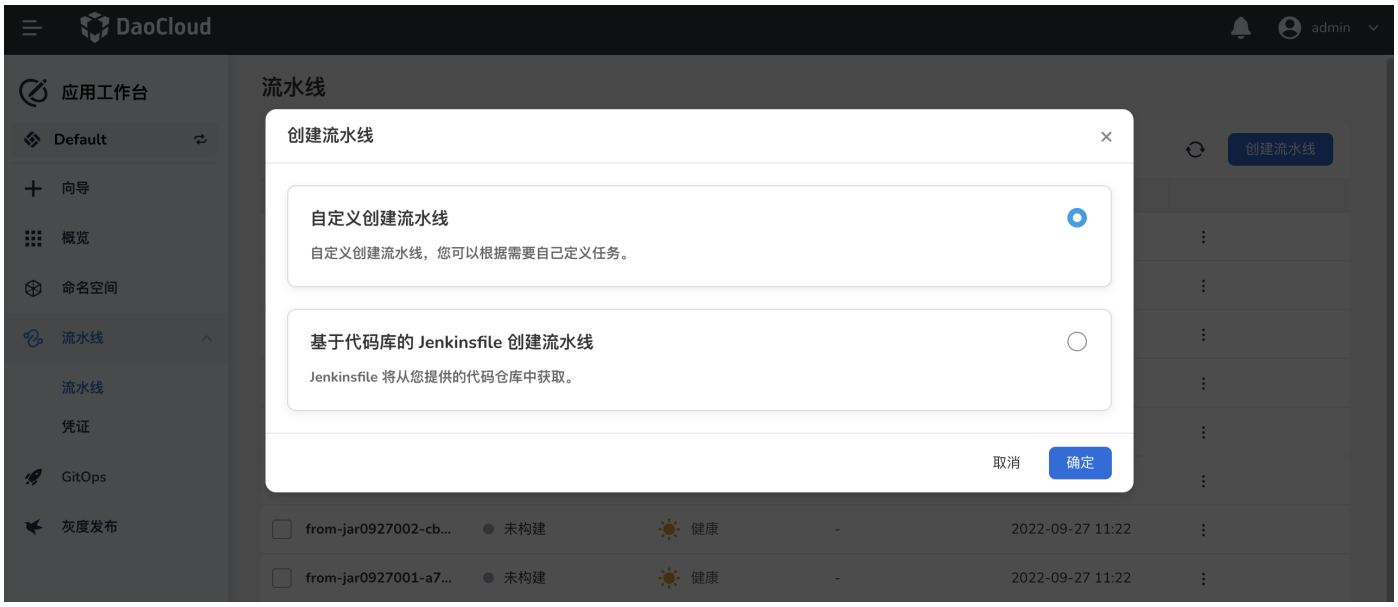
创建流水线

1. 在流水线页面，点击 创建流水线 。



2. 选择 自定义创建 。

Copyright © 2016 - 2023 DaoCloud

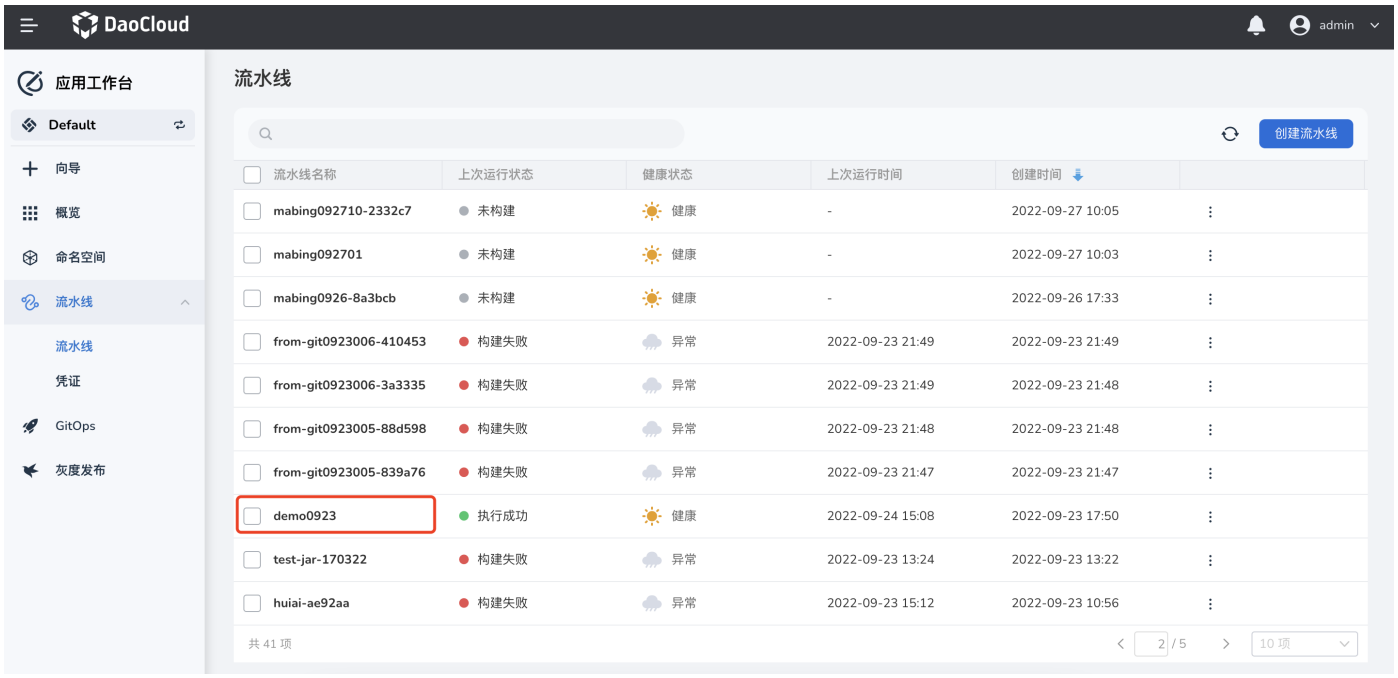


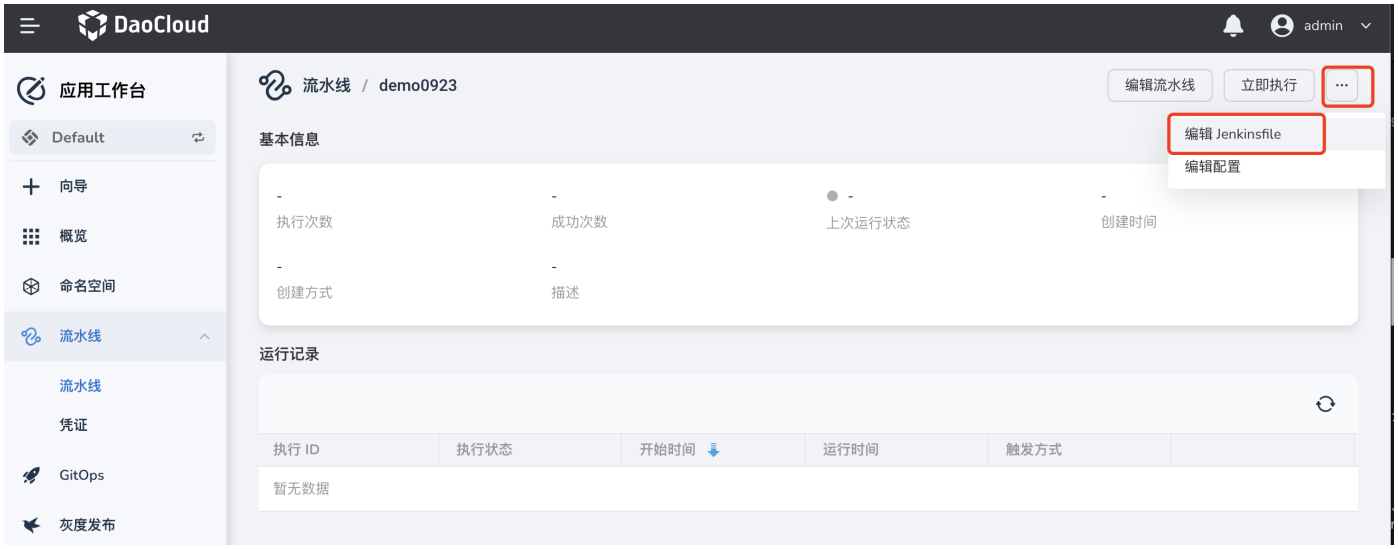
3. 输入名称，其他可使用默认值，点击 `确定` 。

编辑 Jenkinsfile

1. 点击一个流水线进入其详情页面，在右上角点击 `...` -> `编辑 Jenkinsfile` 。

Copyright © 2016 - 2023 DaoCloud





2. 将以下 YAML 代码复制粘贴至 jenkinsfile。

```yaml pipeline { agent { node { label 'go16' }

} stages { stage('git clone') { steps { git(credentialsId: 'mabing-gitlab', branch: 'master', url: 'https://gitlab.daocloud.cn/bing.ma/jenkins-sonarqube-demo.git') } }

```
stage('unit test') {
steps {
    container('go') {
        sh 'go test -json > test-report.out'
        sh 'go test -coverprofile=coverage.out'
    }
}
}

stage("SonarQube analysis") {
steps {
    container('go') {
    withSonarQubeEnv('demo-dev-sonarqube') {
        sh 'sonar-scanner -Dsonar.projectKey=golang-demo -Dsonar.sources=. -Dsonar.host.url=http://10.6.182.101:32313 -
Dsonar.login=4b337cbbafd89ae9bca46a746cddab4c993d2a7a'
    }
    }
}
}

stage("Quality Gate") {
steps {
```


Copyright © 2016 - 2023 DaoCloud
```

```
    container('go') {
    timeout(time: 1, unit: 'HOURS') {
        waitForQualityGate abortPipeline: false // 这里设置为false，表示即使没有通过sonarQube的QUALITY GATE，流水线也可以继续
    }
    }
}
}

stage('build & push') {
steps {
    container('go') {
    withCredentials([usernamePassword(credentialsId:'docker-credential',passwordVariable:'PASS',usernameVariable:'USER')]) {
        sh 'go build -o simple-http-server main/main.go'
        sh 'docker build -f Dockerfile . -t $registry/$project/$name:latest'
        sh 'docker login $registry -u $USER -p $PASS'
        sh 'docker push $registry/$project/$name:latest'
    }

    }

}
}

stage('deply'){
steps {
    container('go'){
    withCredentials([kubeconfigFile(credentialsId: 'kubeconfig-credential', variable: 'KUBECONFIG')]) {
        sh 'kubectl apply -f deploy.yaml'
    }
    }
}

}
```

} parameters { string(name: 'registry', defaultValue: 'release-ci.daocloud.io', description: '') string(name: 'project', defaultValue: 'demo', description: '') string(name: 'name', defaultValue: 'http-hello', description: '') } } ```

!!! note

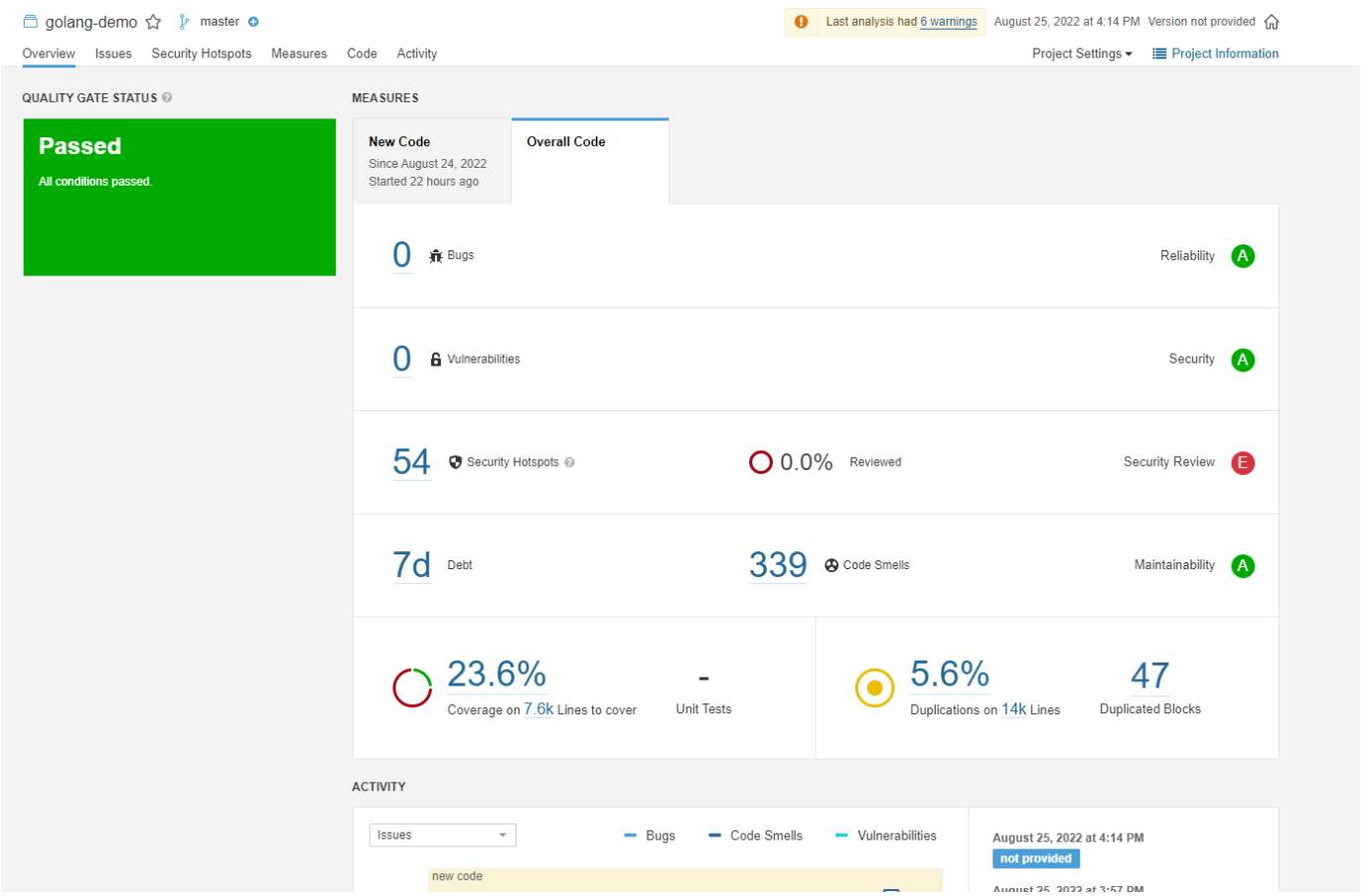
```
在上述代码中，

- waitForQualityGate abortPipeline: false，表示没有通过门禁质量检查也可以继续流水线。如果是 true，则表示放弃
- 对于 withSonarQubeEnv 下的 shell 语句，可以从上一步的 SonarQube 中复制
```

3. 保存后立即运行该流水线。

前往 SonarQube 查看代码扫描结果

等待流水线运行成功后，前往 SonarQube 查看代码扫描结果。

Copyright © 2016 - 2023 DaoCloud



Copyright © 2016 - 2023 DaoCloud

4.6 Jenkins 场景配置

Jenkins 分为 master 和 agent。master 主要用于存储配置，插件和协调，agent 和 master 之间通过 agent 这个 Pod 里的 jnlp 容器通信。流水线的工作都在 agent 上，是资源消耗大户。

本文推荐的配置是基于客户并行运行的所有流水线时不会导致整个集群资源崩溃，尤其是 jenkins master 不会出现被驱逐等异常情况。能够并行多少条流水线主要取决于实际 agent 的资源消耗和 K8s 集群资源的大小。

场景 1：并行 50 条流水线

master 配置

```
resources:
  requests:
    cpu: "1"
    memory: "2Gi"
  limits:
    cpu: "2"
    memory: "4Gi"
  JavaOpts: |-
    -XX:MaxRAMPercentage=70.0
    -XX:MaxRAM=3g
    -Dhudson.slaves.NodeProvisioner.initialDelay=20
    -Dhudson.slaves.NodeProvisioner.MARGIN=50
    -Dhudson.slaves.NodeProvisioner.MARGIN0=0.85
    -Dhudson.model.LoadStatistics.clock=5000
    -Dhudson.model.LoadStatistics.decay=0.2
    -Dhudson.slaves.NodeProvisioner.recurrencePeriod=5000
    -Dhudson.security.csrf.DefaultCrumbIssuer.EXCLUDE_SESSION_ID=true
    -Dio.jenkins.plugins.casc.ConfigurationAsCode.initialDelay=10000
    -Djenkins.install.runSetupWizard=false
    -XX:+UseG1GC
    -XX:+UseStringDeduplication
    -XX:+ParallelRefProcEnabled
    -XX:+DisableExplicitGC
    -XX:+UnlockDiagnosticVMOptions
    -XX:+UnlockExperimentalVMOptions
    -javaagent:/otel-auto-instrumentation/javaagent.jar
```

agent 配置

请参照实际的流水线消耗资源设置，因为主要的资源消耗都是来自 agent。

```
resources:
  requests:
    cpu: "100m"
    memory: "256Mi"
  limits:
    cpu: "100m"
    memory: "128Mi"
```

场景 2：并行 100 条流水线

master 配置

```
resources:
  requests:
    cpu: "2"
    memory: "4Gi"
  limits:
    cpu: "2"
    memory: "8Gi"
  JavaOpts: |-
    -XX:MaxRAMPercentage=70.0
    -XX:MaxRAM=6g
    -Dhudson.slaves.NodeProvisioner.initialDelay=20
    -Dhudson.slaves.NodeProvisioner.MARGIN=50
    -Dhudson.slaves.NodeProvisioner.MARGIN0=0.85
    -Dhudson.model.LoadStatistics.clock=5000
    -Dhudson.model.LoadStatistics.decay=0.2
    -Dhudson.slaves.NodeProvisioner.recurrencePeriod=5000
    -Dhudson.security.csrf.DefaultCrumbIssuer.EXCLUDE_SESSION_ID=true
    -Dio.jenkins.plugins.casc.ConfigurationAsCode.initialDelay=10000
    -Djenkins.install.runSetupWizard=false
    -XX:+UseG1GC
    -XX:+UseStringDeduplication
```


Copyright © 2016 - 2023 DaoCloud

```
    -XX:+ParallelRefProcEnabled
    -XX:+DisableExplicitGC
    -XX:+UnlockDiagnosticVMOptions
    -XX:+UnlockExperimentalVMOptions
    -javaagent:/otel-auto-instrumentation/javaagent.jar
```

agent 配置

参照场景 1 的 agent 配置

场景 3：并行 200 条流水线

master 配置

```
resources:
  requests:
    cpu: "4"
    memory: "8Gi"
  limits:
    cpu: "8"
    memory: "12Gi"
  JavaOpts: |-
    -XX:MaxRAMPercentage=70.0
    -XX:MaxRAM=10g
    -Dhudson.slaves.NodeProvisioner.initialDelay=20
    -Dhudson.slaves.NodeProvisioner.MARGIN=50
    -Dhudson.slaves.NodeProvisioner.MARGIN0=0.85
    -Dhudson.model.LoadStatistics.clock=5000
    -Dhudson.model.LoadStatistics.decay=0.2
    -Dhudson.slaves.NodeProvisioner.recurrencePeriod=5000
    -Dhudson.security.csrf.DefaultCrumbIssuer.EXCLUDE_SESSION_ID=true
    -Dio.jenkins.plugins.casc.ConfigurationAsCode.initialDelay=10000
    -Djenkins.install.runSetupWizard=false
    -XX:+UseG1GC
    -XX:+UseStringDeduplication
    -XX:+ParallelRefProcEnabled
    -XX:+DisableExplicitGC
    -XX:+UnlockDiagnosticVMOptions
    -XX:+UnlockExperimentalVMOptions
    -javaagent:/otel-auto-instrumentation/javaagent.jar
```

agent 配置

参照场景 1 的 agent 配置

Copyright © 2016 - 2023 DaoCloud

5. User guide

5.1 Gitops

Argo CD 应用状态说明

本页面介绍了 Argo CD 应用的健康状态、同步状态说明。

健康状态

健康状态	描述
健康	资源健康
已降级	资源已经被降级
进行中	资源还不健康，但仍在进行中，可能很快就会健康
暂停	资源被暂停并等待一些外部事件恢复（例如暂停的 CronJob 或暂停的部署）
未知	无法判断当前健康状态
丢失	资源已缺失

同步状态

同步状态	描述
已同步	集群中部署的资源与仓库中期望的状态一致
未同步	集群中部署的资源与仓库中期望的状态不一致
未知	无法判断当前同步状态

Copyright © 2016 - 2023 DaoCloud

创建 Argo CD 应用

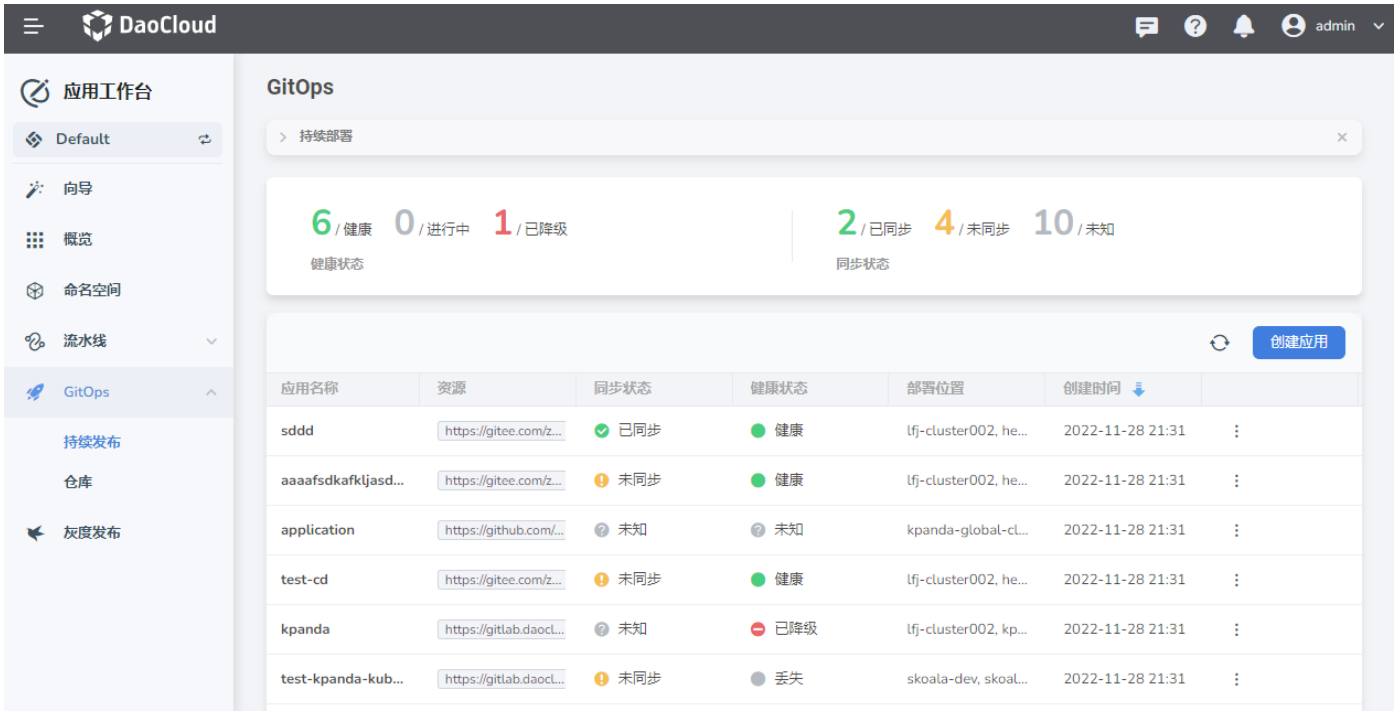
应用工作台基于开源软件 Argo CD 实现持续部署。本页面演示如何实现应用的持续部署。

前提条件

- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 `workspace edit` 角色。 参考创建工作空间、用户和角色。
- 准备一个 Git 仓库。

创建 Argo CD 应用

1. 在 应用工作台 -> 持续发布 页面中，点击 创建应用 按钮。

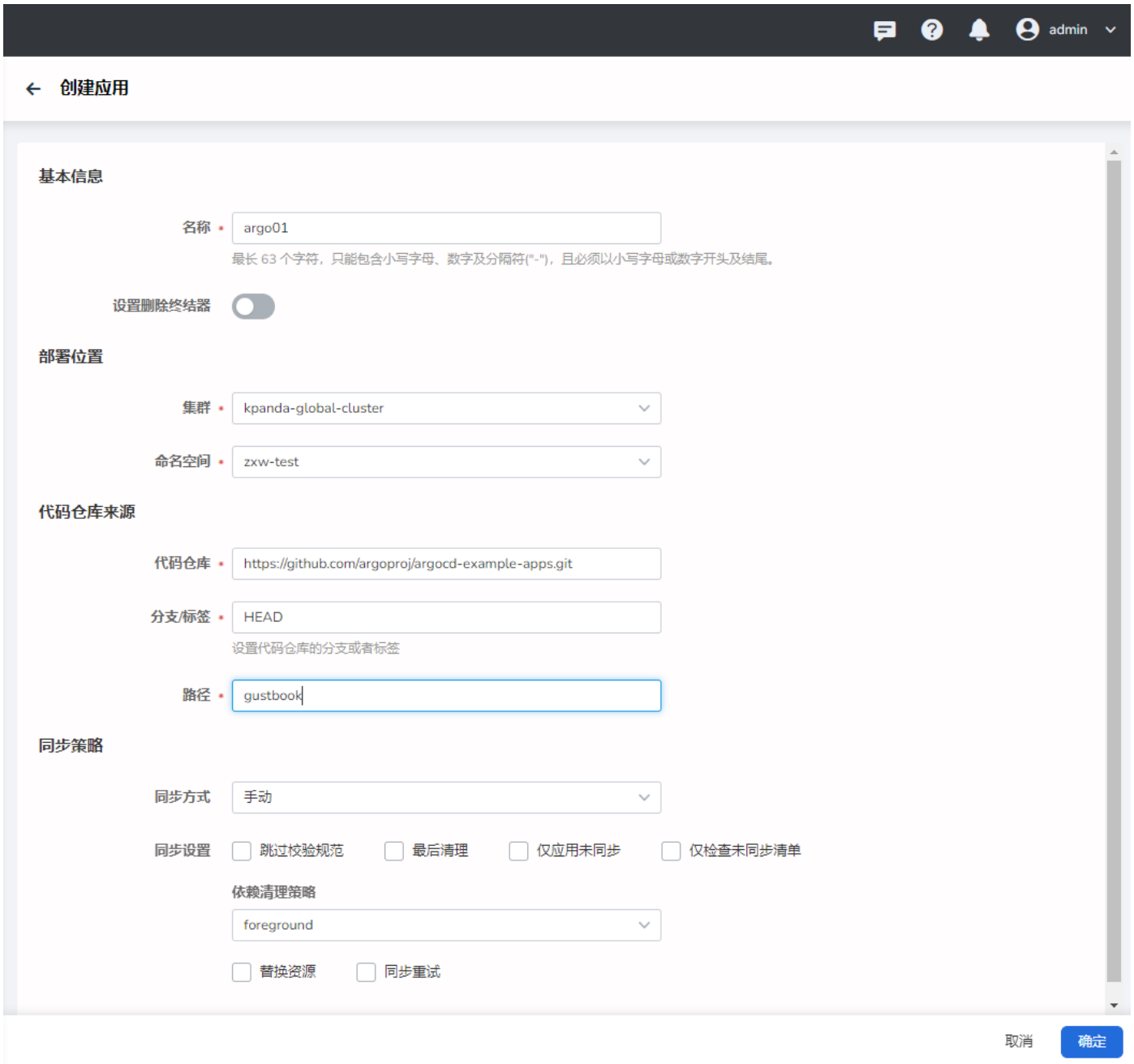


Copyright © 2016 - 2023 DaoCloud

2. 在 创建应用 页面中，分配配置 基本信息 、 部署位置 、 代码仓库来源 和 同步策略 后，点击 确定 。

- 代码仓库来源：
- 代码仓库：选择已导入的代码仓库或者输入公开的代码仓库地址。例如：https://github.com/argoproj/argocd-example-apps.git
- 分支/标签：设置代码仓库的分支或标签，默认为HEAD
- 路径：填写清单文件路径，例如 gustbook
- 同步策略：
- 手动同步：手动点击是否同步
- 自动同步：自动检测代码仓库中清单文件的变化，一旦变化后就会立即同步应用资源至最新状态。并支持清理资源和自恢复选项：
- 清理资源：同步时删除代码仓库不再定义的资源
- 自恢复：确保与代码仓库中期望的状态同步
- 同步设置：
- 跳过校验规范：跳过应用清单文件规范校验
- 最后清理：当所有的资源都已经同步且处于健康状态后，才会删除不存在的资源
- 仅应用未同步：仅应用未同步的资源
- 依赖清理策略：选择一个具体的清理策略：
- foreground：先删除完所有依赖对象之后， 再删除属主对象
- background：先删除属主对象之后，再删除所有依赖对象
- orphan：删除属主对象之后，所有依赖对象仍然存在
- 替换资源：是否需要替换已存在的资源
- 同步重试：对应用同步重试参数化，支持设置重试最大次数、重试持续时间、重试持续最大时间、因子

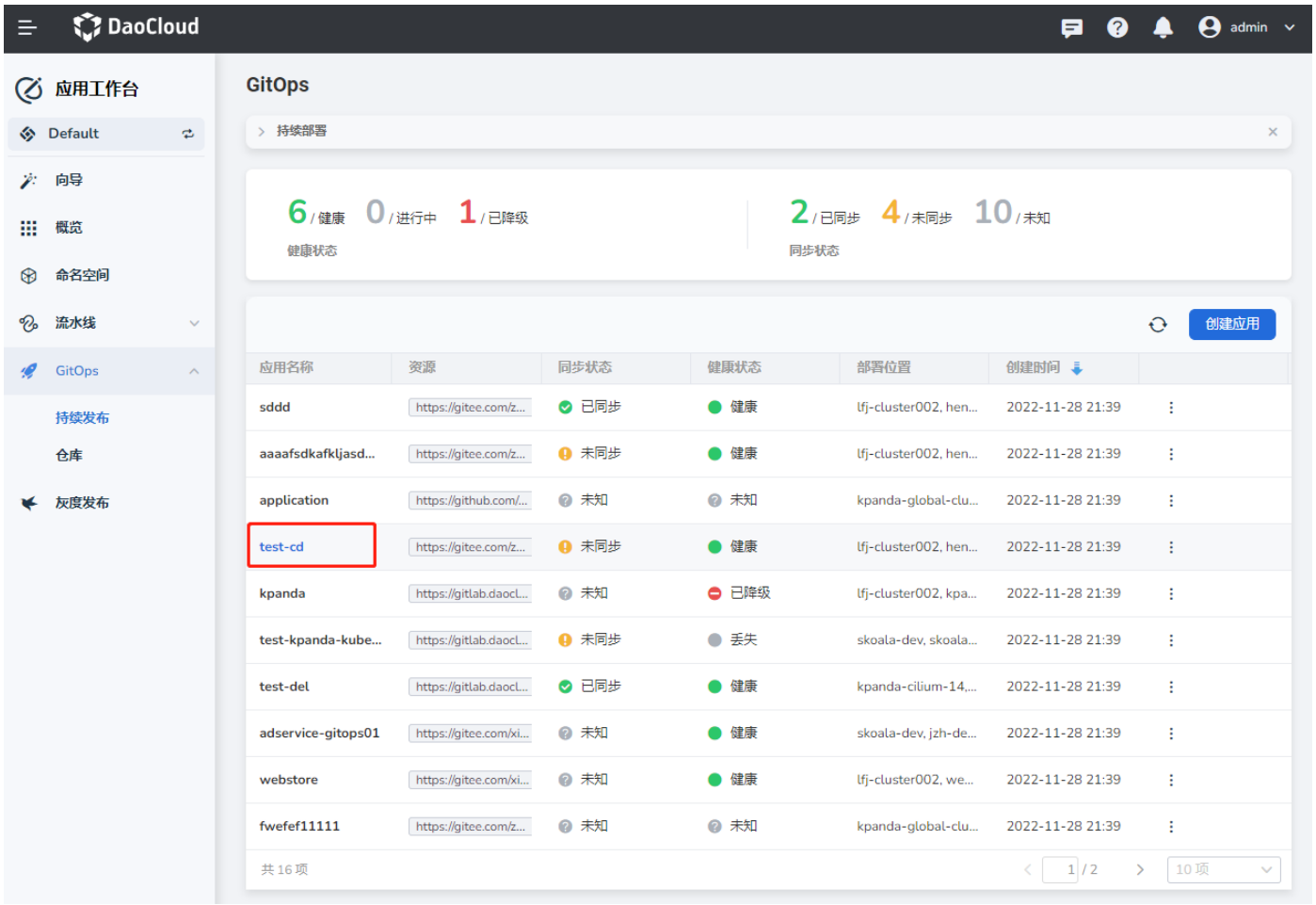
Copyright © 2016 - 2023 DaoCloud



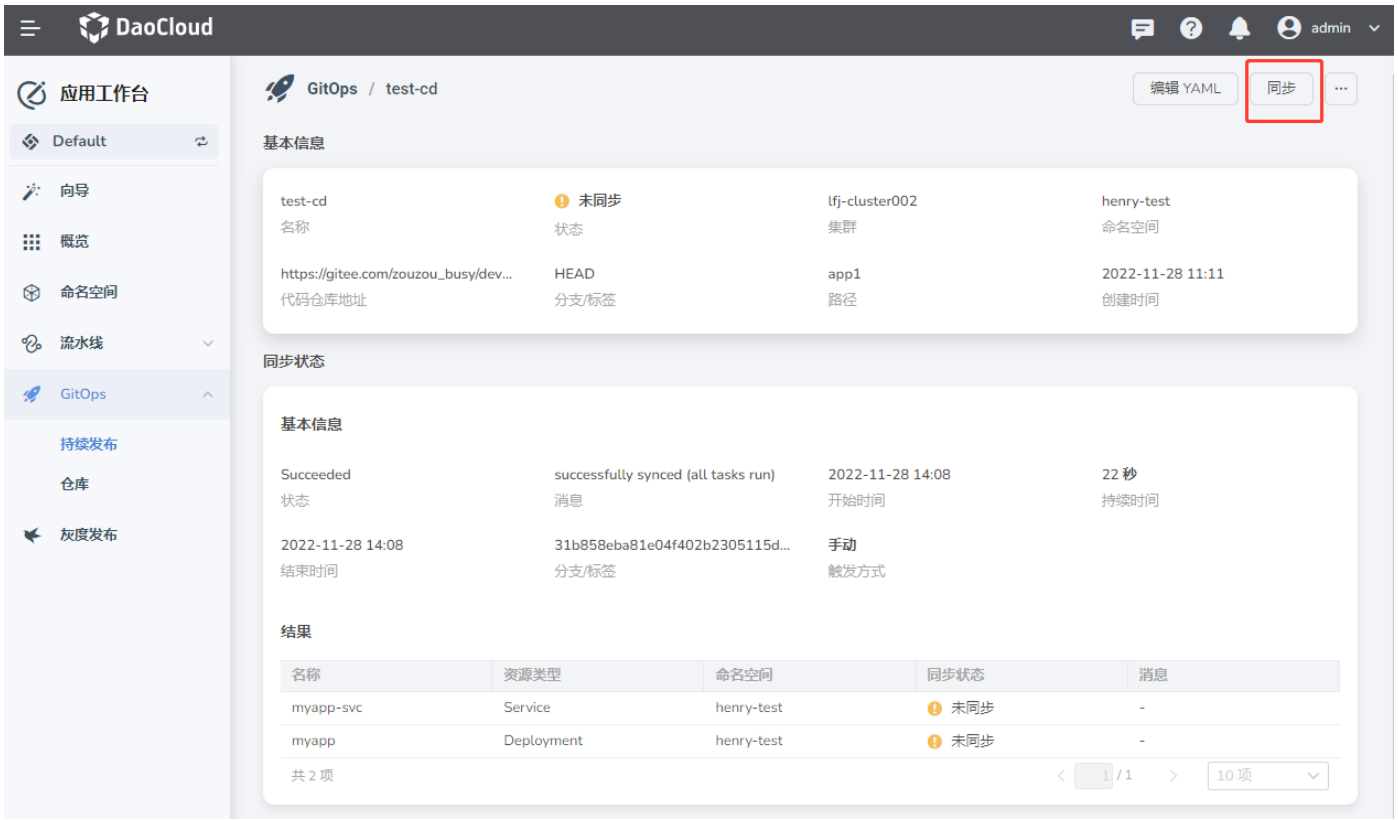
查看应用

1. 创建成功后，点击应用名称进入详情页面，可以查看应用详情。

Copyright © 2016 - 2023 DaoCloud

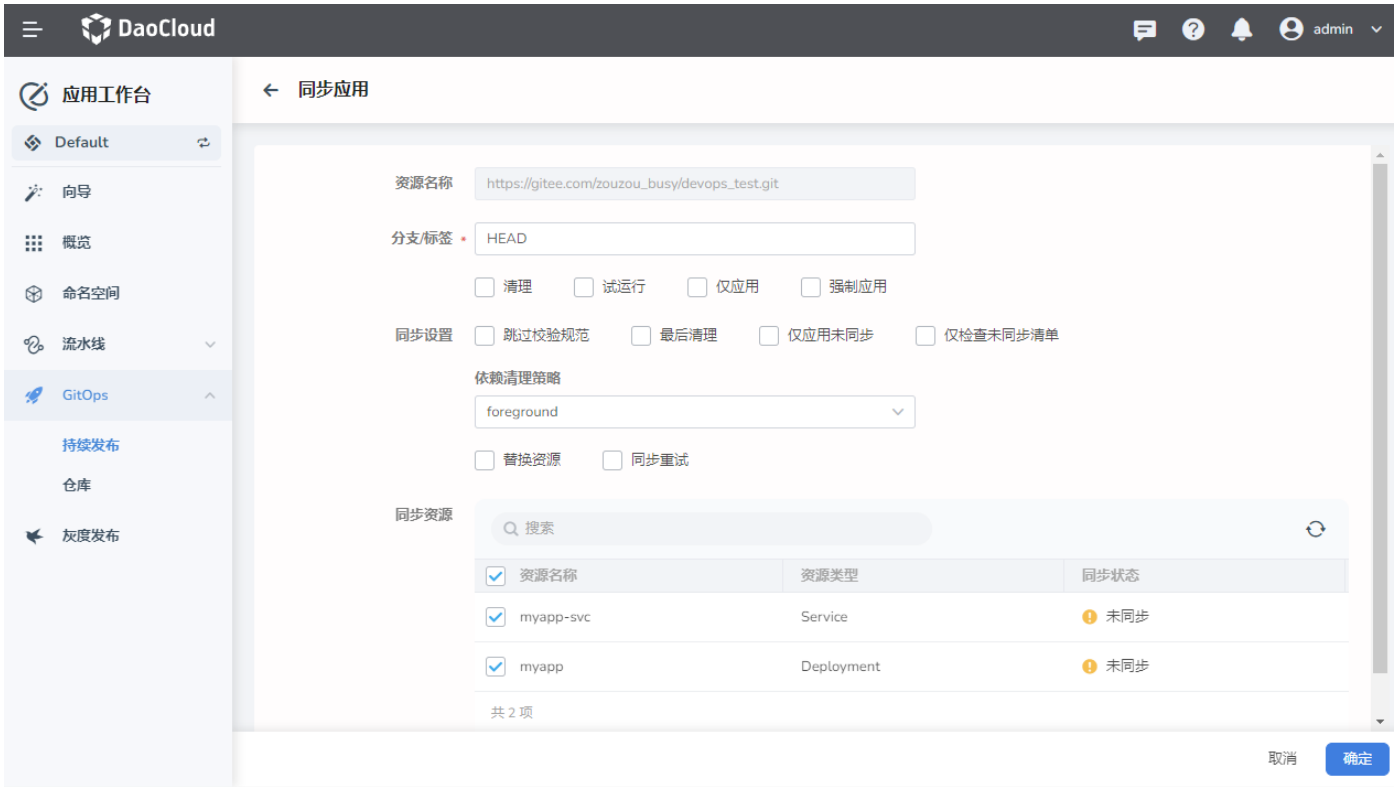


2. 由于同步方式选择的是 `手动同步`，所以我们需要手动进行同步，点击 `同步` 。



Copyright © 2016 - 2023 DaoCloud

3. 同步过程中具体参数说明请参考手动同步应用，点击 `确定` 。



4. 等待同步成功后，查看同步结果。

Copyright © 2016 - 2023 DaoCloud

导入仓库

本页展示如何导入仓库。

前提条件

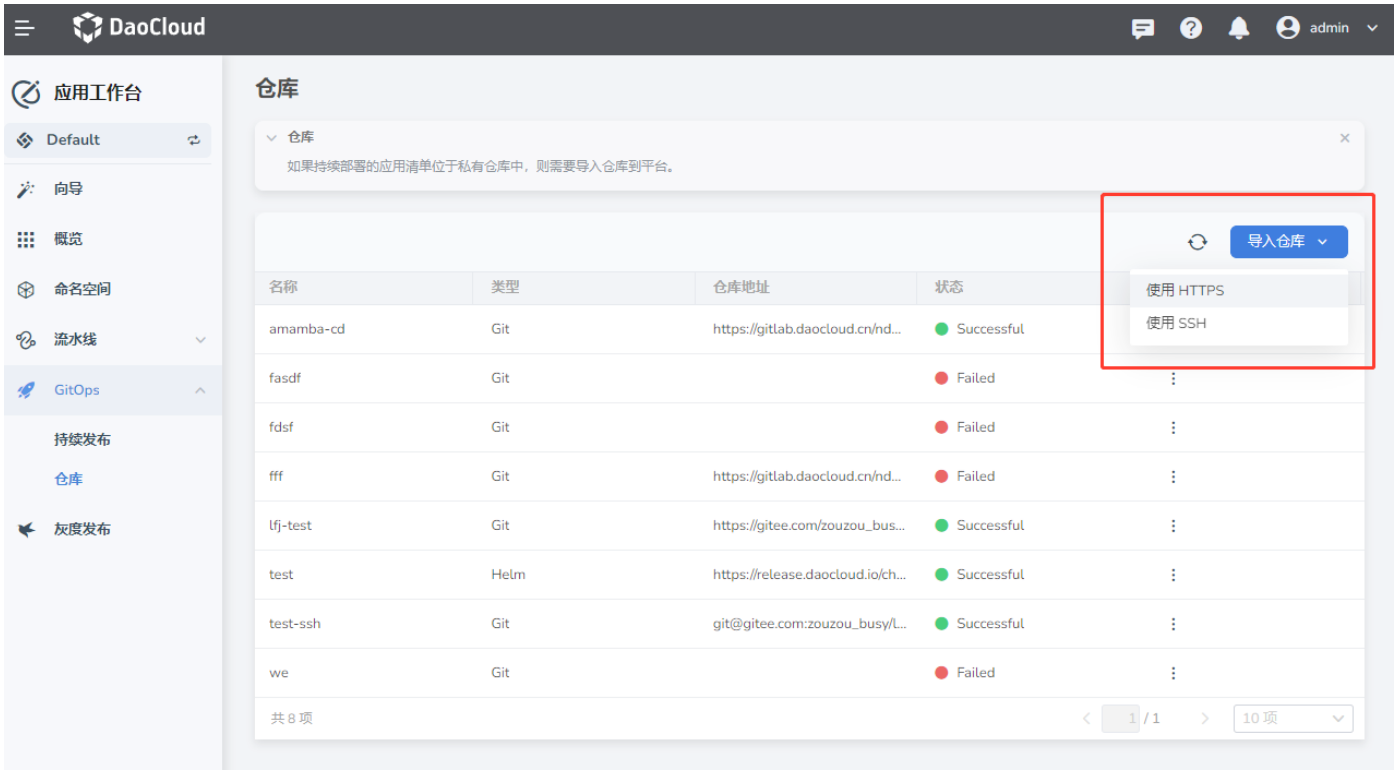
- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 `workspace edit` 角色。 参考创建工作空间、用户和角色。
- 准备一个 Git 仓库。

导入仓库

如果持续部署应用的清单文件所在的代码仓库不是公开的，则需要事先将仓库导入至应用工作台。应用工作台目前支持两种导入方式：`使用 HTTPS 导入仓库` 和 `使用 SSH 导入仓库`。

使用 HTTPS 导入仓库

1. 在 `应用工作台` -> `GitOps` -> `仓库` 页面中，点击 `导入仓库` 按钮，选择 `使用 HTTPS`。

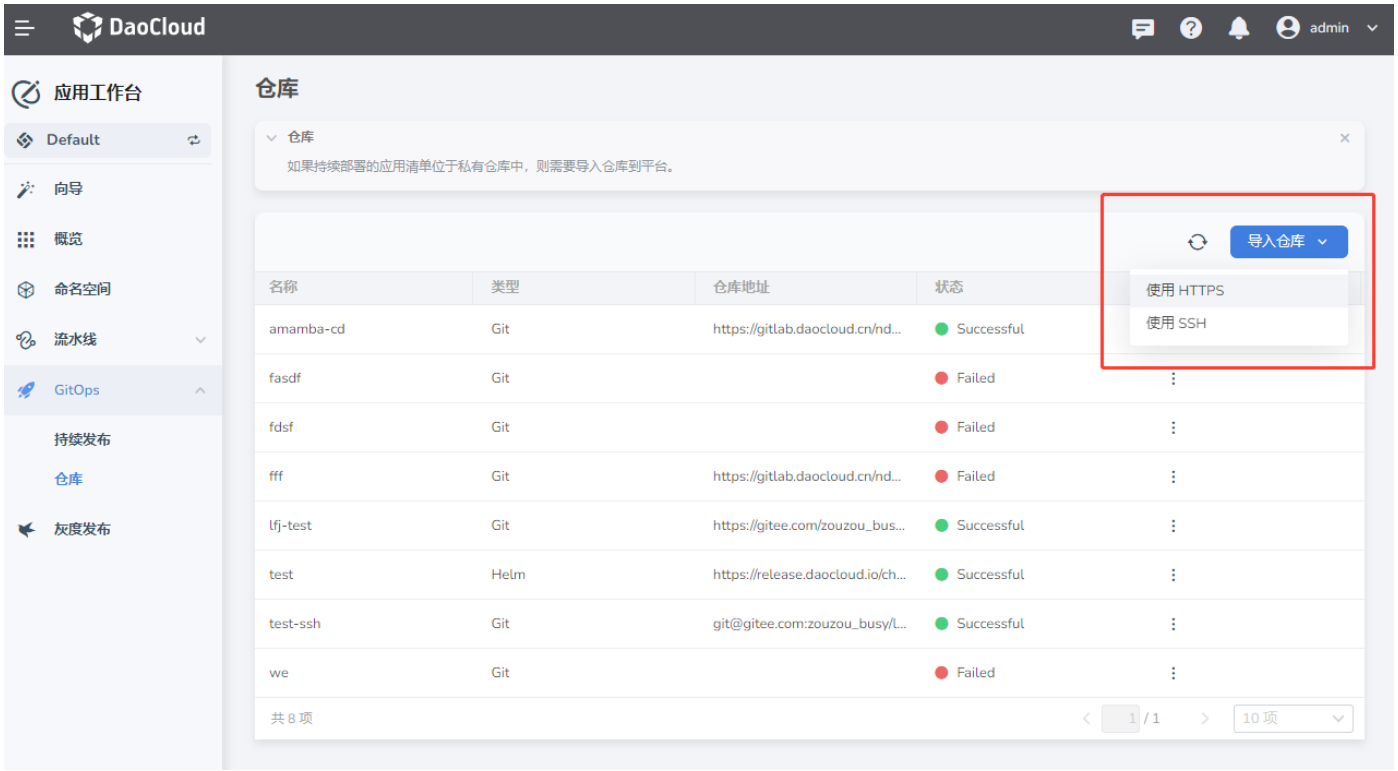


2. 在 `使用 HTTPS 导入仓库` 页面中，配置相关参数后点击 `确定`。

Copyright © 2016 - 2023 DaoCloud

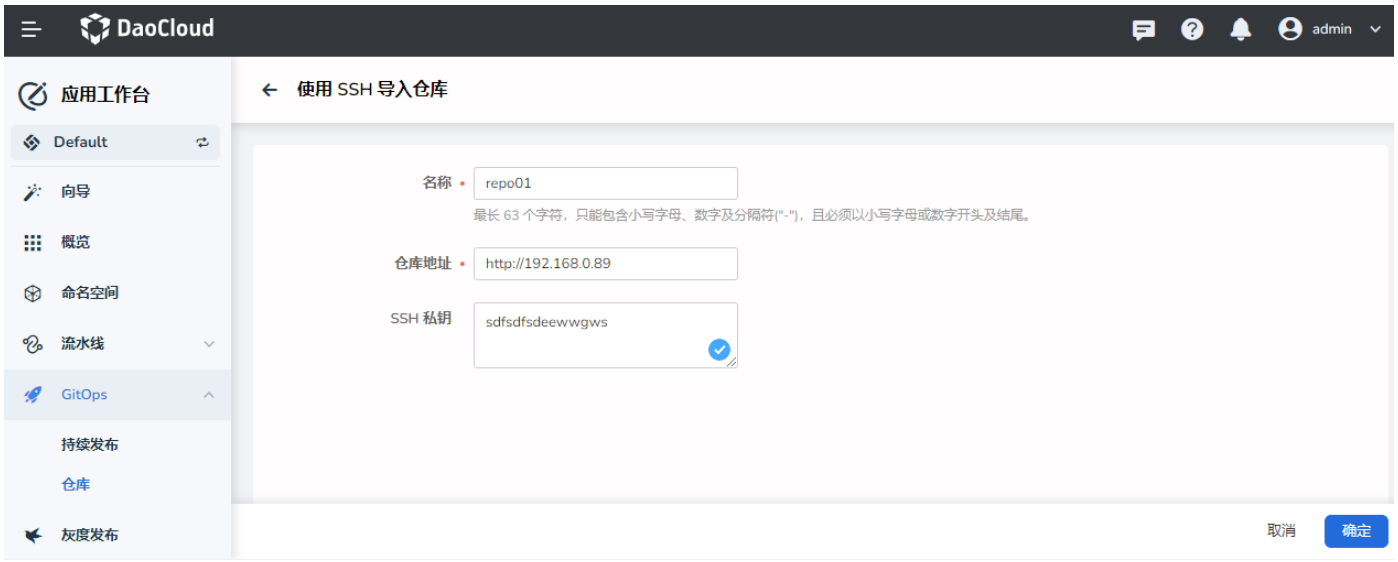
使用 SSH 导入仓库

1. 在 `应用工作台` -> `GitOps` -> `仓库` 页面中，点击 `导入仓库` 按钮，选择 `使用 SSH` 。



2. 在 `使用 SSH 导入仓库` 页面中，配置相关参数后点击 `确定` 。

Copyright © 2016 - 2023 DaoCloud

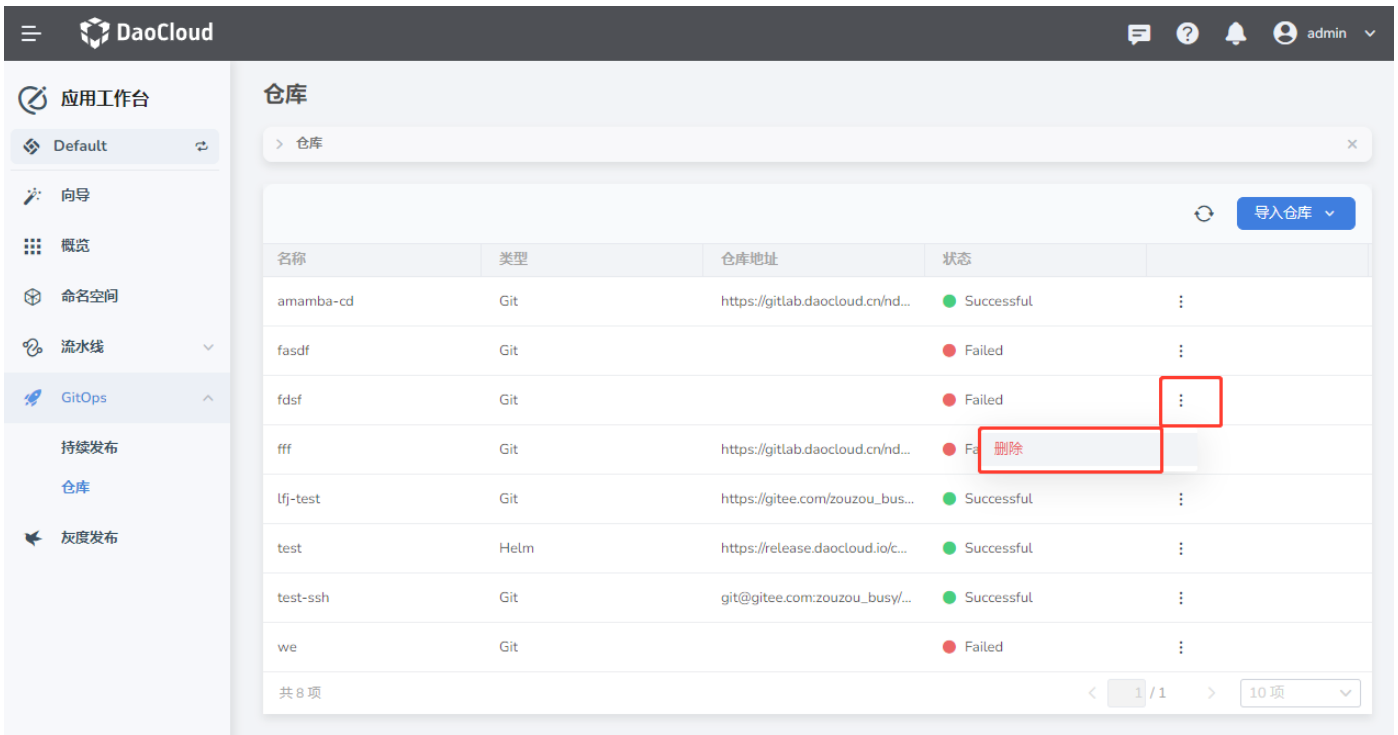


Copyright © 2016 - 2023 DaoCloud

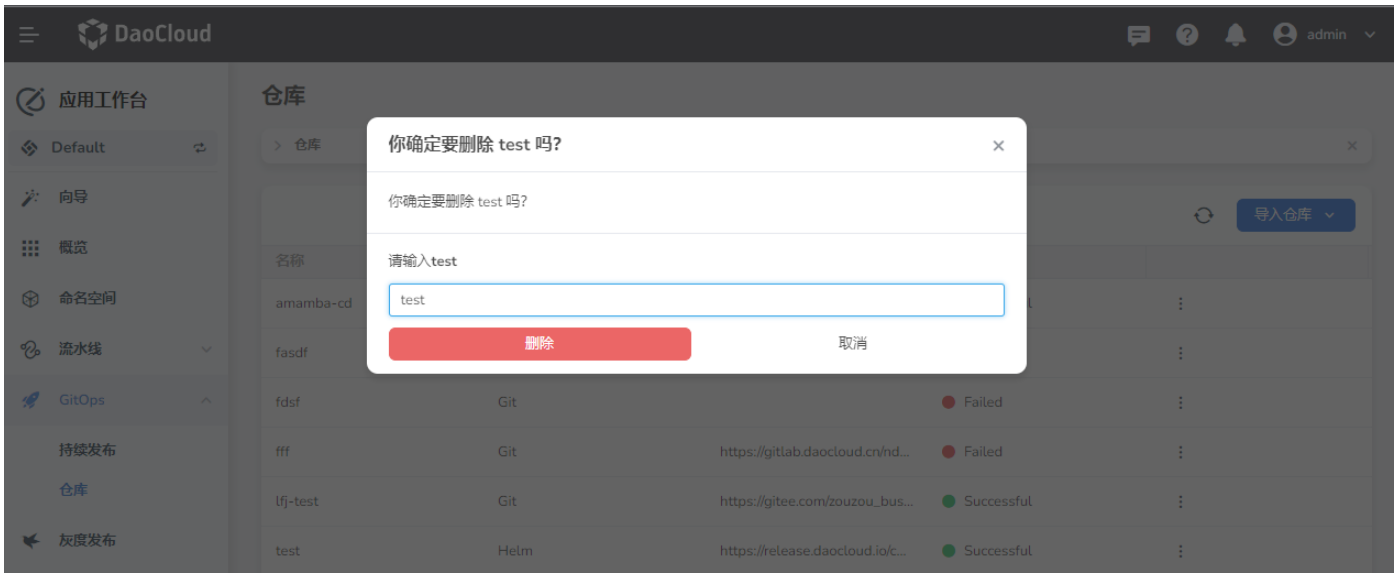
删除仓库

如果不再使用某个代码仓库，可参照以下步骤删除。

1. 在仓库列表页选择某一个仓库，点击 ⋮ ，在弹出的菜单中点击 删除 。



2. 在二次确认弹窗中点击 确定 。



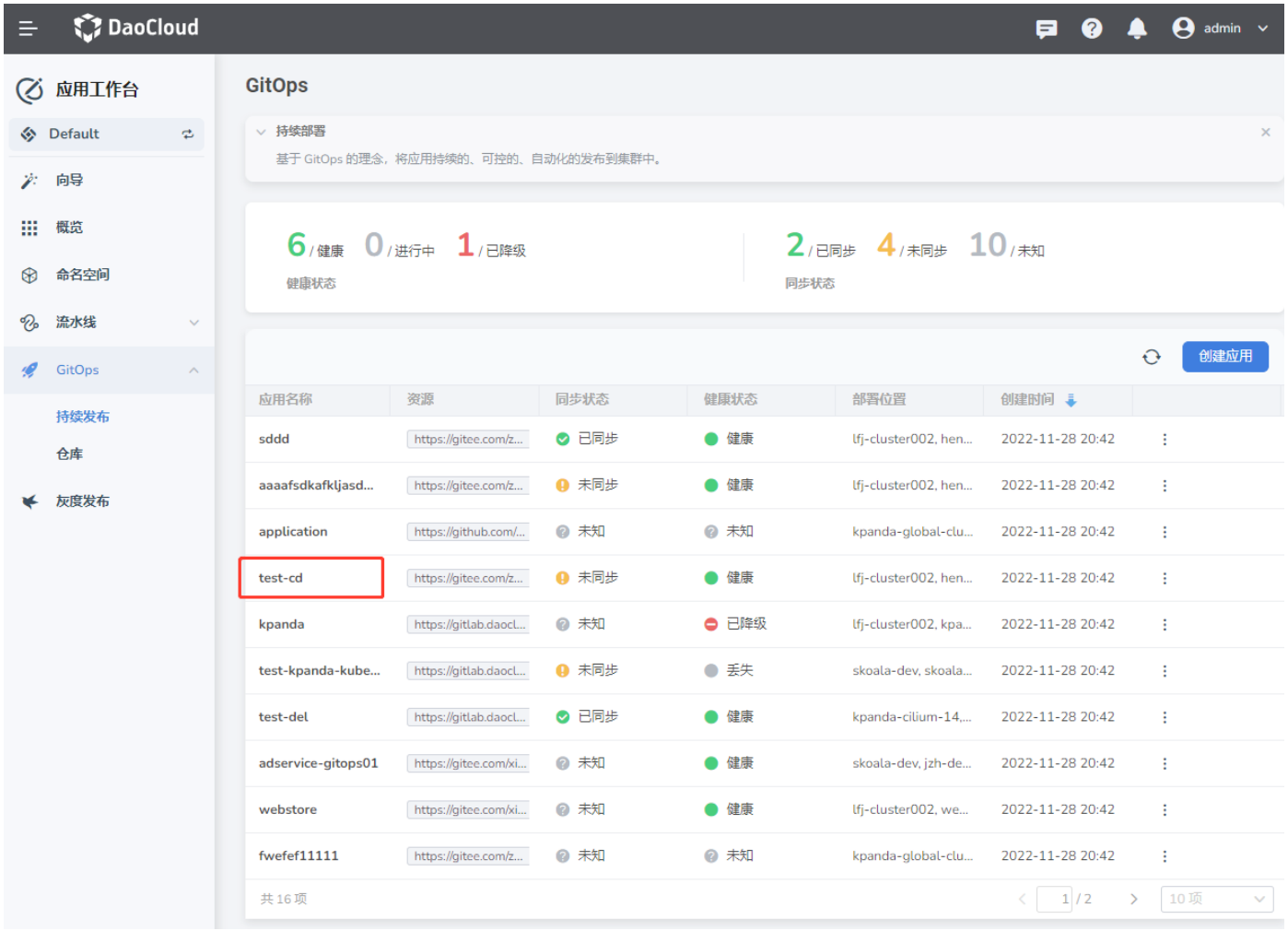
Copyright © 2016 - 2023 DaoCloud

手动同步应用

本页面演示如何手动同步持续部署的应用。

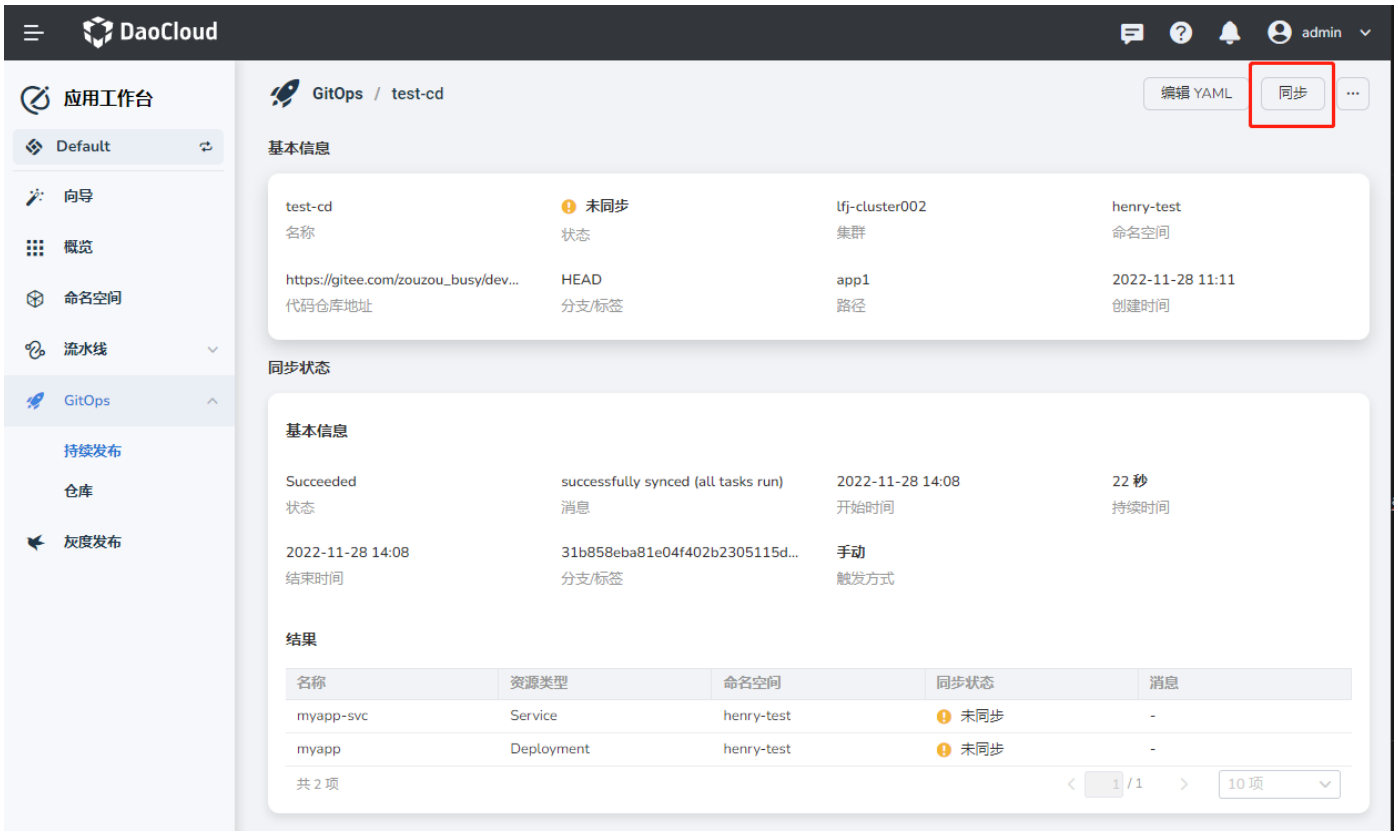
Copyright © 2016 - 2023 DaoCloud

1. 在 应用工作台 -> 持续发布 页面中，点击一个同步状态为 未同步 的应用名称。



2. 在应用详情页面，点击 同步 按钮，进入同步页面：

Copyright © 2016 - 2023 DaoCloud

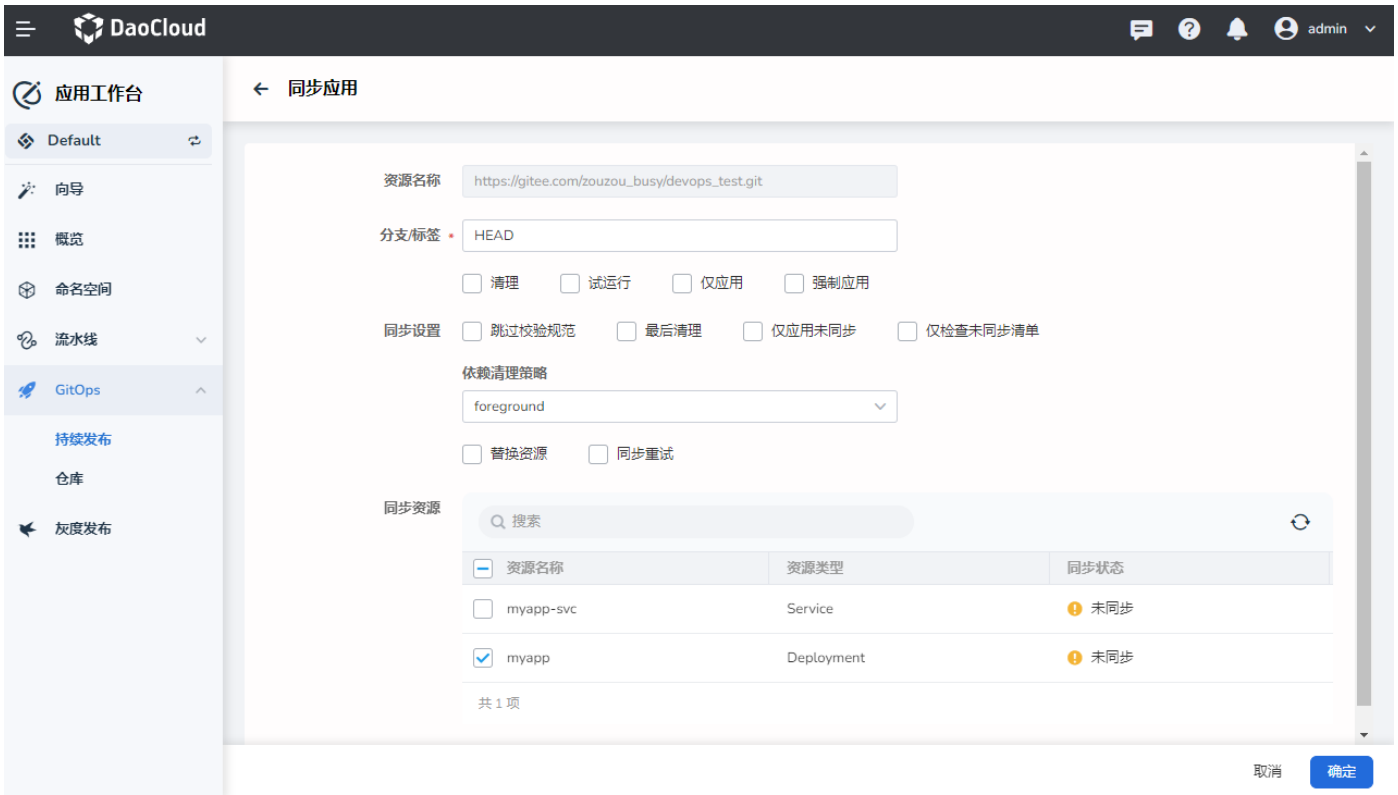


3. 在 同步应用 页面中，配置以下参数：

- 资源名称：只读状态，不支持二次编辑
- 分支/标签：设置同步的分支，如果同步方式为自动同步，更改后会同步报错
- 清理：勾选后，如果清单文件中删除了一些资源，相关资源在同步时也会被清理，反之相关资源则会被跳过清理
- 试运行：模拟同步过程
- 仅应用：勾选后，会通过 kubectl apply 的方式部署资源
- 强制应用：勾选后，如果部署资源时如果遇到冲突，会立即删除并重新创建该资源
- 同步设置：同步设置的参数与创建时一致，具体参数说明请参考创建应用

在 同步资源 区域，至少选择一个应用进行同步。例如一些资源已经被同步，只需要同步未同步的资源即可。

Copyright © 2016 - 2023 DaoCloud



4. 点击 确定 ，等待同步成功后，查看同步结果。

Copyright © 2016 - 2023 DaoCloud

5.2 Namespace

命名空间管理

命名空间是 Kubernetes 中用来隔离资源的一种抽象。

前提条件

- 当前工作空间已经拥有集群资源，参考绑定资源。
- 当前用户授权为 `Workspace Admin` 角色，详情参考工作空间最佳实践。

Copyright © 2016 - 2023 DaoCloud

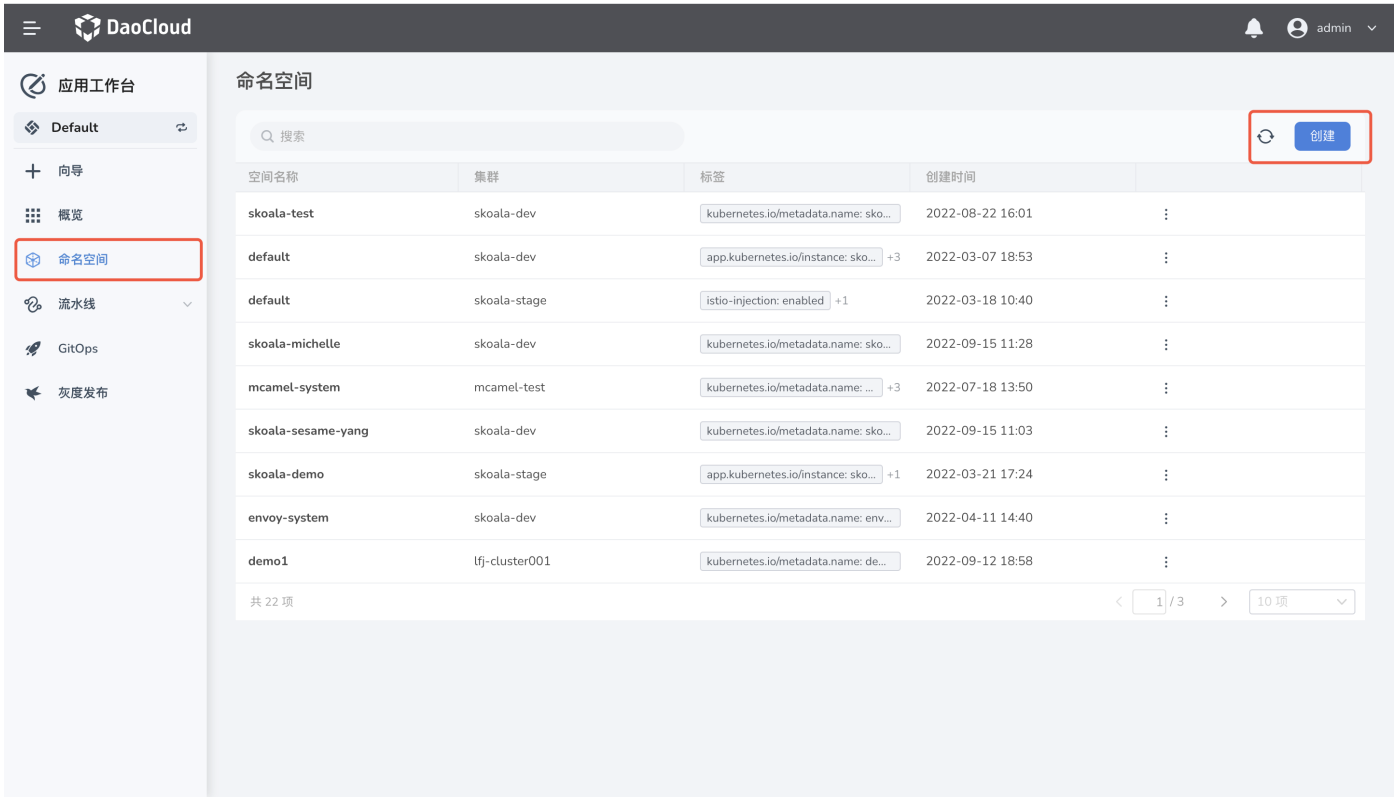
创建命名空间

!!! note

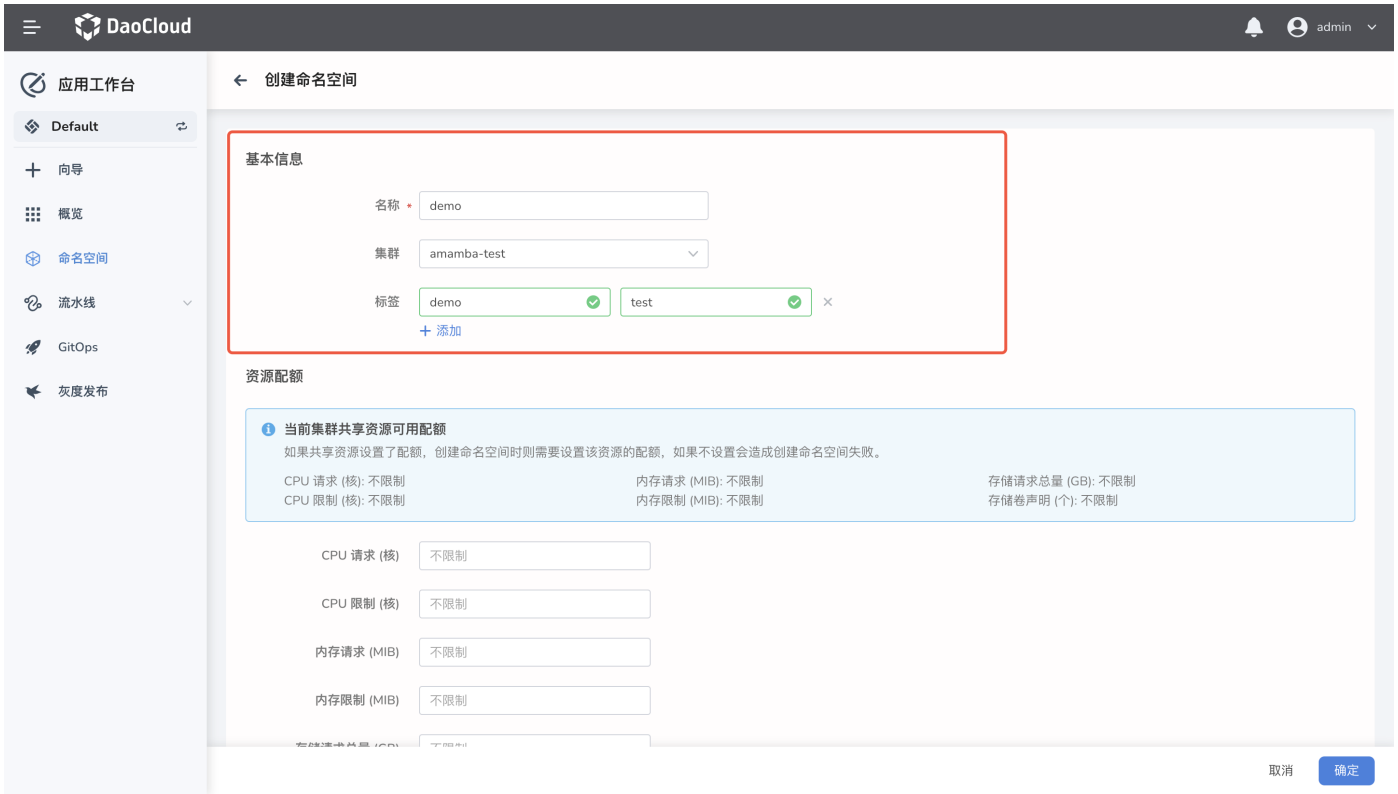
```
仅支持 `Workspace Admin` 创建命名空间，并设置命名空间配额，其他角色均不支持此操作。
```


Copyright © 2016 - 2023 DaoCloud

1. 在左侧导航栏中点击 命名空间 ，进入命名空间列表，点击右上角的 创建 。



2. 在 创建命名空间 页面中，配置命名空间的基本信息。



- 名称：设置命名空间的名称
- 集群：从当前工作空间下绑定的所有集群中选择一个集群
- 标签：设置命名空间的标签

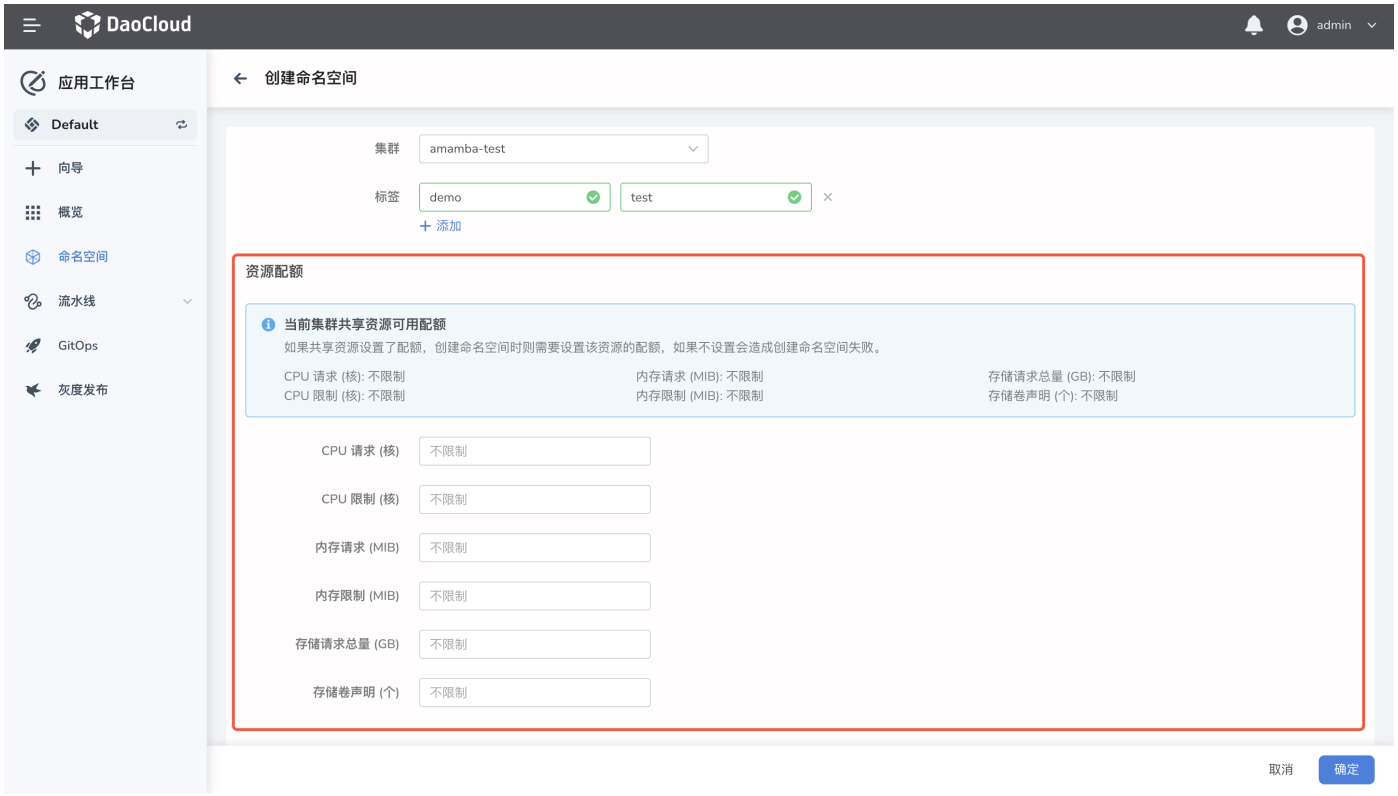
Copyright © 2016 - 2023 DaoCloud

3. 基本信息设置完成后，还需要配置命名空间的资源配额。

   !!! note

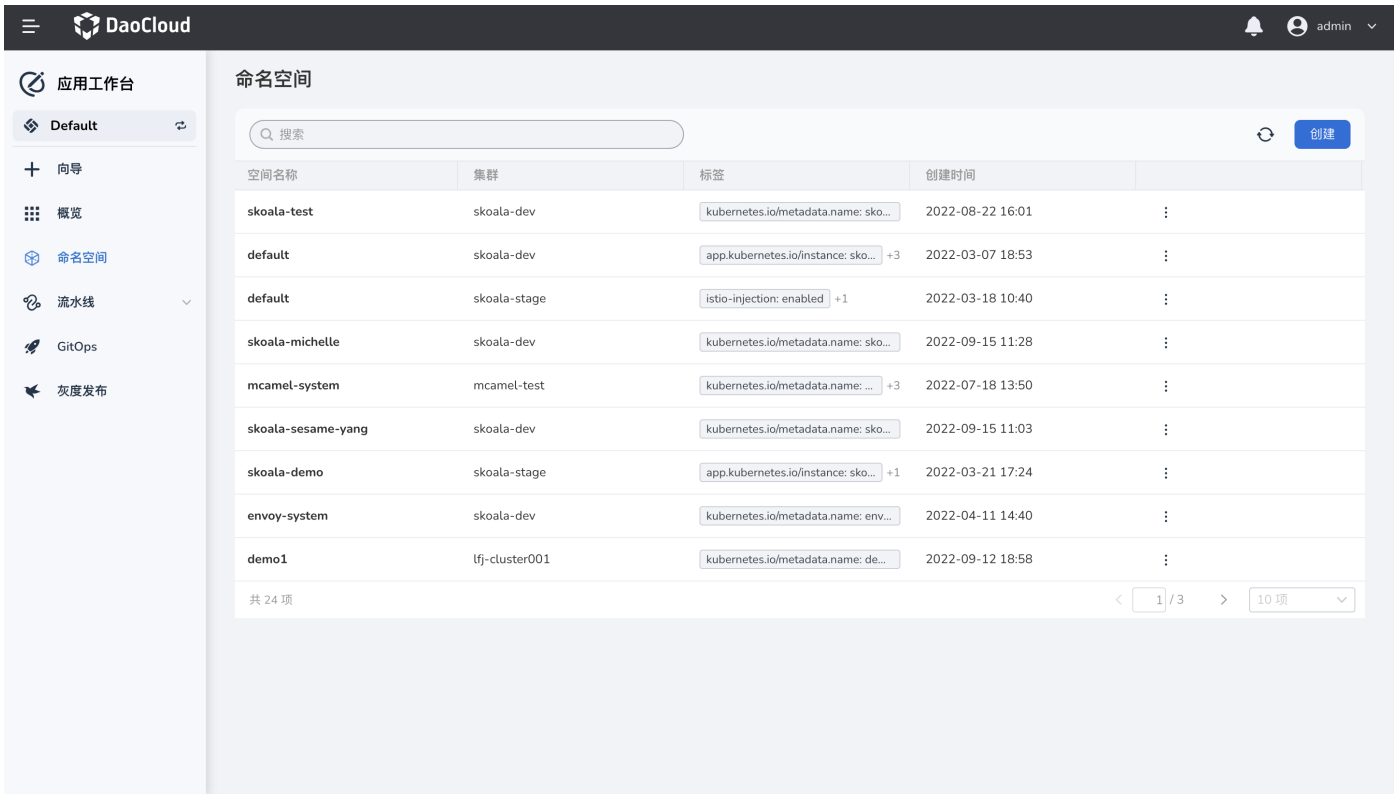
   - 如果当前集群在工作空间设置了任何资源配额，则必须要在创建命名空间时为命名空间设置此资源配额。
   - 设置命名空间的资源配额不能超过当前集群在该工作空间设置的资源使用上限。
   - 资源配额的请求或限制字段为空意味不为当前命名空间设置任何配额。

   目前工作空间中支持的资源配额有：CPU 请求、CPU 限制、内存请求、内存限制、存储请求总量、存储卷声明。其中 CPU、内存资源的限制值必须大于请求值。



4. 点击 确定 完成资源创建。屏幕提示创建成功，返回命名空间列表页面。

Copyright © 2016 - 2023 DaoCloud



5. 点击列表右侧的 ⋮，可以在弹出菜单中选择 配额管理 、 编辑标签 、 删除 等操作。

!!! warning

如果删除了某个命名空间，会删除该命名空间下所有的资源，请谨慎操作。

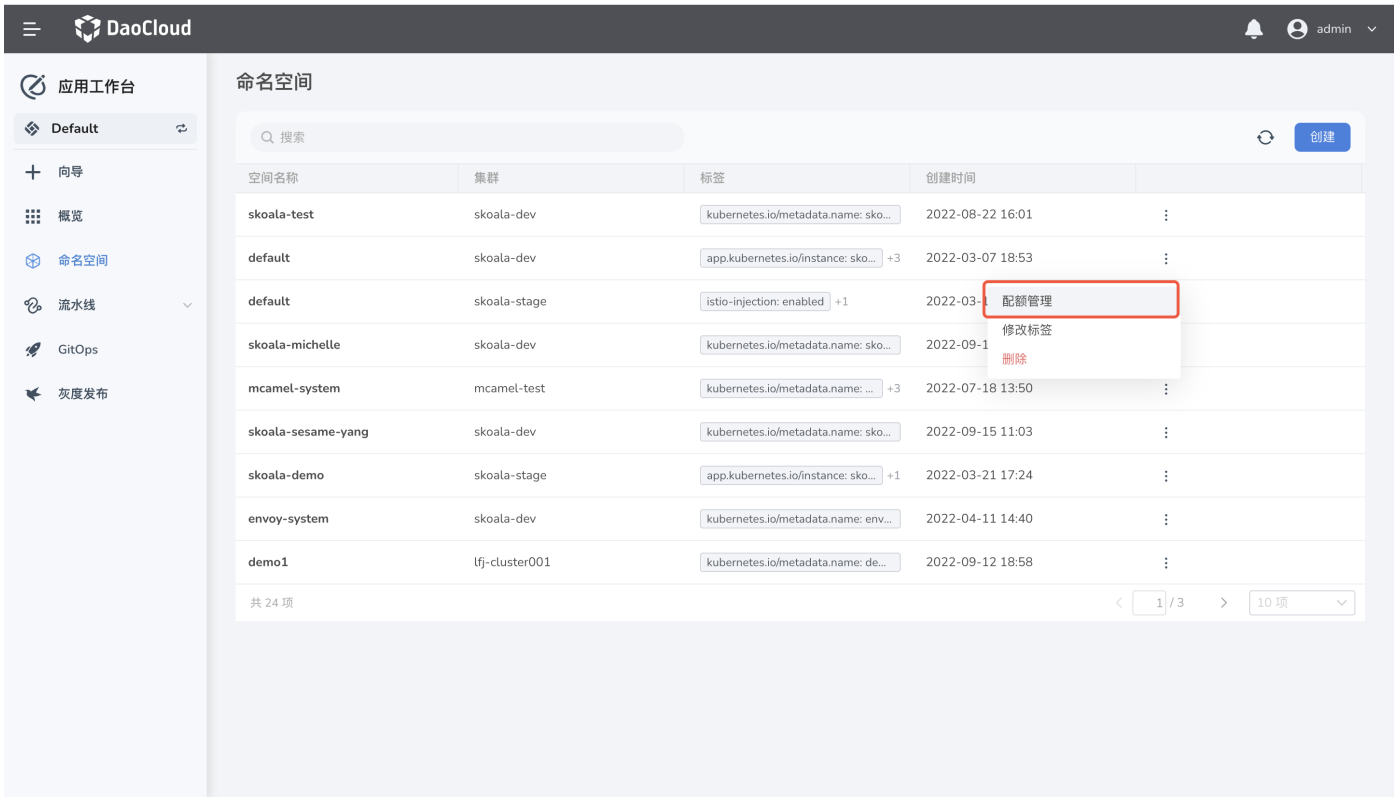
Copyright © 2016 - 2023 DaoCloud

命名空间配额

除了在创建命名空间中需要的 CPU 请求、CPU 限制、内存请求、内存限制、存储请求总量、存储卷声明资源配额，在命名空间配额管理中还可以设置其他资源配额，例如命名空间下的容器组、无状态负载、有状态负载、普通任务、定时任务等资源。

Copyright © 2016 - 2023 DaoCloud

1. 在命名空间列表页面，选择某一个命名空间，点击 配额管理 。



2. 在弹出的 资源配额 对话框中，可以看到目前命名空间的资源配额信息。在 应用资源 下点击 添加 ，选择一个资源并设置配额，具体请参阅 Kubernetes 资源配额。

3. 点击 确定 完成配额设置。

Copyright © 2016 - 2023 DaoCloud

5.3 Pipeline

创建自定义流水线

应用工作台流水线支持自定义创建，通过该方式创建的流水线，您可对流水线进行可视化编排。

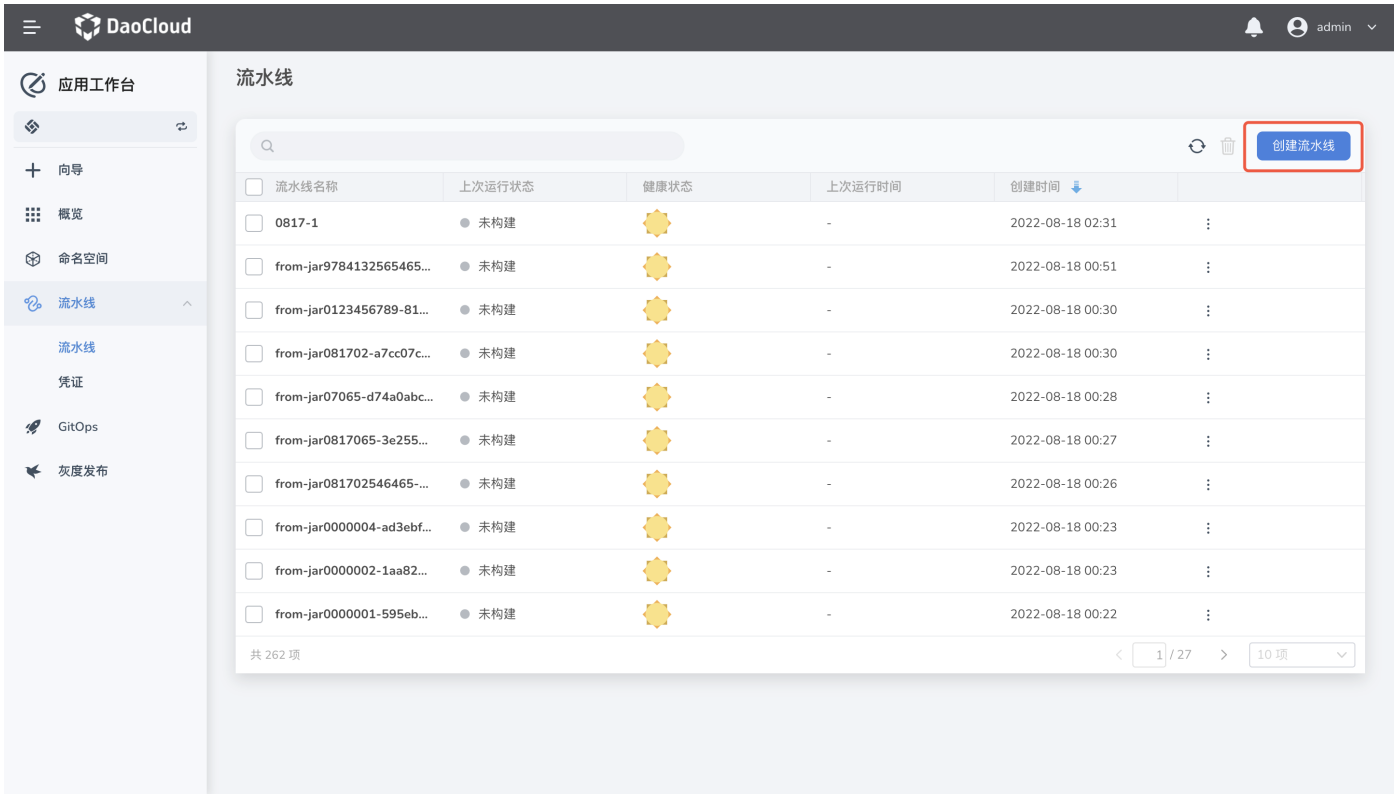
您需要创建一个工作空间和一个用户，必须邀请该用户至工作空间中且赋予 `workspace edit` 角色。 可参考创建工作空间、用户和角色。

Copyright © 2016 - 2023 DaoCloud

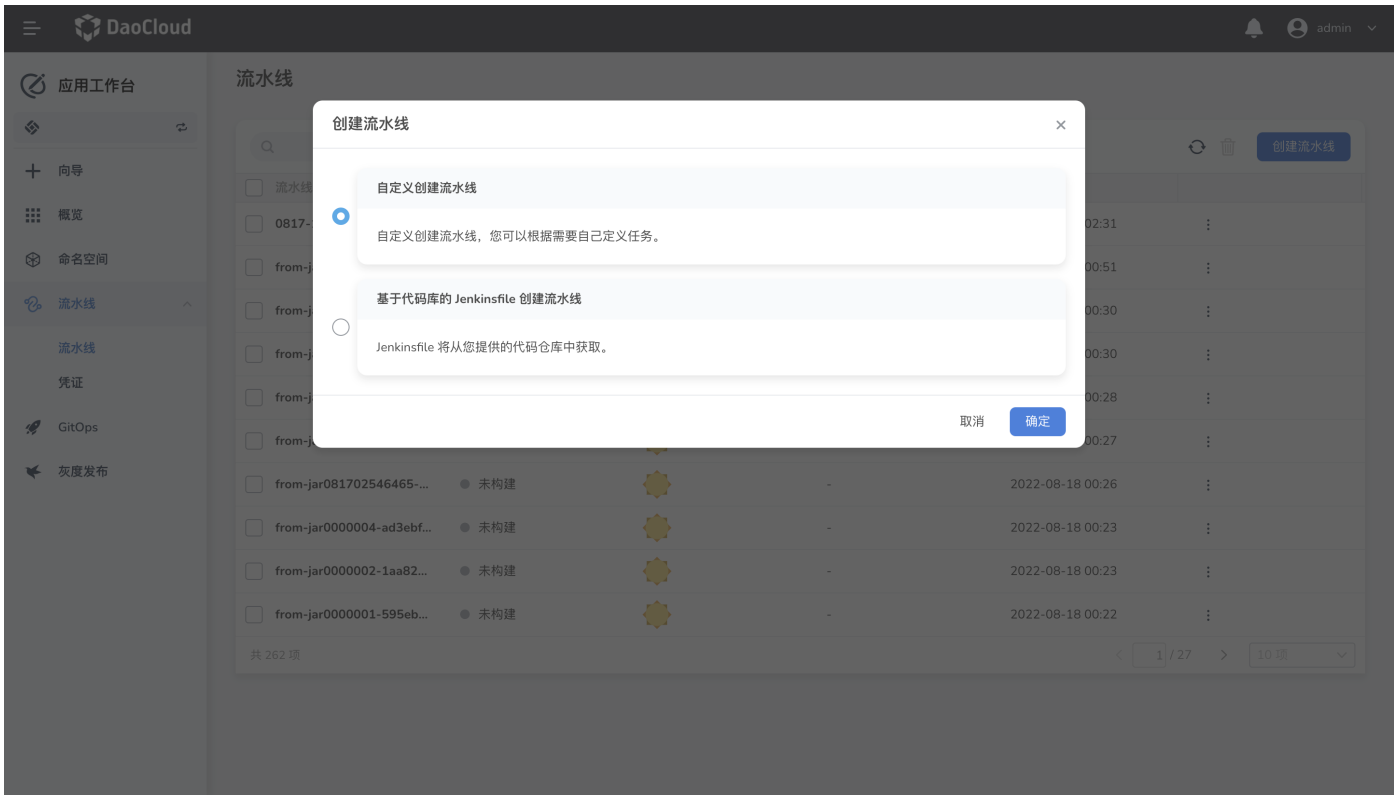
具体操作步骤如下：

Copyright © 2016 - 2023 DaoCloud

1. 在流水线列表页点击 创建流水线 。

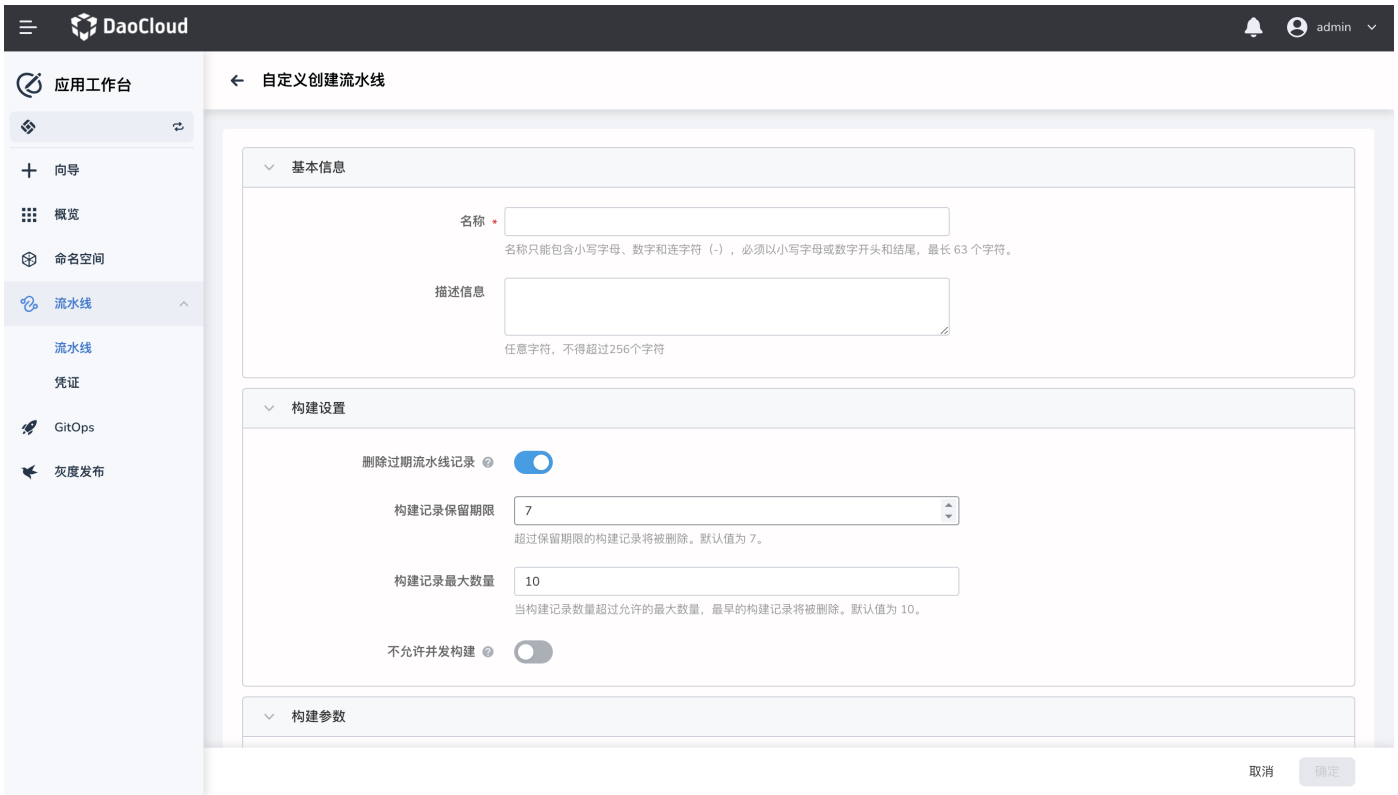


2. 在弹出的对话框中，选择 自定义创建流水线 ，点击 确定 。



3. 进入 自定义创建流水页面 ，并配置相关参数。

Copyright © 2016 - 2023 DaoCloud



4. 填写基本信息。流水线的名称，同一个工作空间下名称必须唯一。

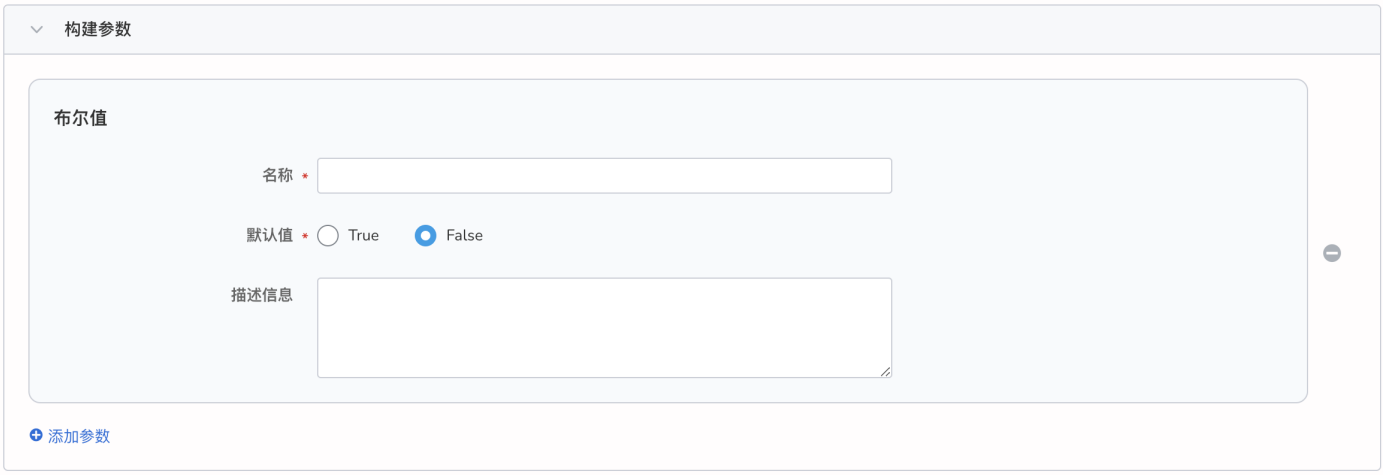
5. 填写构建设置。

- 删除过期构建记录：确定何时删除分支下的构建记录，以节省 Jenkins 所使用的磁盘空间。
- 不允许并发构建：如果开启，则不能并发运行多个构建。

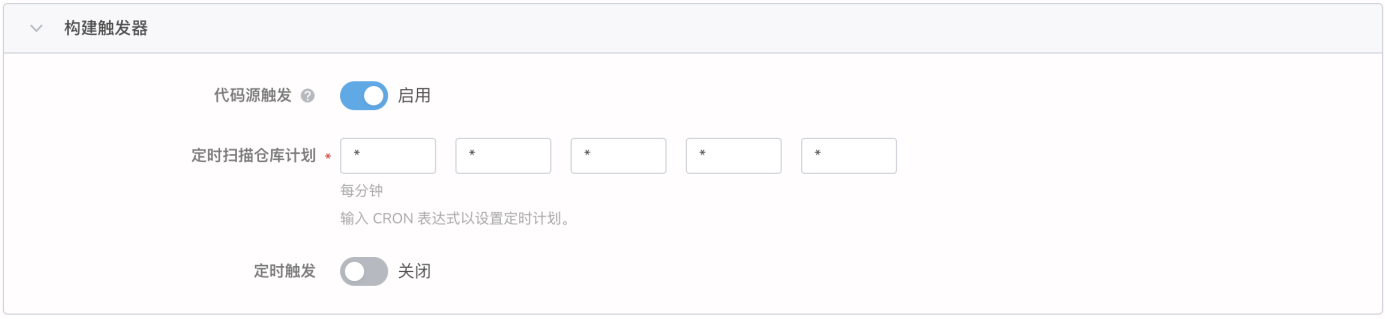
6. 填写构建参数。

   参数化的构建过程允许您在开始运行流水线时传入一个或多个参数。默认提供五种参数类型，包括 `字符串` 、 `多行字符串` 、 `布尔值` 、 `选项` 以及 `密码` 。 当参数化项目时，构建会被替换为参数化构建，其中将提示用户为每个定义的参数输入值。

Copyright © 2016 - 2023 DaoCloud

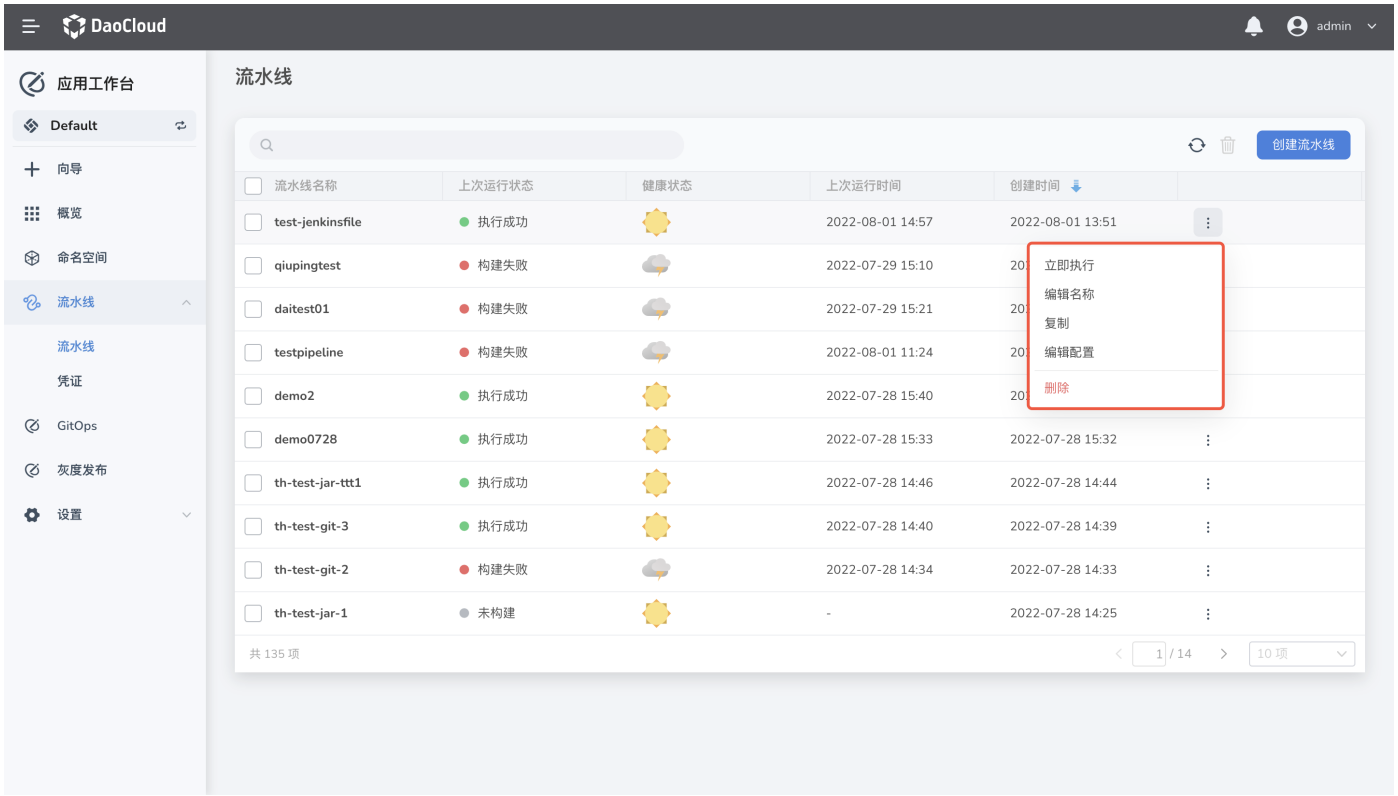


7. 填写构建触发器。



- 代码源触发：允许定期执行构建流水线。
- 定时触发：允许定期执行扫描远程代码仓库，如果代码仓库有变更，则执行构建流水线。

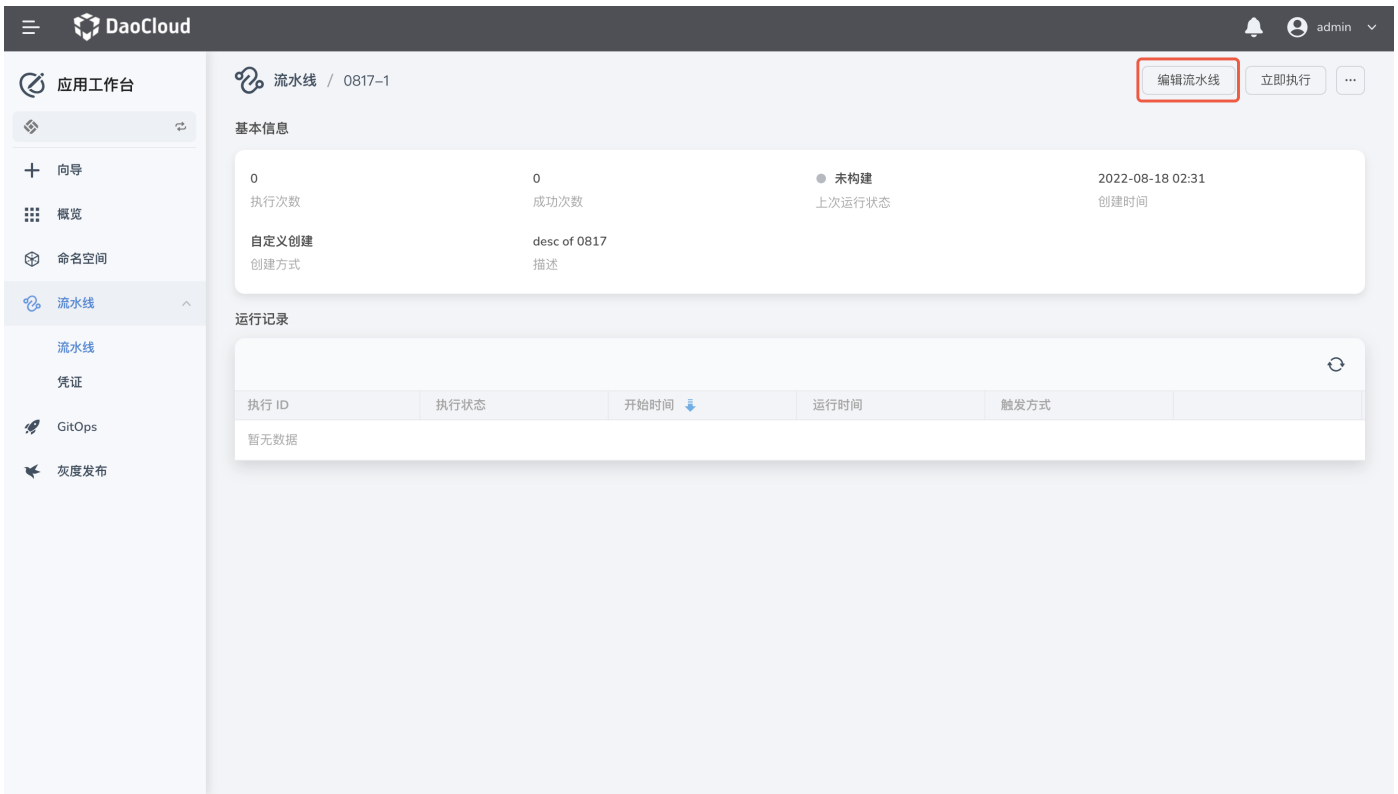
8. 完成创建。确认所有参数输入完成后，点击 确定 按钮，完成自定义流水线创建，自动返回流水线列表。点击列表右侧的 ⋮ 可以执行各项操作。



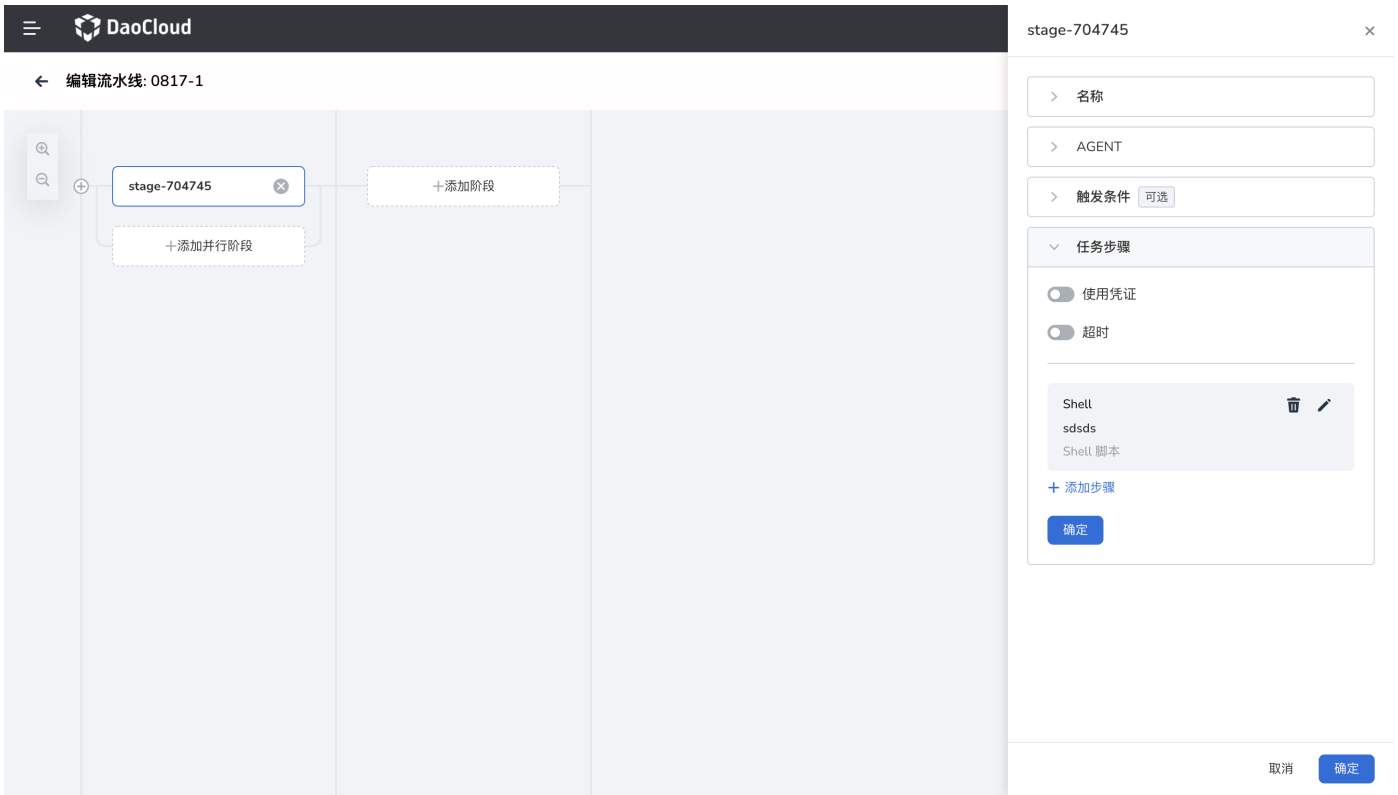
Copyright © 2016 - 2023 DaoCloud

编辑流水线

1. 自定义流水线创建完成后，您可在 流水线列表 页面点击该流水线进入 流水线详情 页面。然后点击 编辑流水线 。



2. 在编辑流水线页面您可以对该流水进行可视化编排，详细操作请参考文档使用图形化编辑流水线



Copyright © 2016 - 2023 DaoCloud

使用 Jenkinsfile 创建单代码分支流水线

应用工作台流水线支持使用代码仓库中的 Jenkinsfile 创建流水线。

前提条件

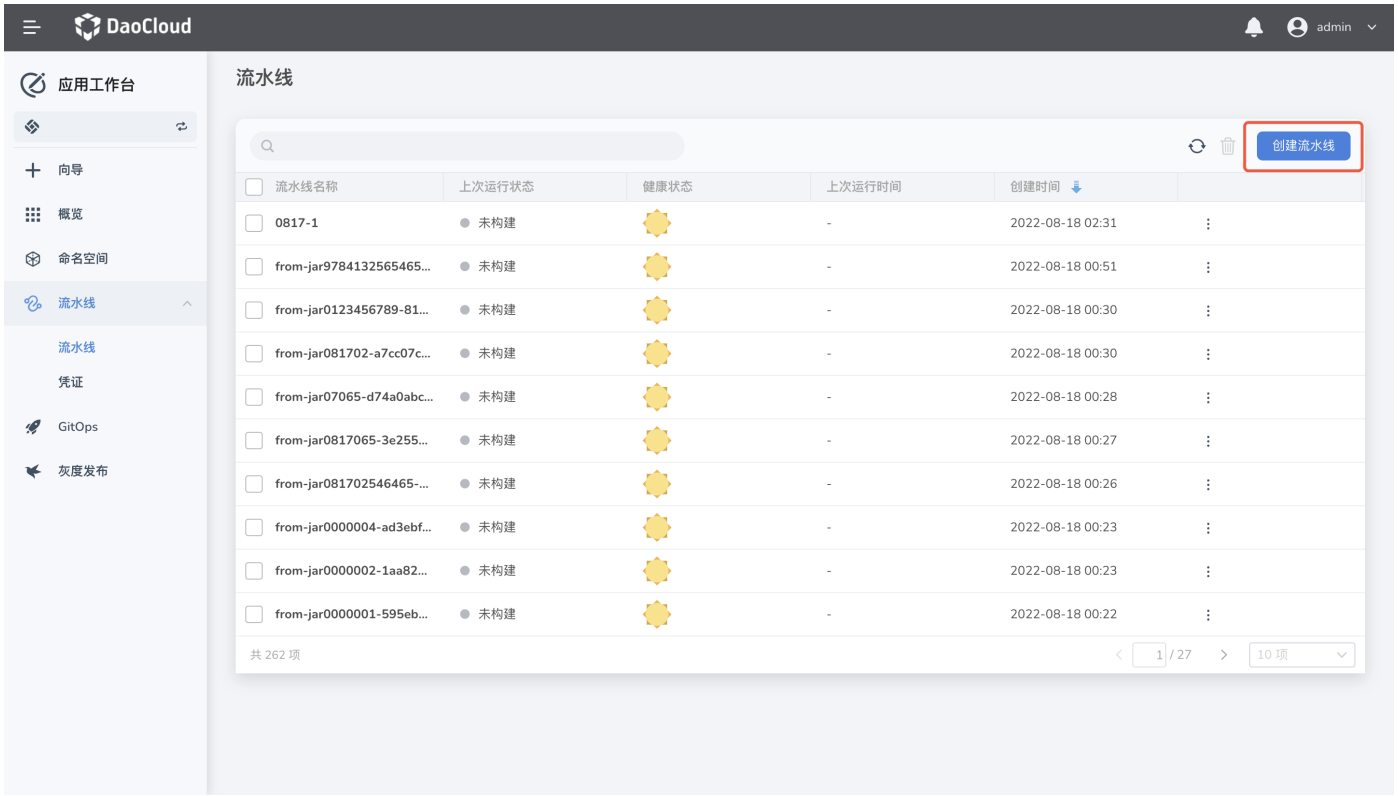
- 需要创建一个工作空间和一个用户，必须邀请该用户至工作空间中且赋予 `workspace edit` 角色。 参考创建工作空间、用户和角色。
- 需要有一个代码仓库，并且代码仓库中的源码有一个 Jenkinsfile 文本文件。

Copyright © 2016 - 2023 DaoCloud

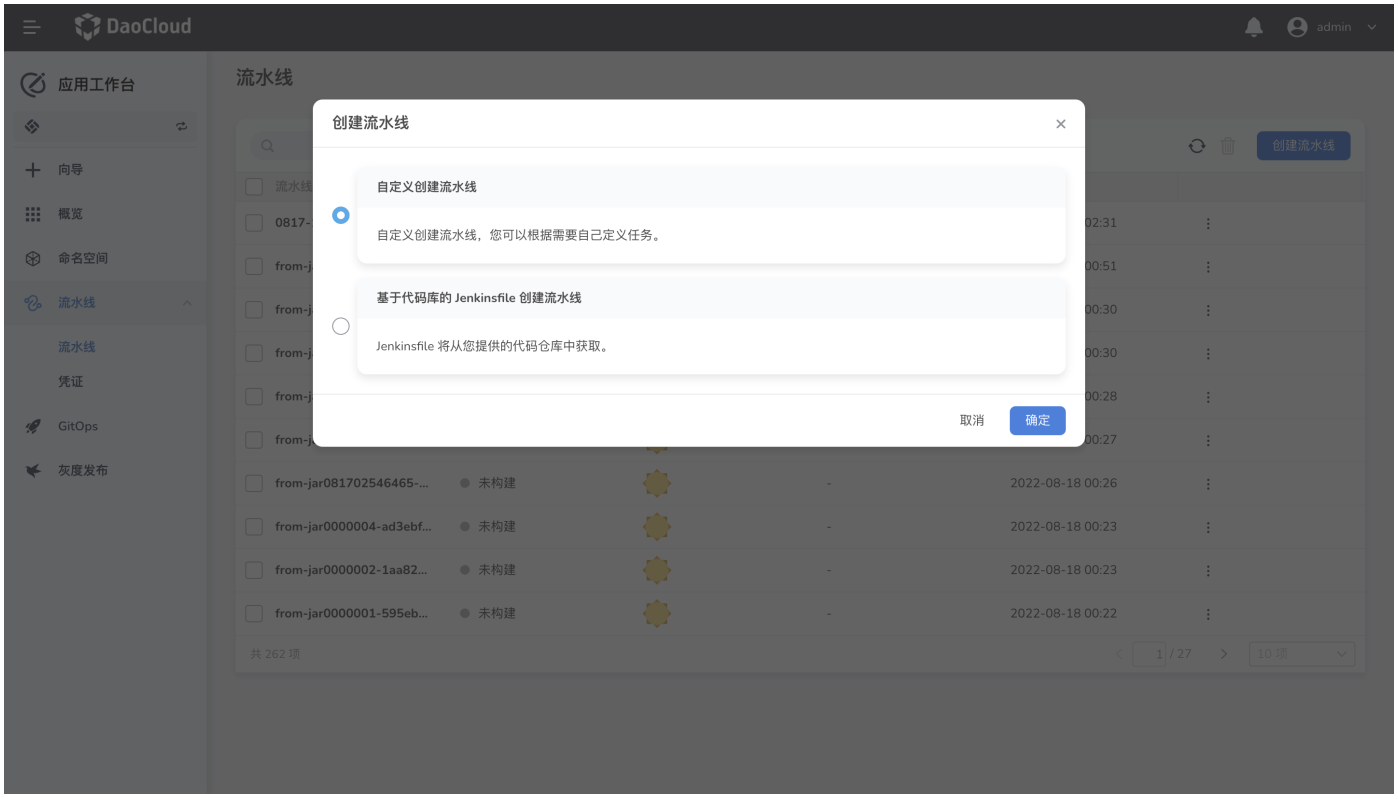
具体操作步骤如下：

Copyright © 2016 - 2023 DaoCloud

1. 在流水线列表页点击 `创建流水线`。

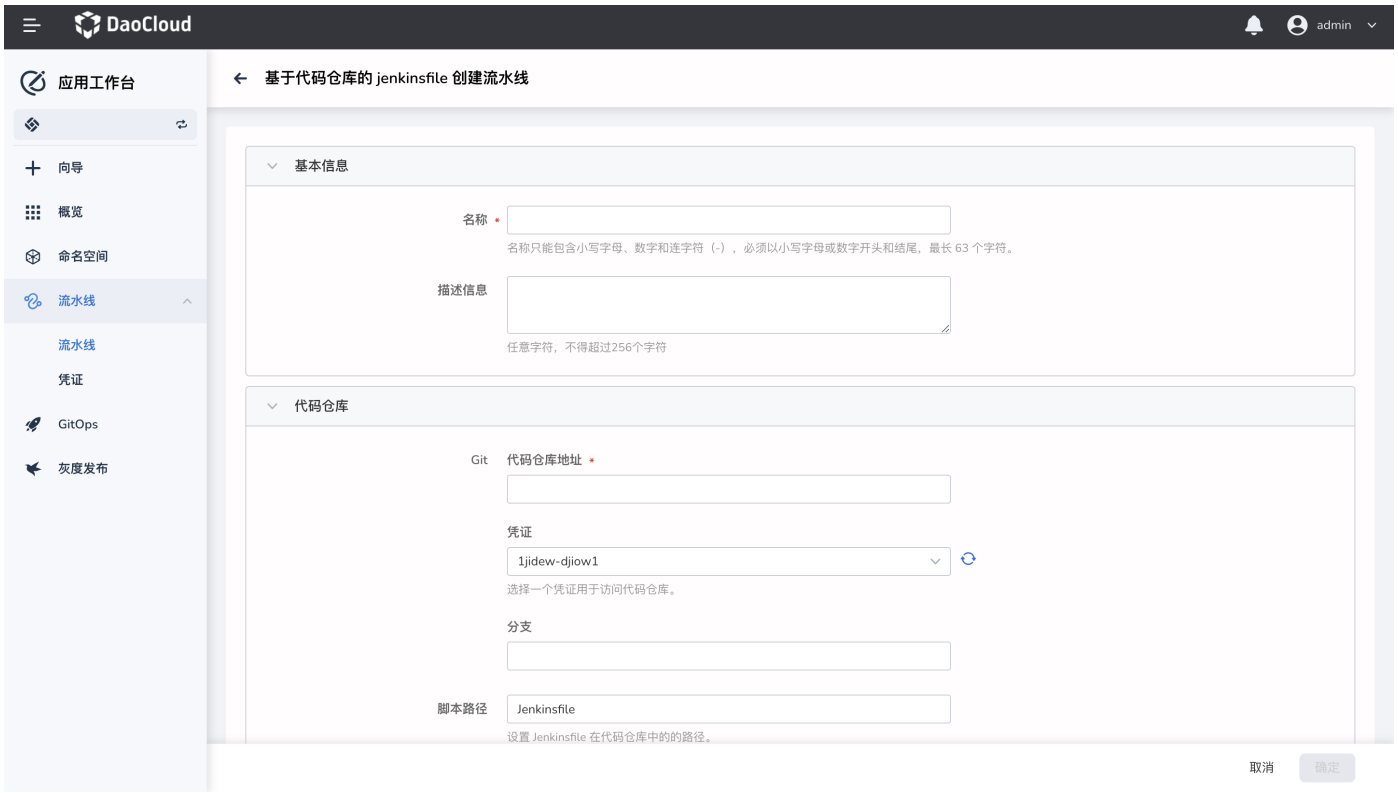


2. 在弹出的对话框中，选择 `自定义创建流水线`，点击 `确定`。



3. 进入 `基于代码仓库的 Jenkinsfile 创建流水线`，并配置相关参数。

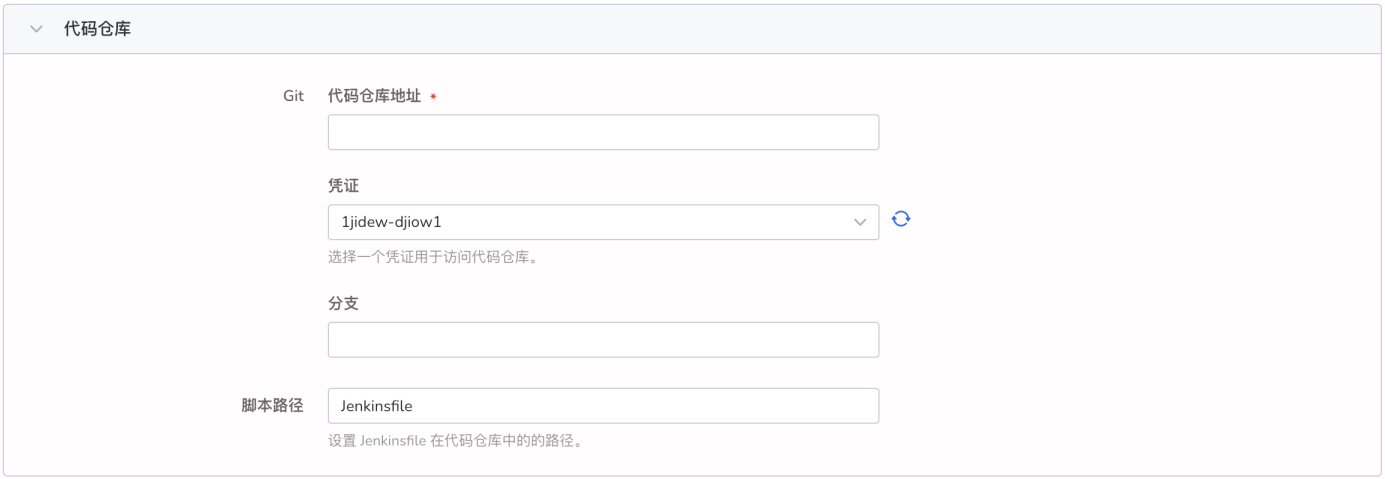
Copyright © 2016 - 2023 DaoCloud



4. 填写基本信息。流水线的名称，同一个工作空间下名称必须唯一。

5. 填写代码仓库信息。

Copyright © 2016 - 2023 DaoCloud

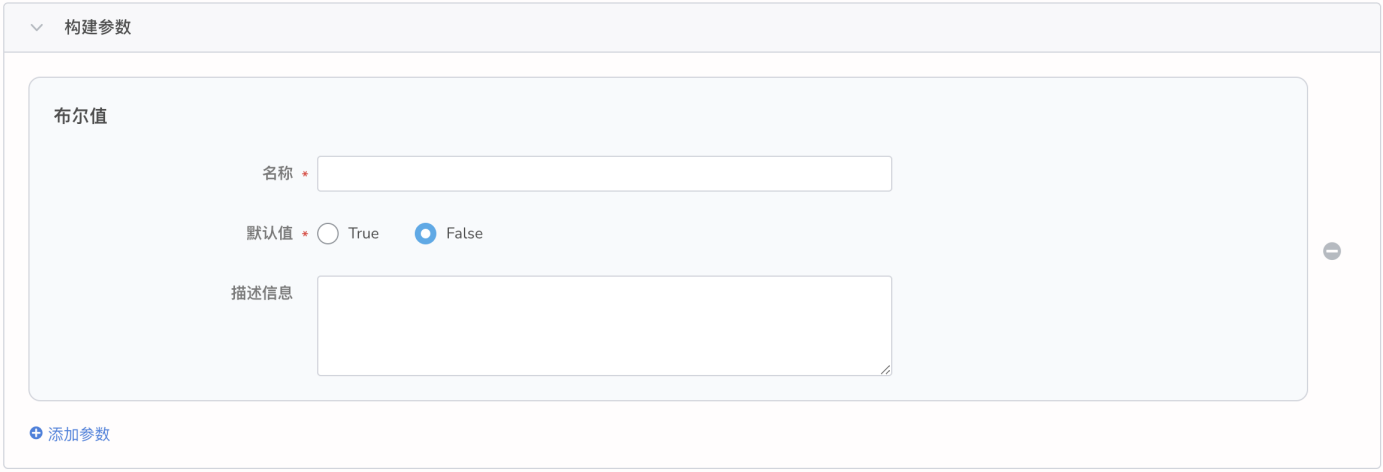


- 代码仓库地址：填写远程代码仓库的地址。
- 凭证：果您的远程仓库是私有的，请提前创建凭证并在使用时选择。
- 分支：填写您想要拉取的代码所在的分支，不填默认拉去 master 分支。
- 脚本路径：Jenkinsfile 文件在代码仓库中的绝对路径。

6. 填写构建设置。

- 删除过期构建记录：确定何时删除分支下的构建记录，以节省 Jenkins 所使用的磁盘空间。
- 不允许并发构建：如果开启，则不能并发运行多个构建。

7. 填写构建参数。参数化的构建过程允许您在开始运行流水线时传入一个或多个参数。 默认提供五种参数类型：`字符串`、`多行字符串`、`布尔值`、`选项` 以及 `密码`。 当参数化项目时，构建会被替换为参数化构建，其中将提示用户为每个定义的参数输入值。

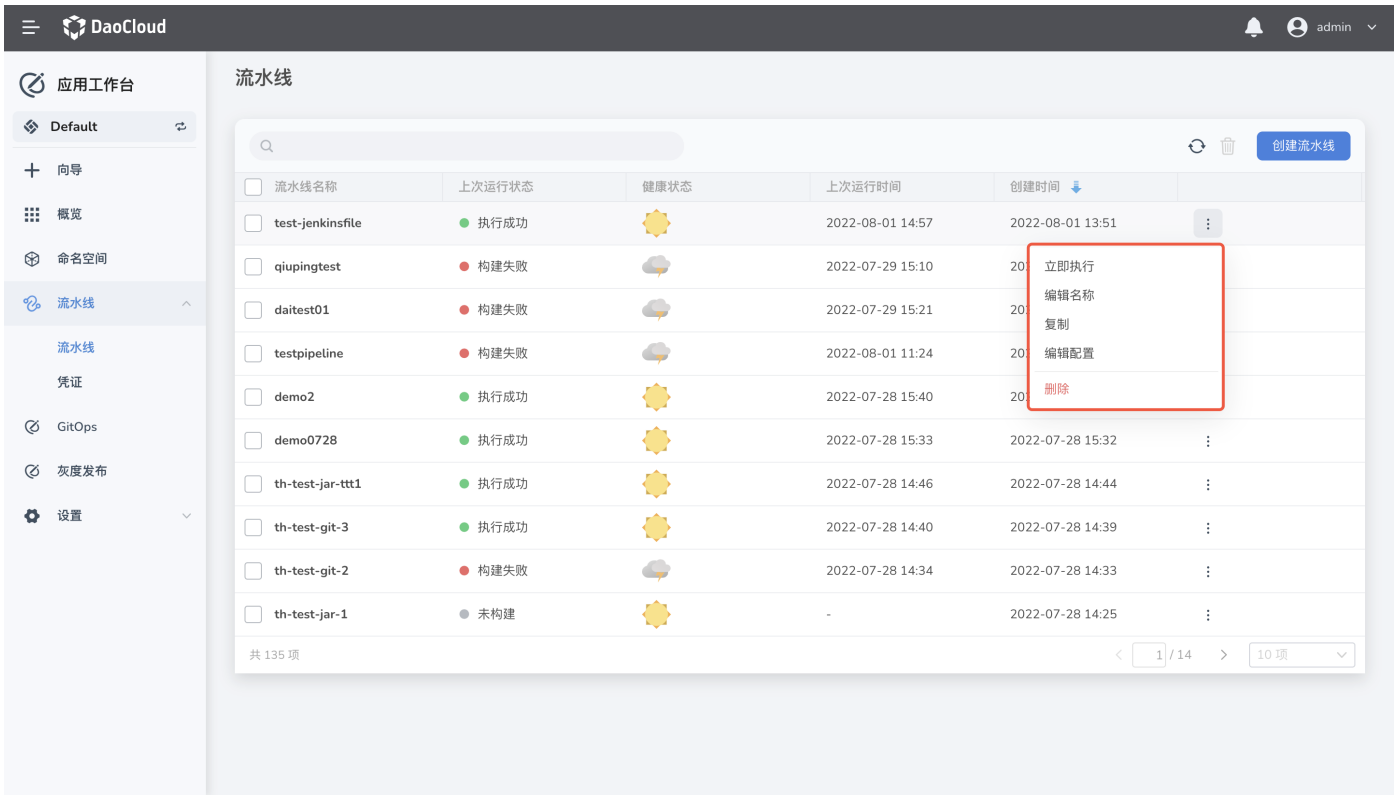


8. 填写构建触发器。

Copyright © 2016 - 2023 DaoCloud

- 代码源触发：允许定期执行构建流水线。
- 定时触发：允许定期执行扫描远程代码仓库，如果代码仓库有变更，则执行构建流水线。

9. 完成创建。确认所有参数输入完成后，点击 确定 按钮，完成自定义流水线创建，自动返回流水线列表。点击列表右侧的 ⋮ 可以执行各项操作。



Copyright © 2016 - 2023 DaoCloud

凭证管理

凭证可以保存敏感信息，例如用户名密码、访问令牌（Secret text）、Kubeconfig 等，使保存的数据更安全灵活并免于暴露到镜像中。 当流水线运行时，大多数情况下会与第三方网站或应用程序进行交互，如 Git 仓库、镜像仓库等。 此过程中需要提供相应的凭证，所以需要用户来为流水线配置凭证，用户配置凭证后，便可以利用这些凭证来与第三方网站或应用程序进行交互。

目前，您可以在应用工作台中创建以下 3 种类型的凭证：

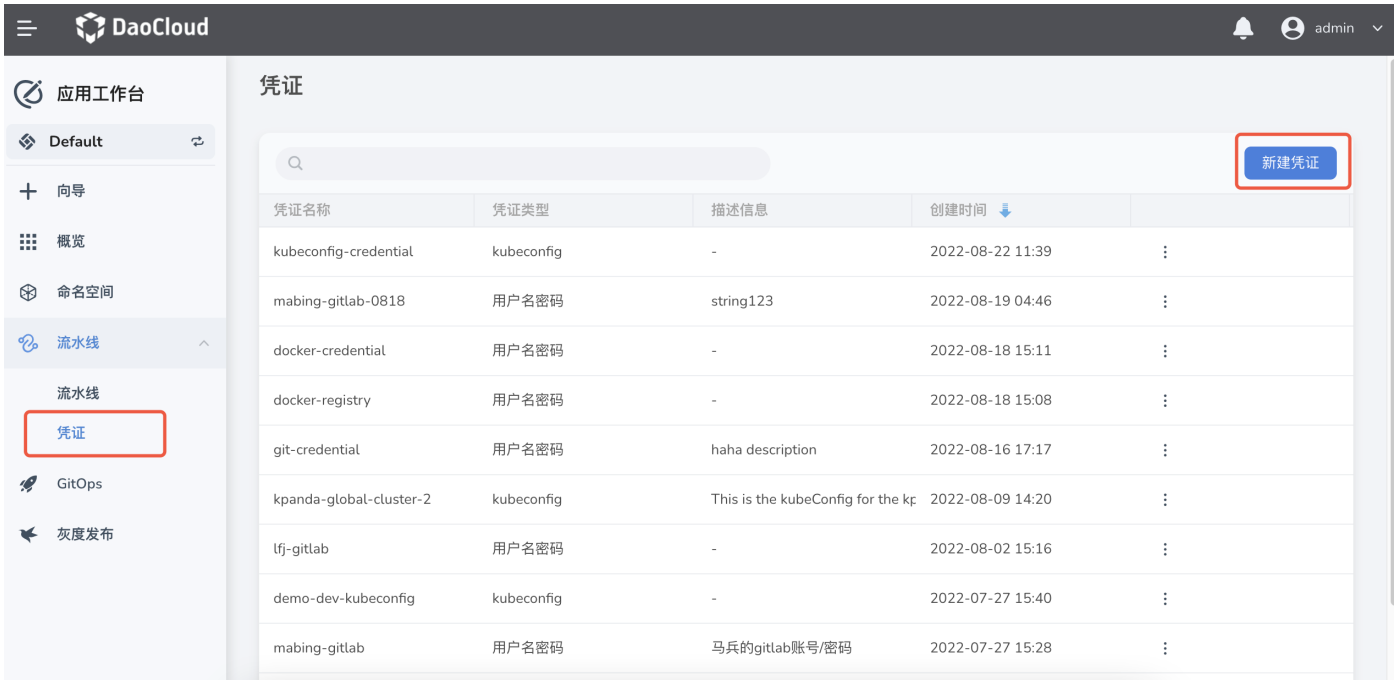
- 用户名和密码：用于存储用户名和密码的认证信息，若第三方网站或应用程序支持用户名/密码的方式访问，则可以选择这种类型，例如 GitHub、GitLab 和 Docker Hub 的帐户。
- 访问令牌（Secret text）：API token 之类的 token (如 GitHub 个人访问 token)。
- kubeconfig：用于配置跨集群认证。

Copyright © 2016 - 2023 DaoCloud

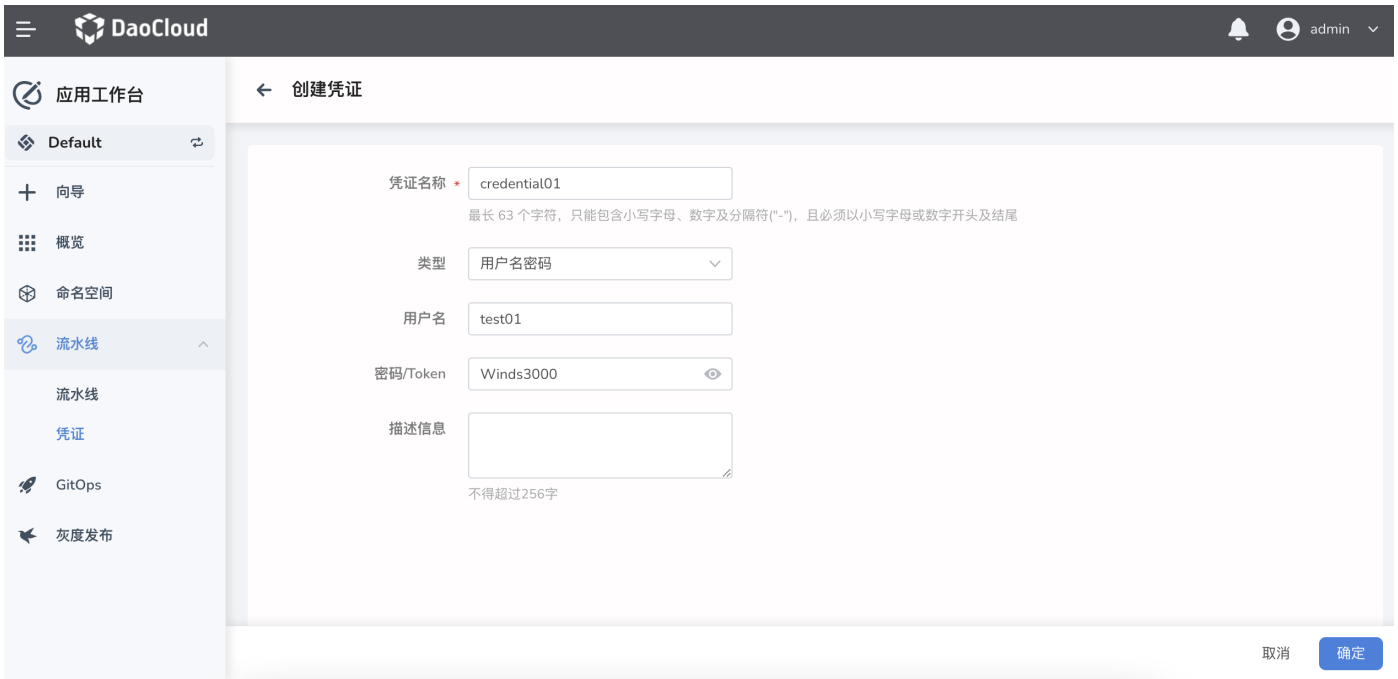
创建和管理凭证的具体步骤如下：

Copyright © 2016 - 2023 DaoCloud

1. 在左侧导航栏中点击 `流水线 -> 凭证`，进入凭证列表，点击右上角的 `新建凭证`。



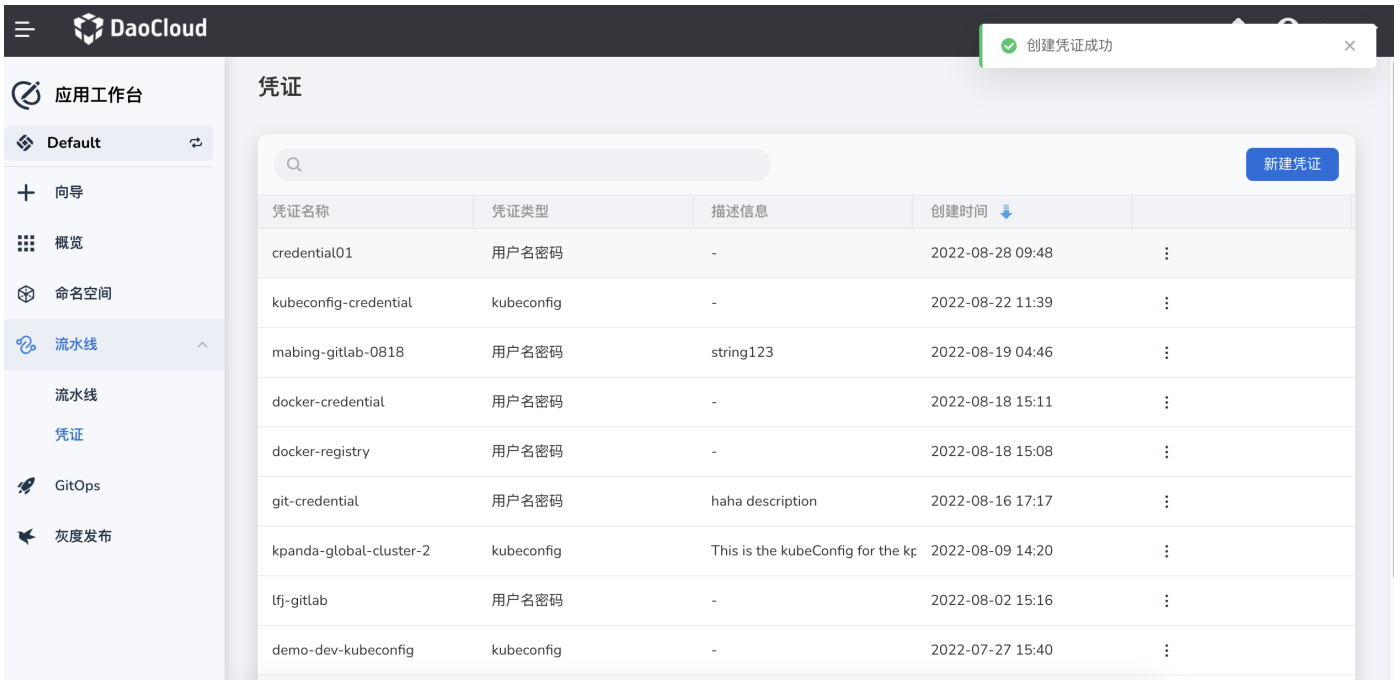
2. 在 `创建凭证` 页面中，配置相关参数后点击 `确定`。



- 填写 `凭证名称`，设置可以在流水线中使用的 ID，例如 `dockerhub-id`。注意：一旦设置就不能再进行更改。
- 在 `类型` 字段中，选择要添加的凭证类型。
- 根据不同的凭证类型填写相应的字段：
- 用户名和密码：在对应字段指定凭证的 `用户名` 和 `密码`。
- 访问令牌（Secret text）：复制加密文本到 `令牌` 字段中。
- Kubeconfig：复制集群证书到 `Kubeconfig` 字段中。

3. 屏幕提示创建成功，新创建的凭证默认位于第一个。

Copyright © 2016 - 2023 DaoCloud



4. 点击列表右侧的 `⋮`，可以在弹出菜单中选择编辑或删除等操作。

   !!! warning

   > 如果删除了某个流水线正使用的凭证，可能会影响流水线的访问，请谨慎操作。

Copyright © 2016 - 2023 DaoCloud

使用图形化编辑流水线

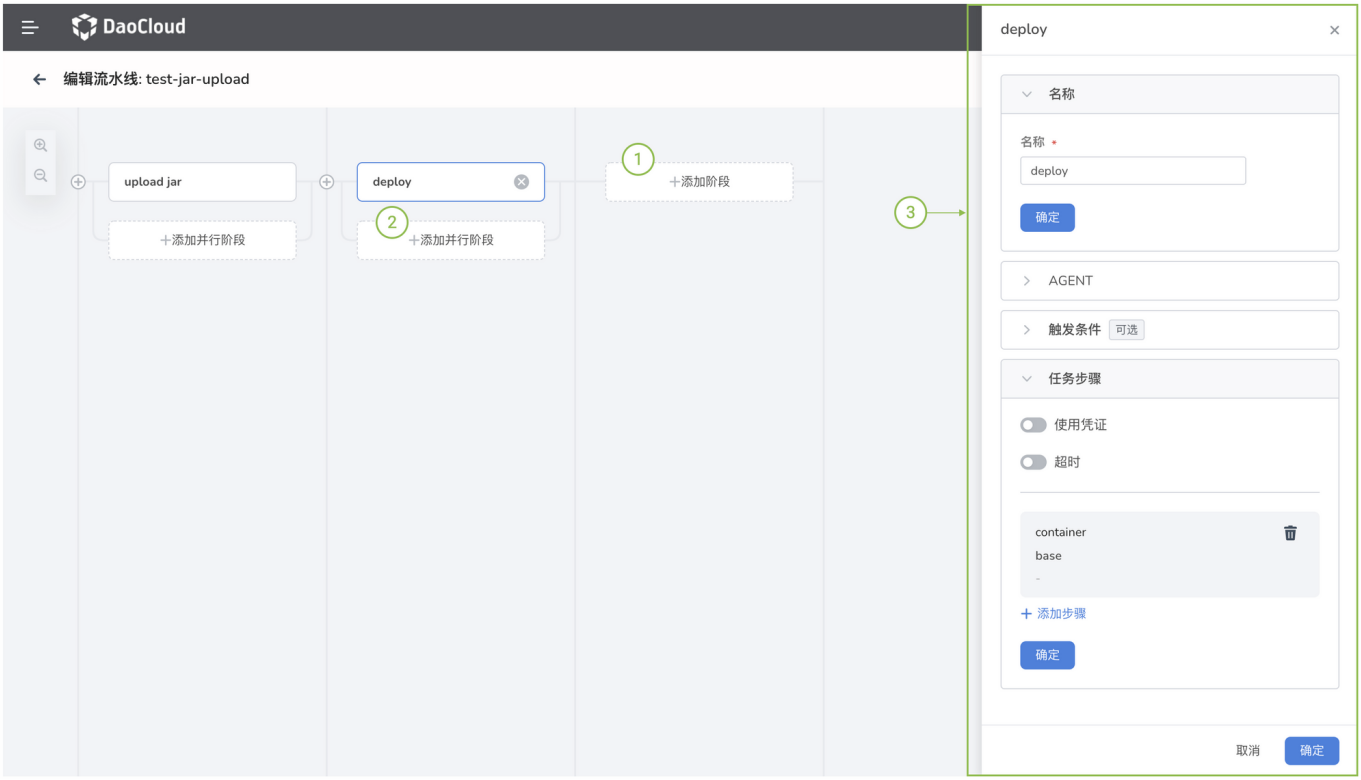
应用工作台设计了图形化流水线编辑视图，能够兼容编辑 Jenkinsfile 中的大部分自定义操作。 可以直观的让您可以看到流水线上的每个阶段(Stage)、每个步骤 (Step)，从而实现所见即所得的流水线编辑体验。

前提条件

- 您需要创建一个工作空间和一个用户，必须邀请该用户至工作空间中且赋予 `workspace edit` 角色。可参考创建工作空间、用户和角色。
- 创建可以访问代码仓库、镜像仓库、集群的三个凭证，创建凭证的更多信息，请参考凭证管理)。
- 创建一条自定义流水线，请参考创建自定义流水线，其中创建自定义流水线时需要在构建参数中添加两个字符串参数，这些参数将用于镜像构建的命令中。参数说明如下：

参数类型	参数名称	描述信息
字符串	registry	镜像仓库地址。本例中使用 `release-ci.daocloud.io`
字符串	project	镜像仓库中的项目名称。本例中使用 `demo`
字符串	name	镜像的名称。本例中使用 `http-hello`

画布说明



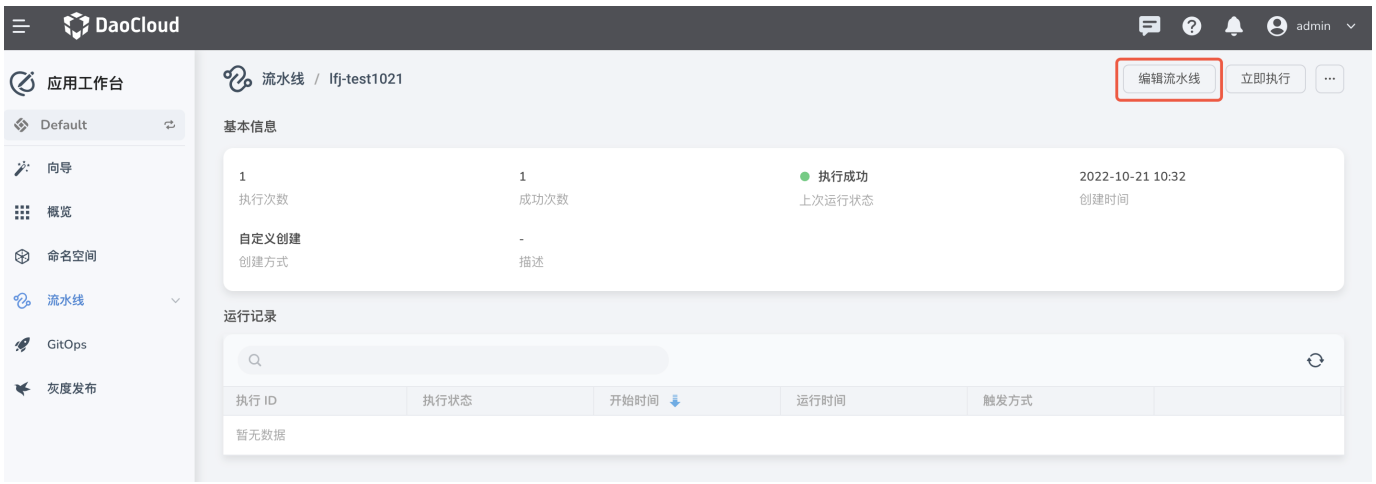
图形化编辑页面包括两个区域：画布 和 阶段设置。

1. 点击 `+添加阶段` 后，会生成一个串行阶段，点击后，会生成一个新的阶段，可以在该阶段中添加流水线步骤，通过选择步骤类型，可以快速创建当前阶段中的流水线步骤。
2. 点击 `+添加并行阶段` 后，会生成一个并行阶段，点击后，会生成一个新的阶段，可以在该阶段中添加流水线步骤，通过选择步骤类型，可以快速创建当前阶段中的流水线步骤。
3. 阶段创建后，点击 `阶段`，可以用过阶段进行阶段配置。支持设置阶段的名称、Agent、触发条件、步骤。

Copyright © 2016 - 2023 DaoCloud

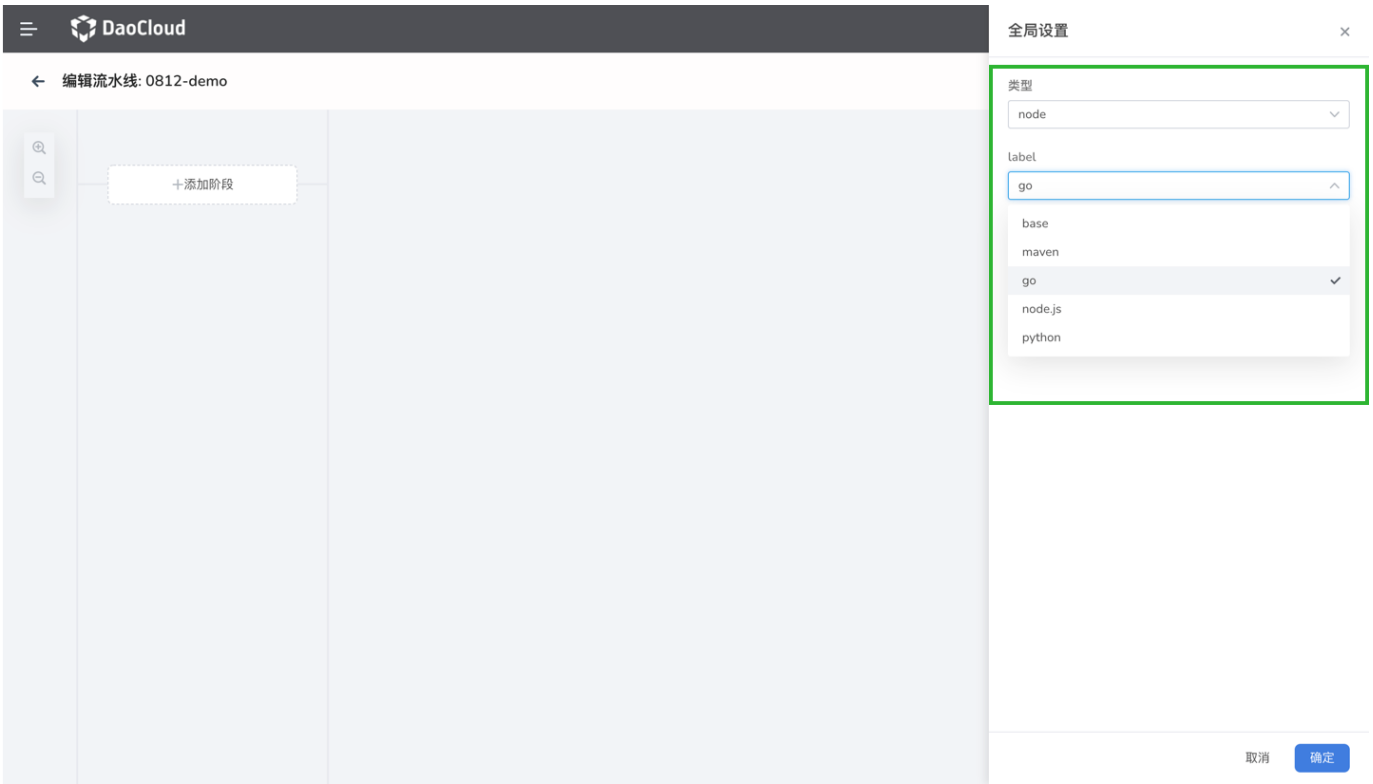
操作步骤

点击一个创建好的自定义流水线。在右上角点击 编辑流水线 ，进入图形化编辑页面。



配置全局设置

点击 全局设置 ，从 类型 下拉列表中选择 node，从 label 下拉列表选择 go 16。

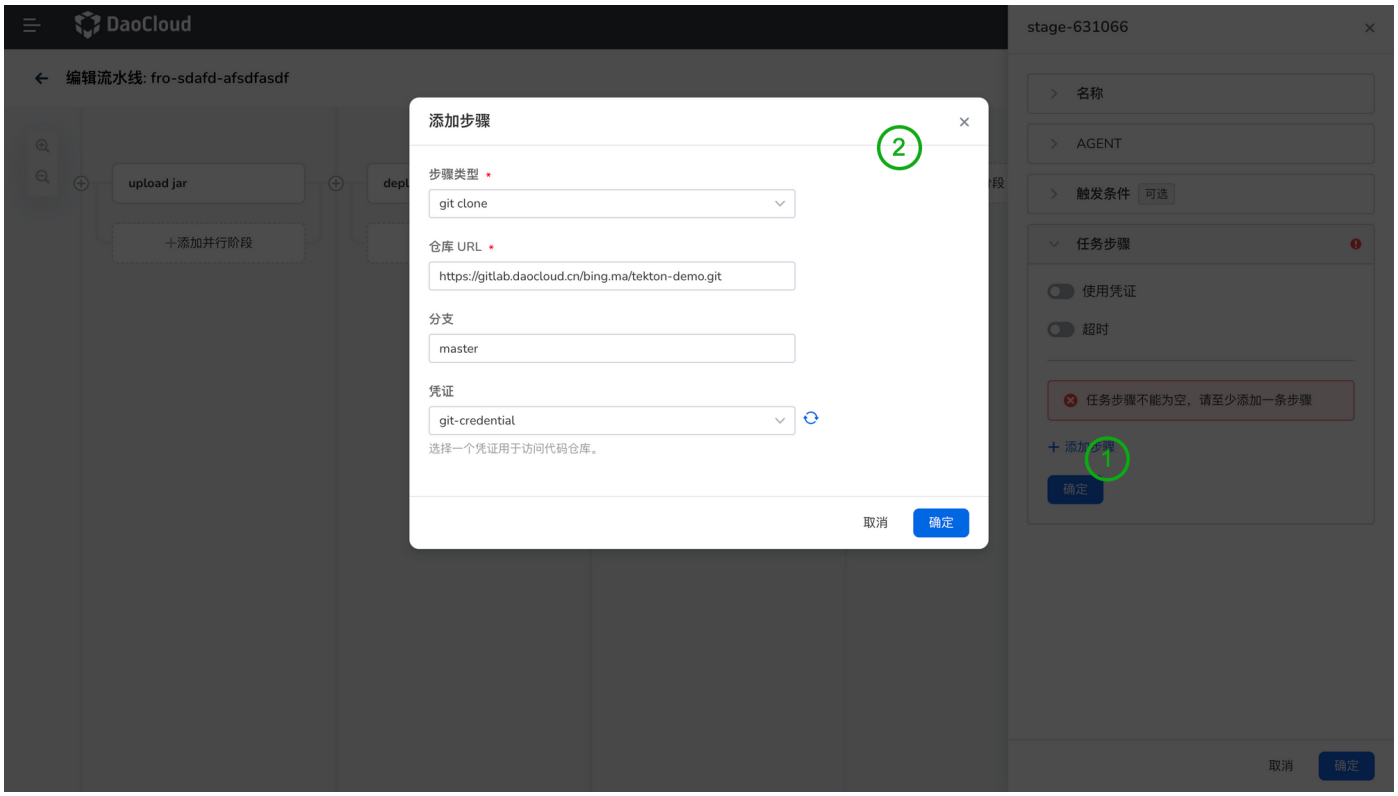


添加阶段 - 拉取源代码

1. 点击画布中的 添加阶段 。在右侧的阶段设置中设置名称：git clone。
2. 点击 添加步骤 ，在弹出的对话框中步骤类型下选择 `git clone` ，参照下表，对相关参数进行配置。

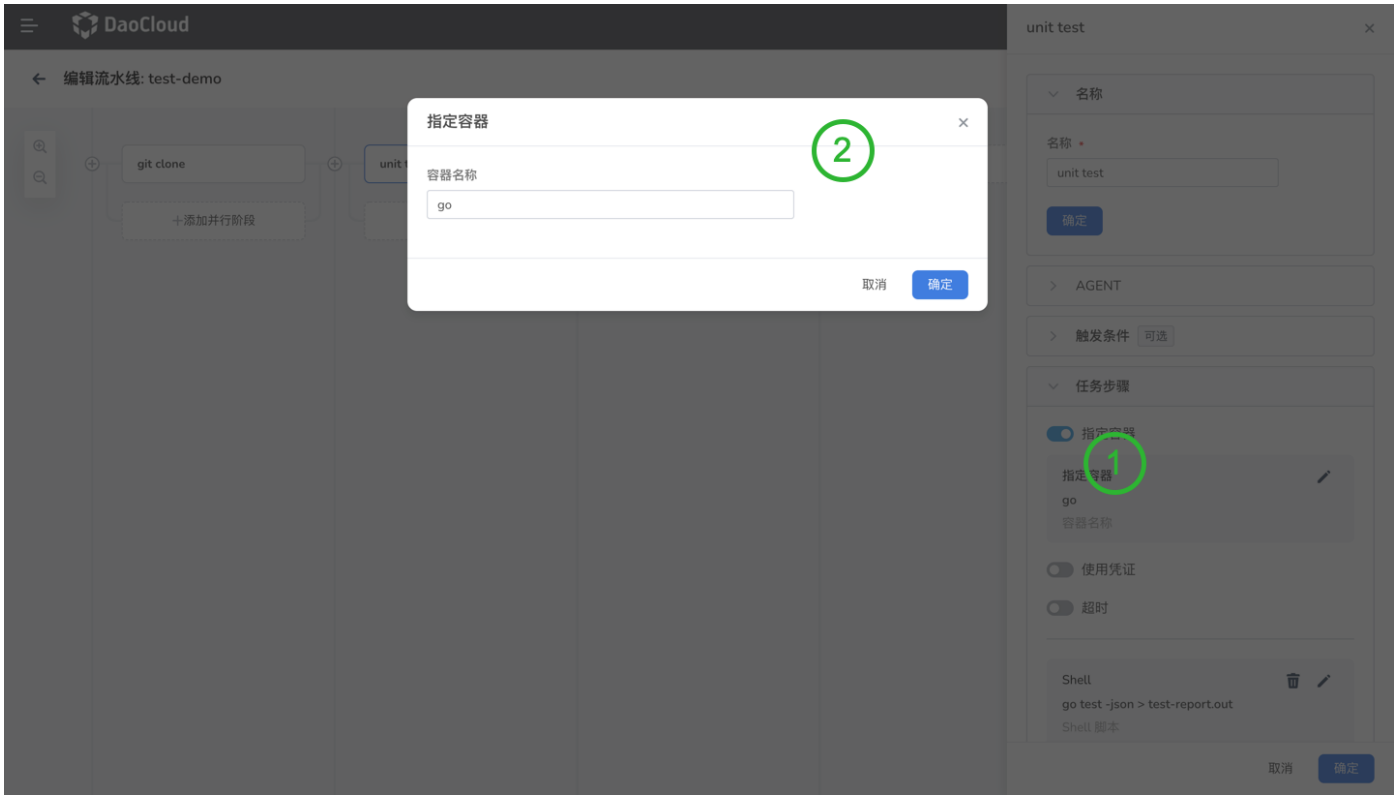
- 仓库 URL：输入仓库地址。
- 分支：不填写，默认为m aster 分支。
- 凭证：如果您的仓库属于私有仓库则需要提供一个凭证。

Copyright © 2016 - 2023 DaoCloud



添加阶段 - 单元测试

1. 点击画布中的 添加阶段 。在右侧的阶段设置中设置名称：unit test。
2. 在步骤模块中选择开启 指定容器 ，在弹出的对话框中填写容器名称：go，然后点击 确定 。

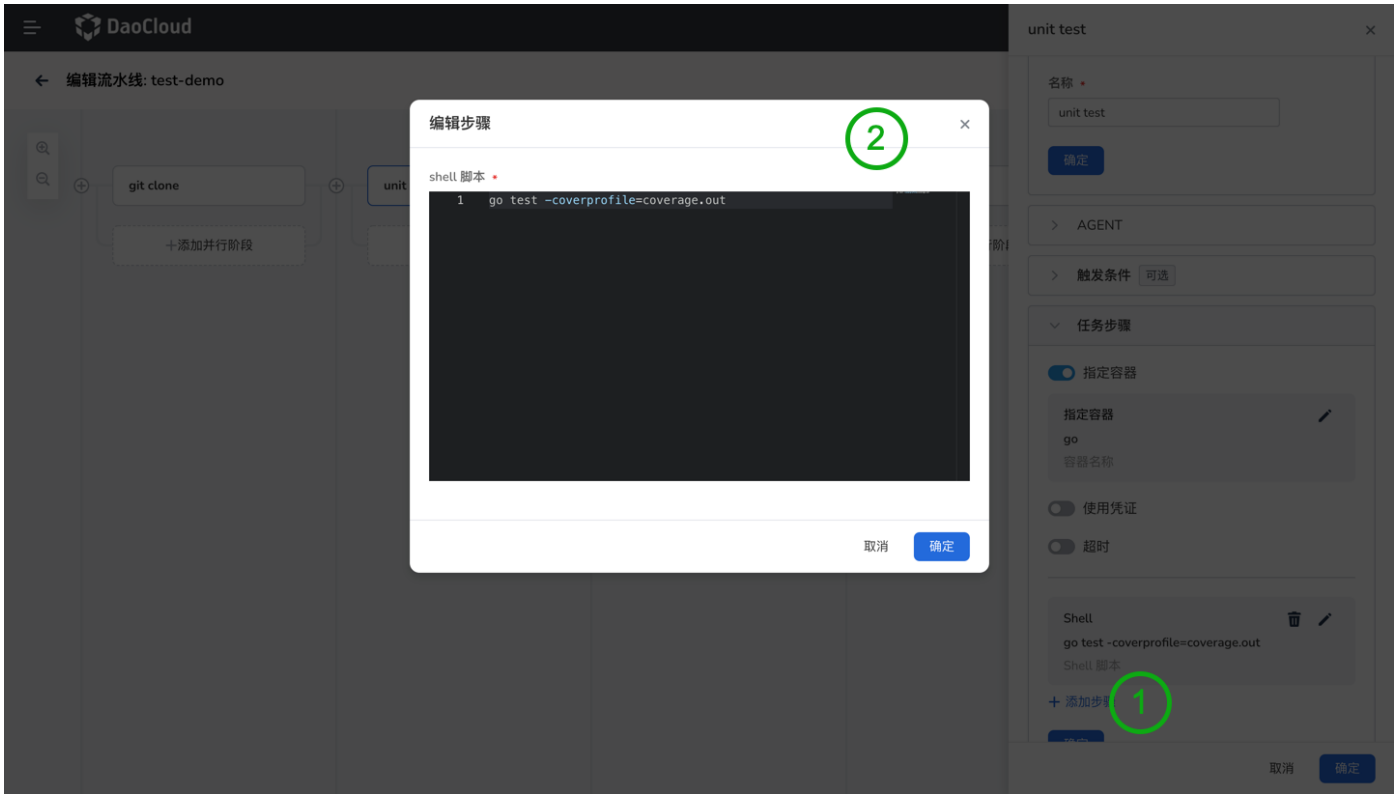


3. 点击 添加步骤以对代码进行单元测试，并生成测试报告 ，在弹出的对话框中步骤类型下选择 shell，并在命令行中输入以下命令，然后点击 确定 。

Copyright © 2016 - 2023 DaoCloud

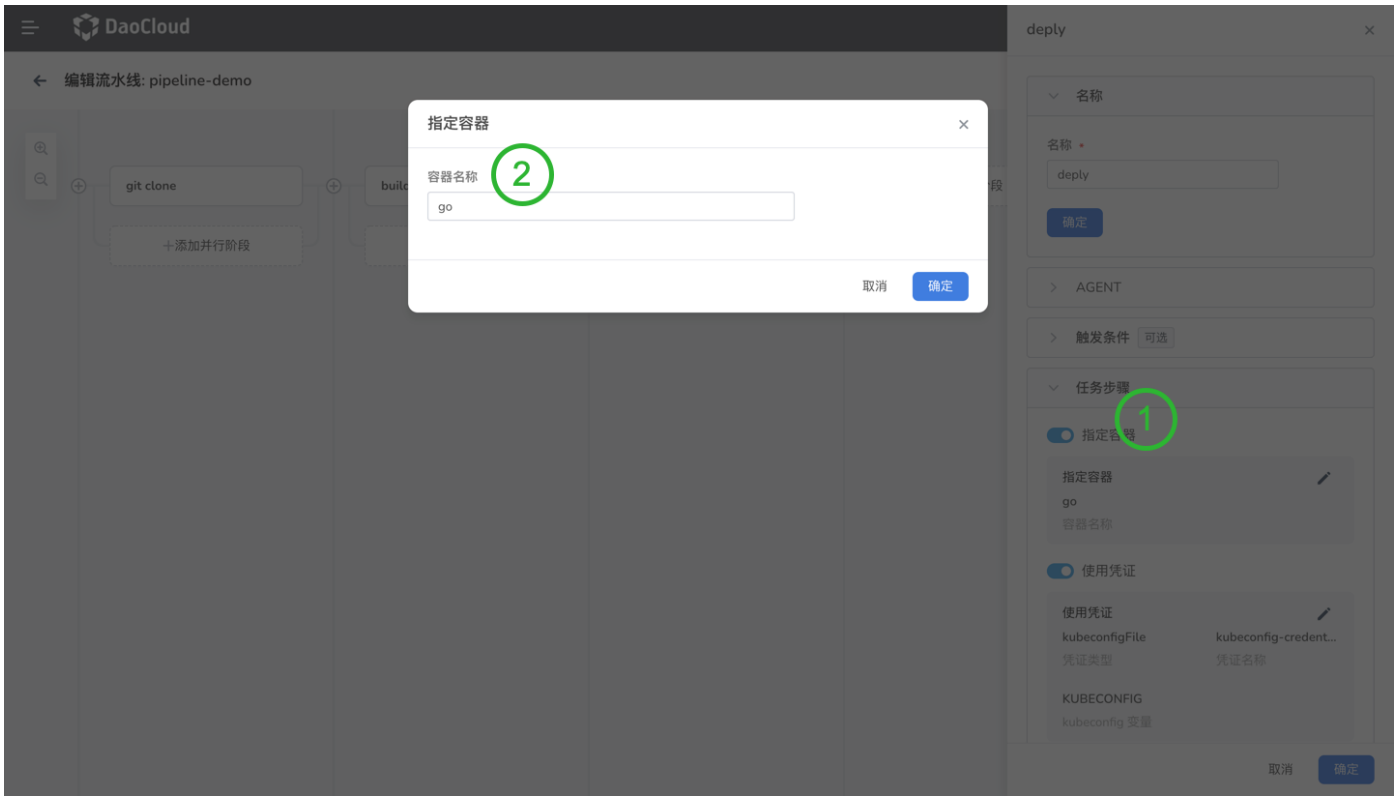
`go`

`go test -coverprofile=coverage.out`



添加阶段 - 构建并推送镜像

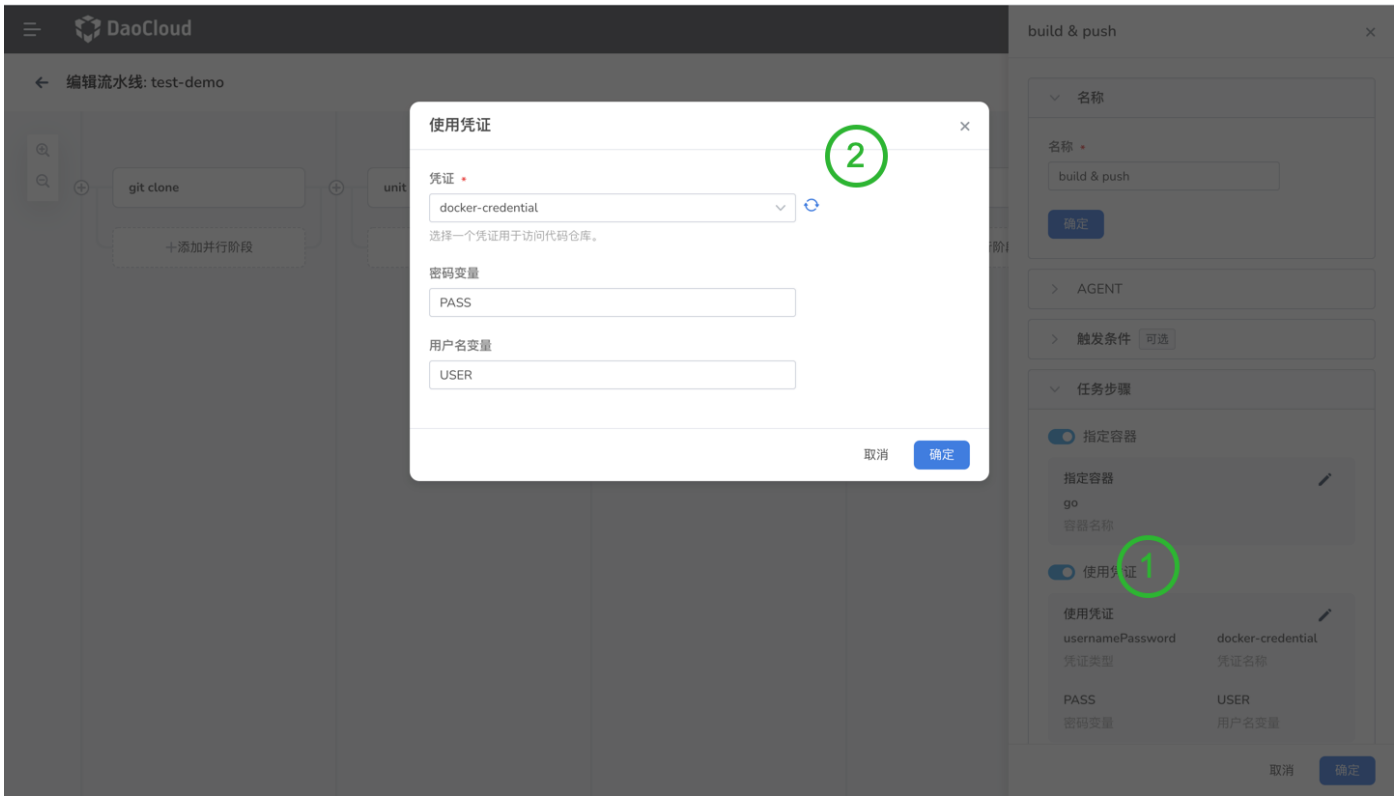
1. 点击画布中的 `添加阶段` 。在右侧的阶段设置中设置名称：build & push。
2. 在步骤模块中选择开启 `指定容器` ，在弹出的对话框中填写容器名称：go，然后点击 `确定` 。



Copyright © 2016 - 2023 DaoCloud

3. 在步骤模块中选择开启 `使用凭证`，在弹出的对话框中填写相关参数，然后点击 `确定`。

- 凭证：选择创建的 docker hub 凭证，用户访问镜像仓库。选择已创建好的 “docker-credential” 凭证。
- 密码变量：PASS
- 用户名变量：USER



4. 点击 `添加步骤` 进行代码构建，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定`。

```
go
go build -o simple-http-server main.go
```

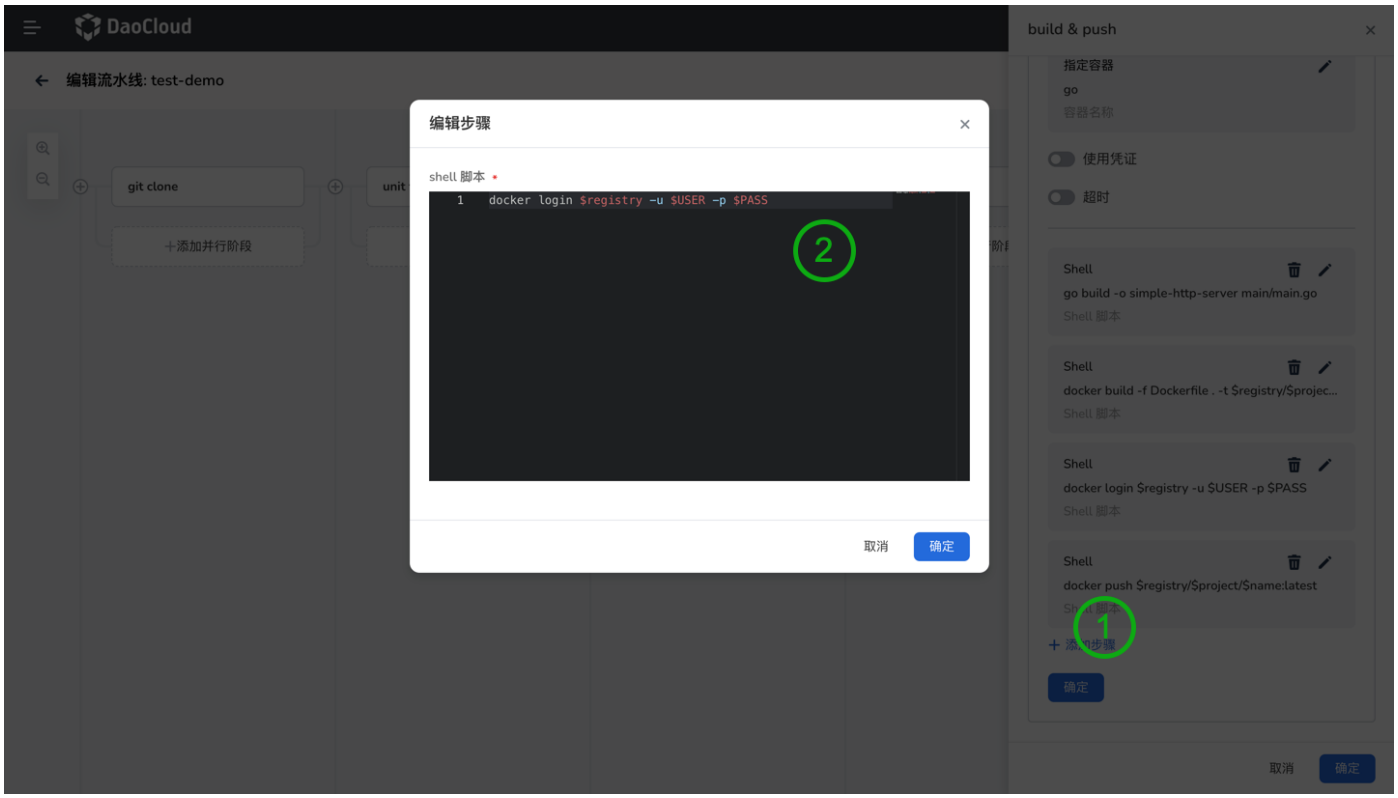
5. 点击 `添加步骤` 以根据源码中的 Dockerfile 构建 Docker 镜像，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定`。

```
docker
docker build -f Dockerfile . -t $REGISTRY/$PROJECT/$NAME:latest
```

6. 点击 `添加步骤` 以登录镜像仓库并将镜像推送至镜像仓库中，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定`。

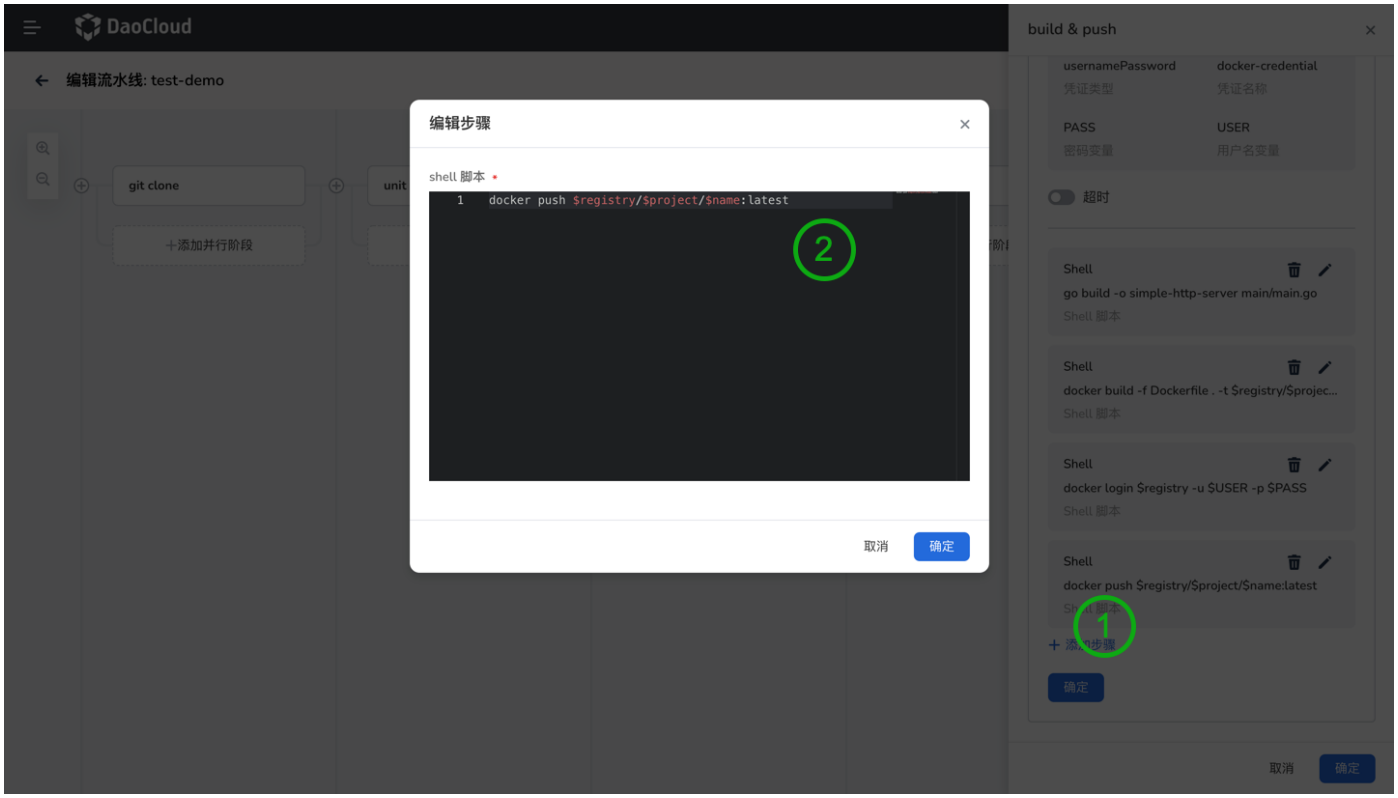
```
docker
docker login $REGISTRY -u $USER -p $PASS
```


Copyright © 2016 - 2023 DaoCloud



7. 点击 `添加步骤` 以将镜像推送至镜像仓库中，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

```
docker
docker push $REGISTRY/$PROJECT/$NAME:latest
```



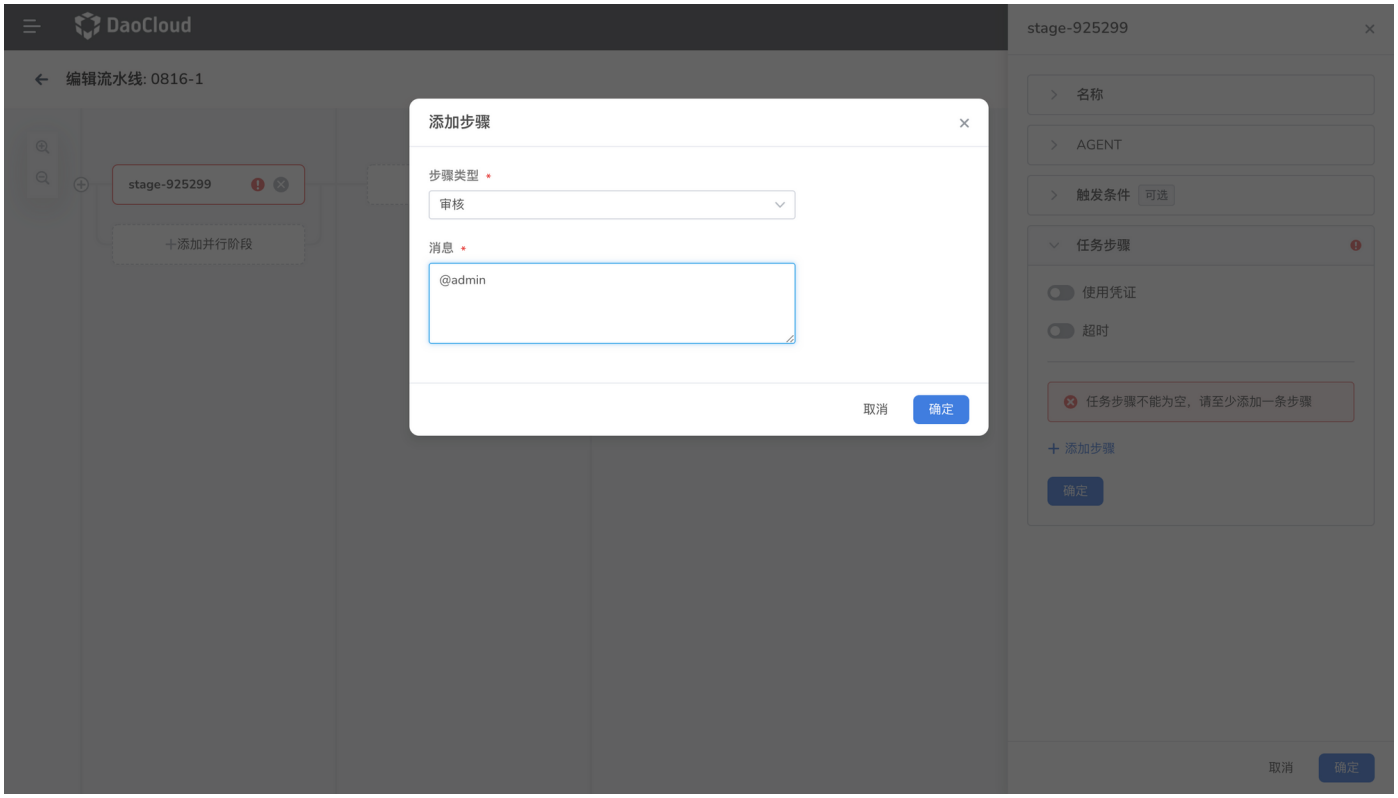
!!! note

```
镜像更新后还可以实现触发流水线，具体操作请参考[触发流水线](trigger.md)
```


Copyright © 2016 - 2023 DaoCloud

添加阶段 - 审核

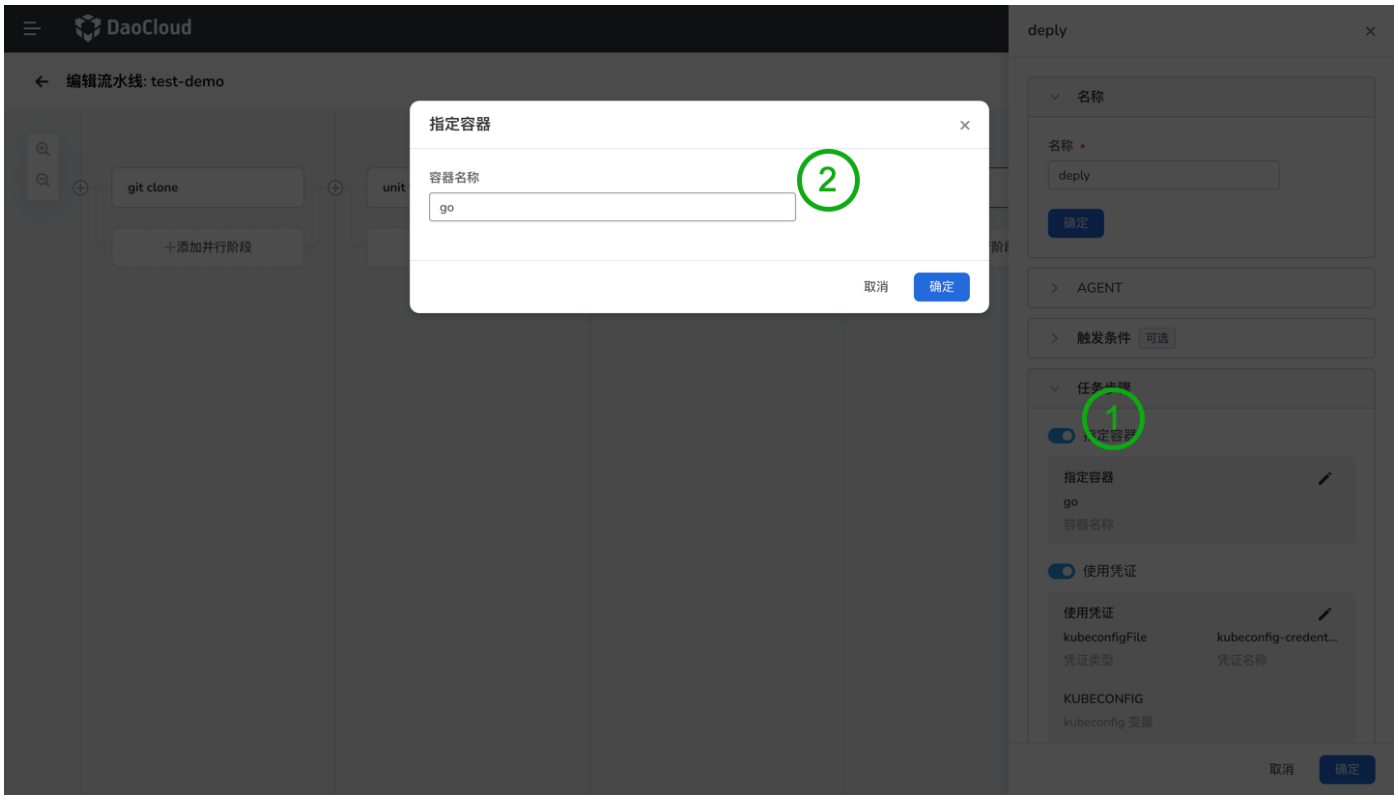
1. 点击画布中的 `添加阶段` 。在右侧的阶段设置中设置名称：review。
2. 点击 `添加步骤` ，在弹出的对话框中步骤类型下选择 `审核` ，在消息字段中填入 `@admin` ，即 `admin` 帐户在流水线运行到该阶段时会进行审核，然后点击 `确定` 。



添加阶段 - 部署至集群

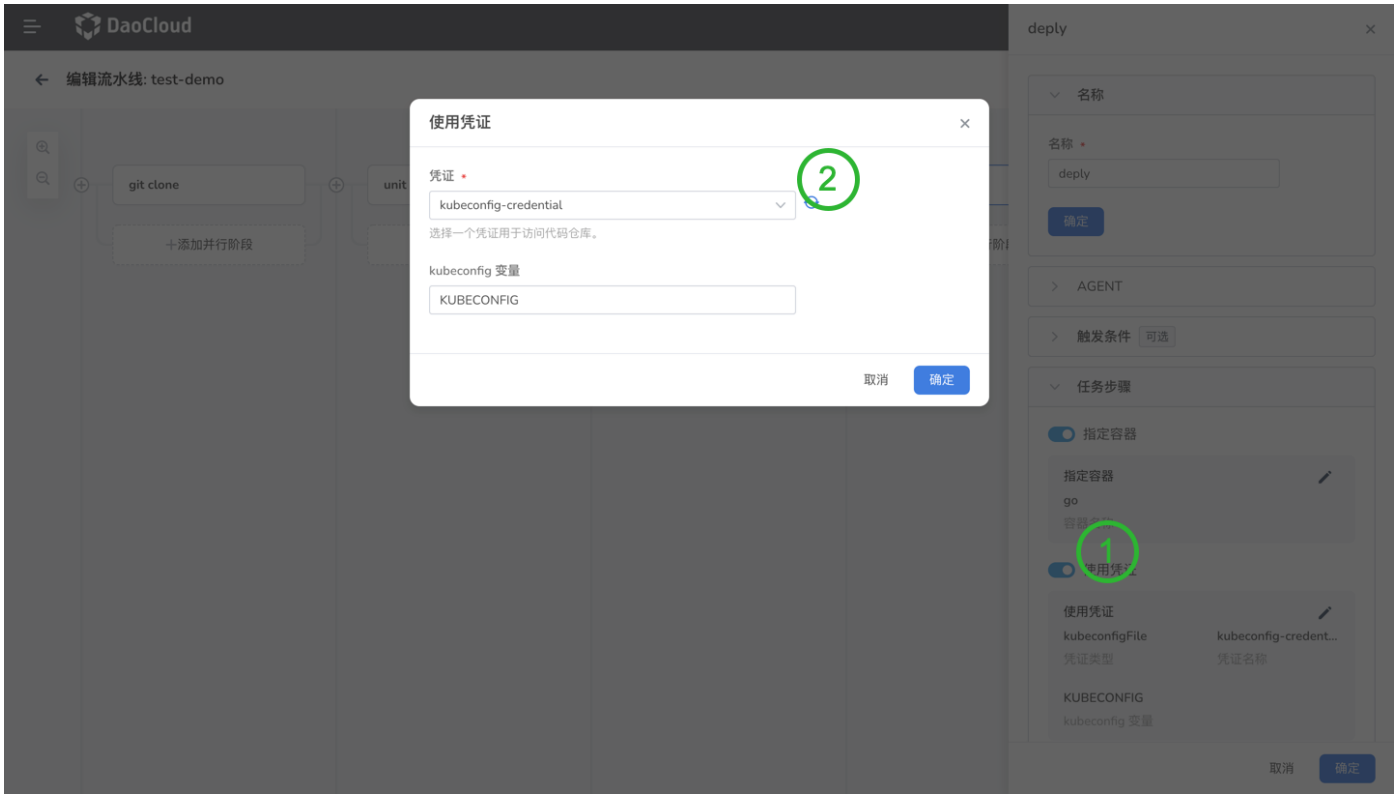
1. 点击画布中的 `添加阶段` 。在右侧的阶段设置中设置名称：deploy。
2. 在步骤模块中选择开启 `指定容器` ，在弹出的对话框中填写容器名称：go，然后点击 `确定` 。

Copyright © 2016 - 2023 DaoCloud



3. 在步骤模块中选择开启 `使用凭证` ，在弹出的对话框中填写相关参数，然后点击 `确定` 。

- 凭证：选择 kubeconfig 类型的凭证。
- kubeconfig 变量：如果使用的是 kubectl apply 的部署方式，变量值必须为 KUBECONFIG。



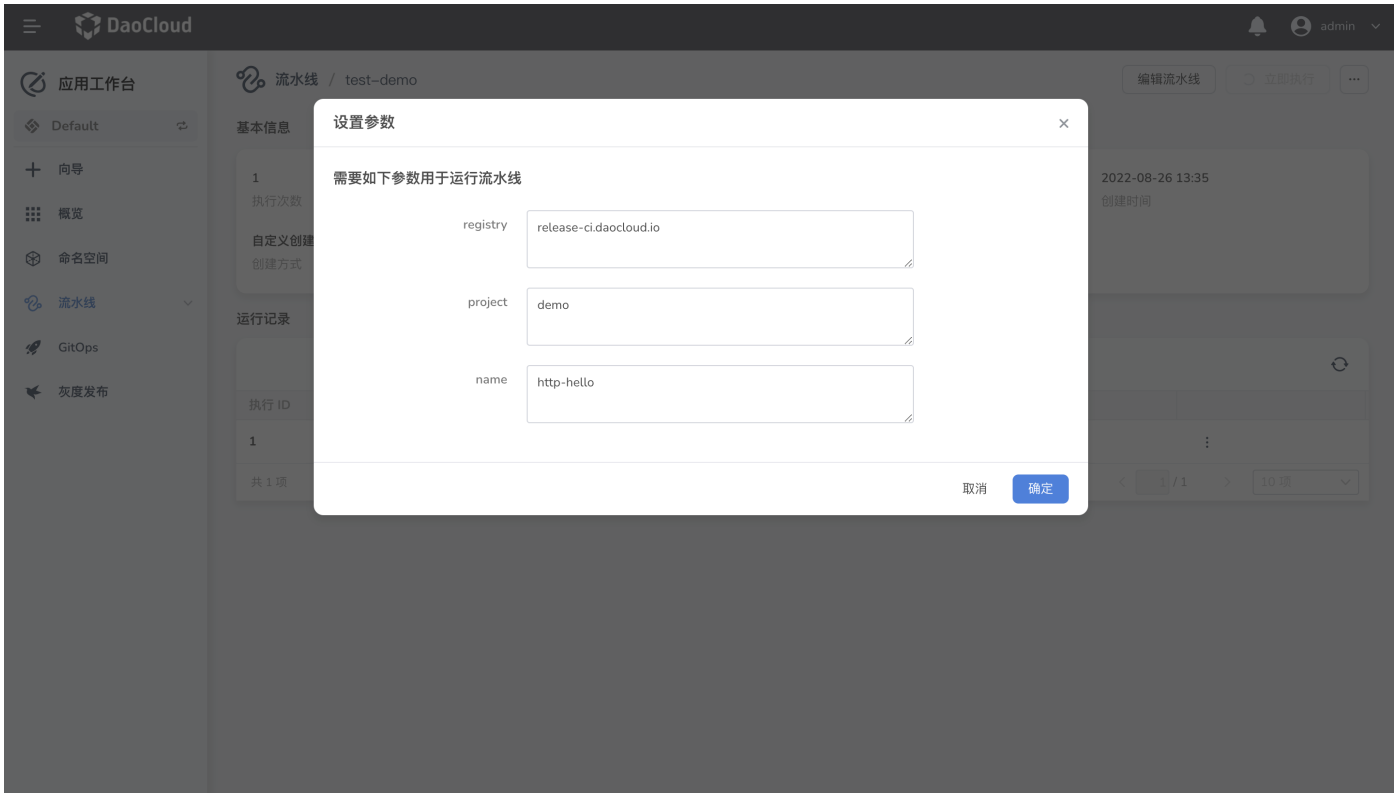
4. 点击 `添加步骤` 以进行集群部署操作，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击 `确定` 。

Copyright © 2016 - 2023 DaoCloud

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

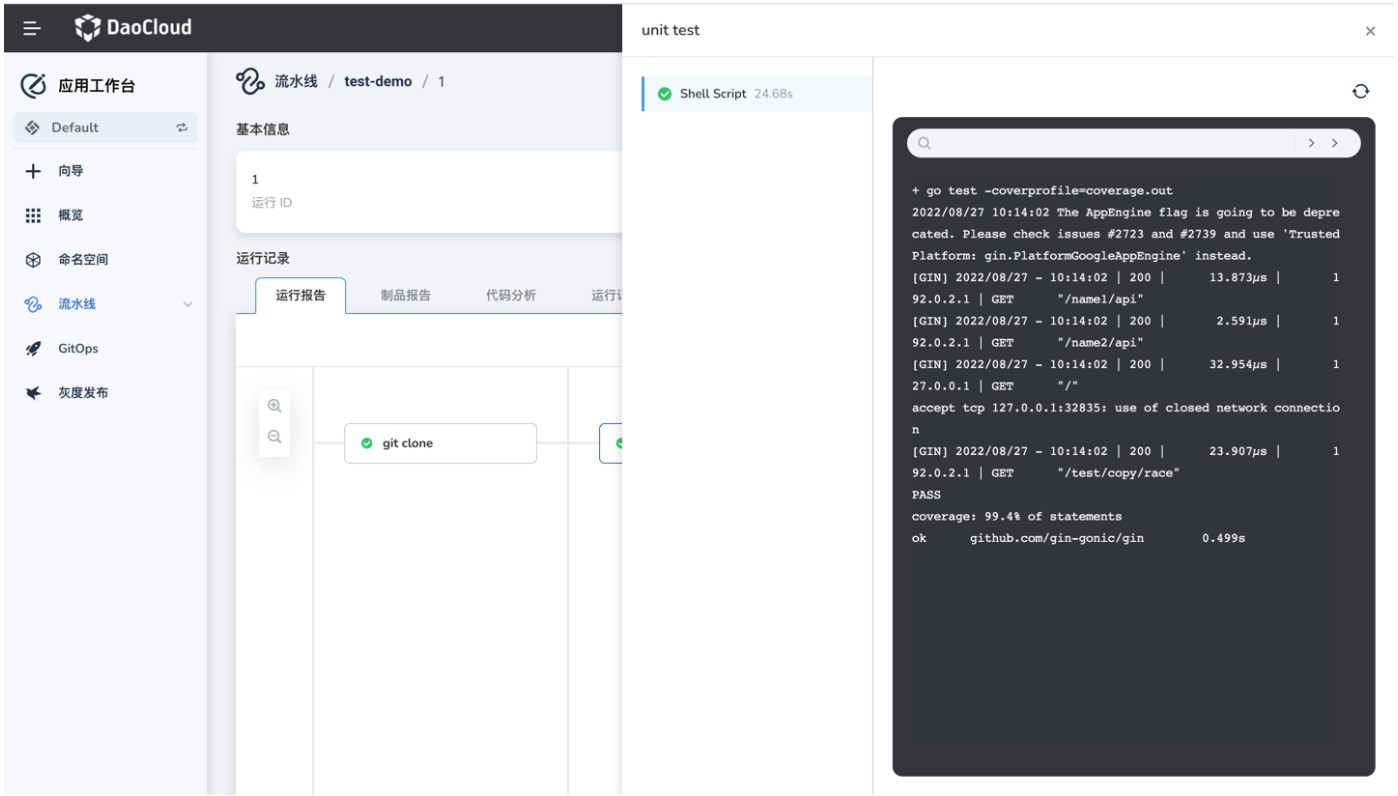
运行流水线

1. 在流水详情页面点击 `立即执行` 。在弹出的对话框中设置前提条件中定义的三个字符串参数，点击 `确定` 来运行流水线。



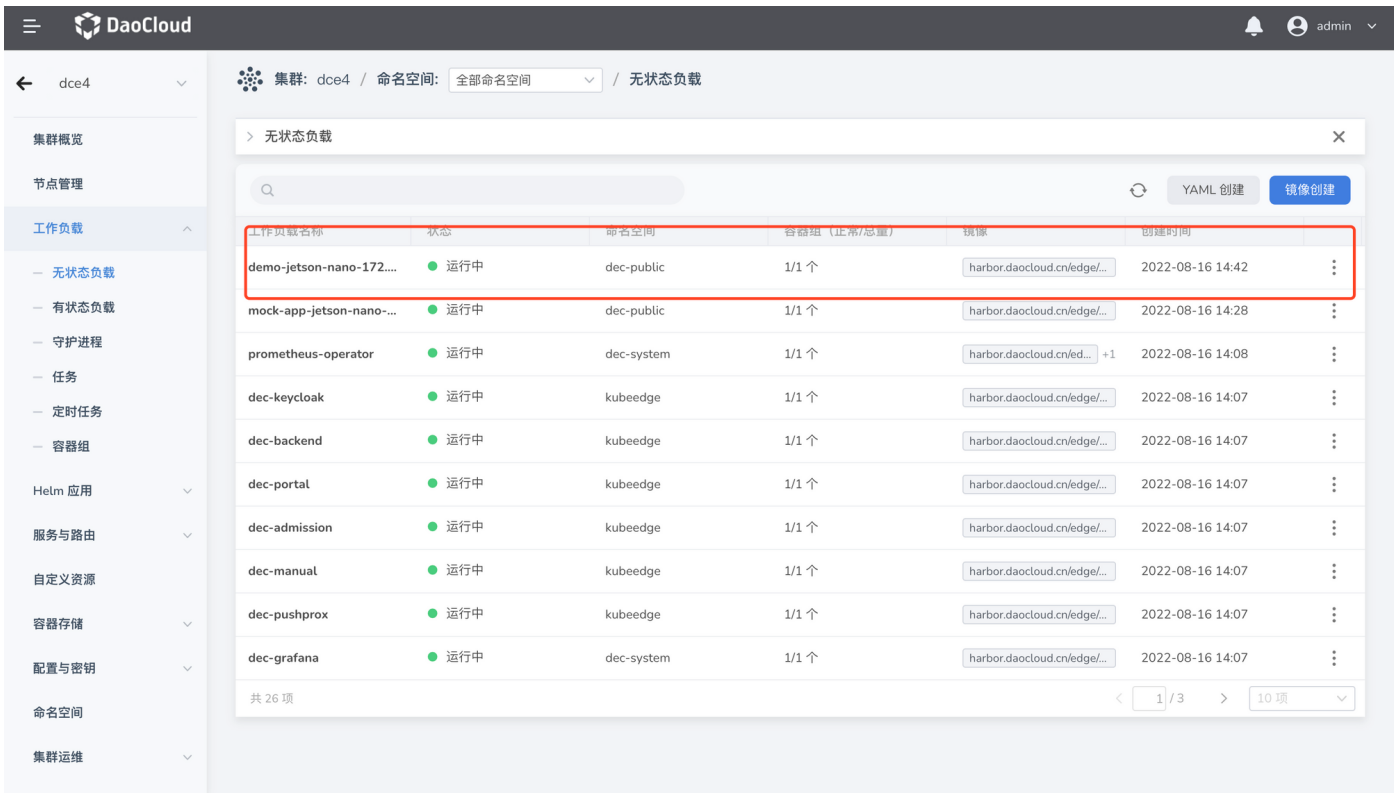
2. 成功开始运行后，页面会自动切换到流水线详情页面，点击当前运行的流水线记录。
3. 进入流水线记录详情页面后，您可以查看到当前流水线运行的过程，需要 admin 或者平台管理员来为流水线进行审核，审核成功后，资源会部署至集群。

Copyright © 2016 - 2023 DaoCloud



验证集群资源

1. 如果流水线的每个阶段都成功运行，则会自动构建一个 Docker 镜像并推送至您的 Docker Hub 仓库。最终，流水线将在您事先设置的项目中自动创建一个无状态负载。
2. 前往容器管理平台，点击集群下的 `工作负载`，您可以看到列表中显示的无状态工作负载。



Copyright © 2016 - 2023 DaoCloud

安装 Jenkins

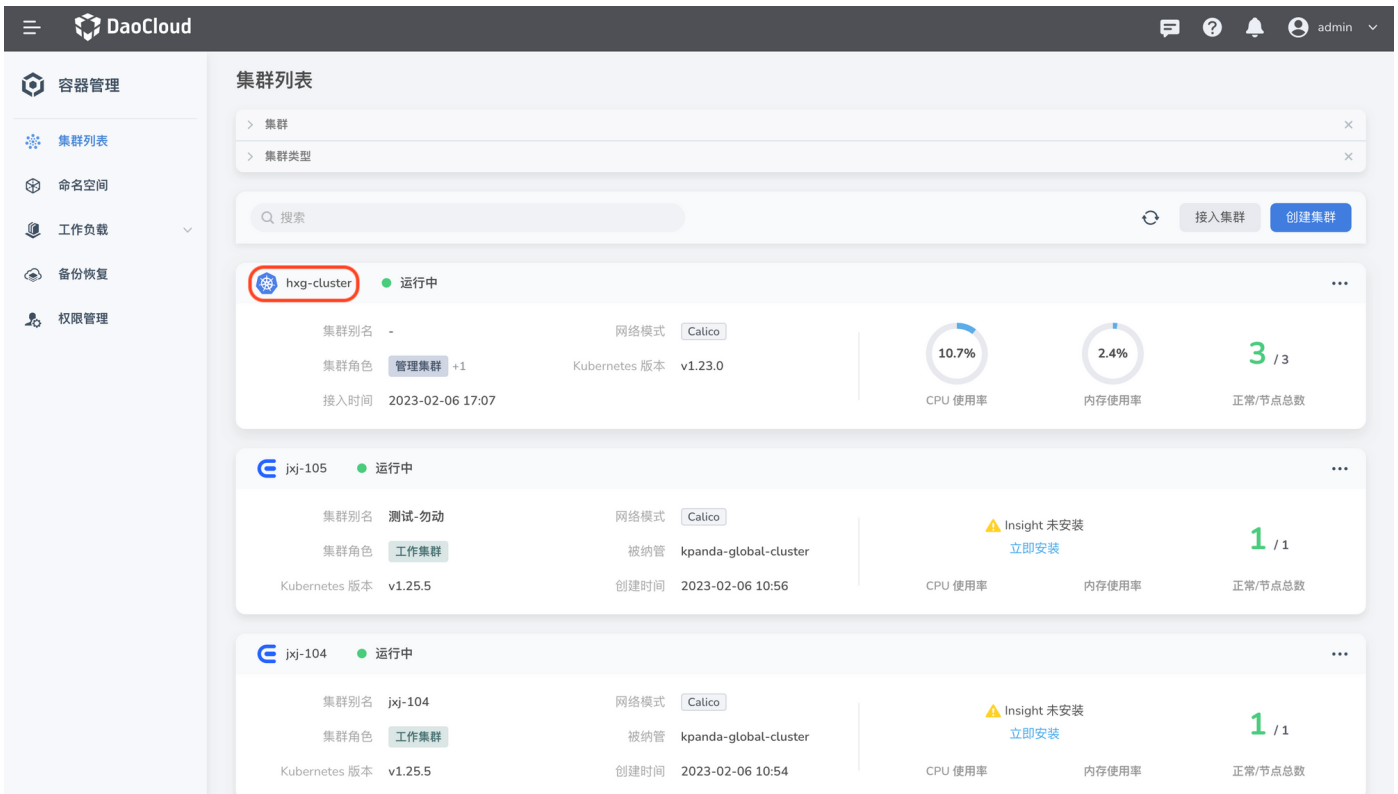
安装 Jenkins 之前需要确保将要 Jenkins 的集群中存在默认的存储类。

Copyright © 2016 - 2023 DaoCloud

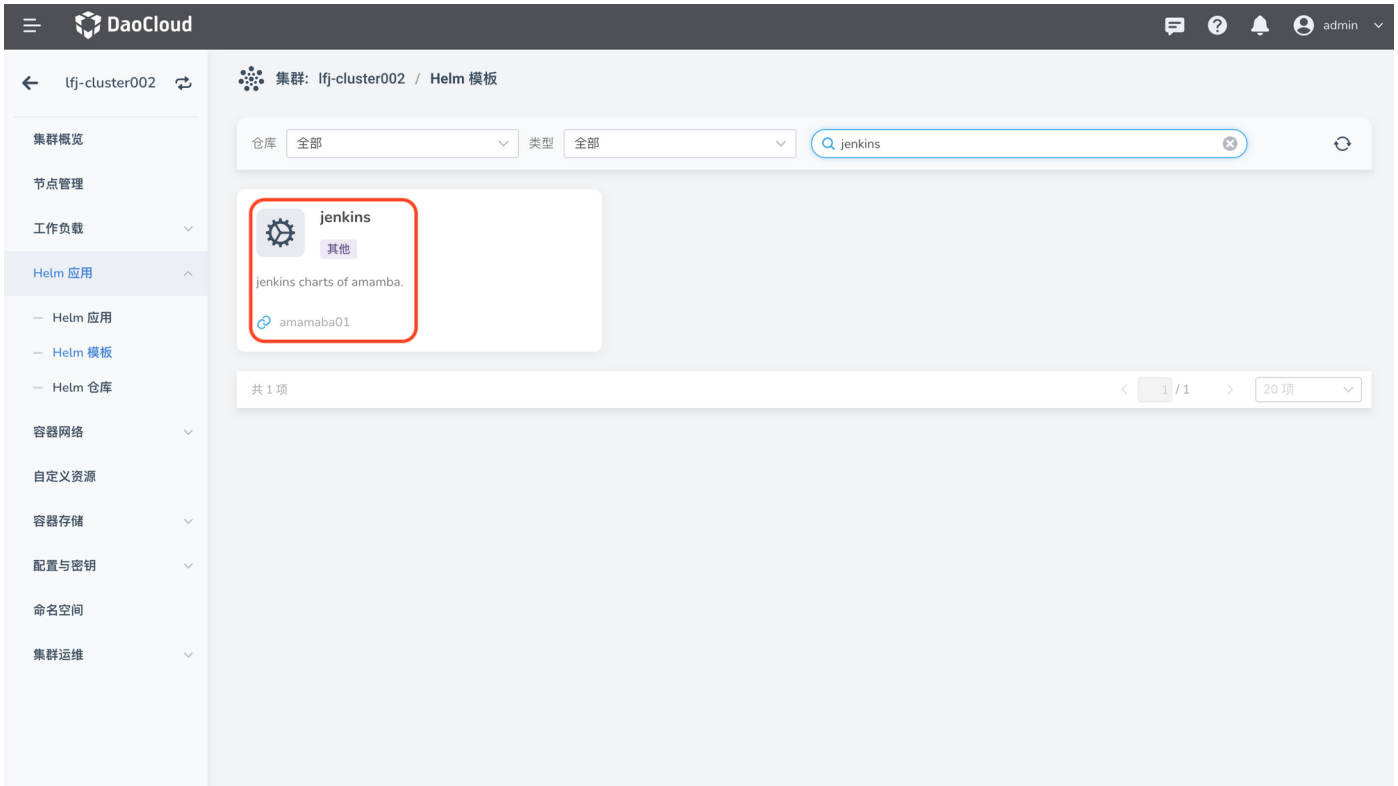
1. 进入 `容器管理` 模块，在 `集群列表` 中找到需要安装 Jenkins 的集群，点击该集群的名称。

   !!! note

   需要根据实际情况选择 Jenkins 的部署集群。目前不建议将其部署在全局服务集群，因为 Jenkins 执行流水线高并发时会占用大量资源，可能会导致全局服务集群的瘫痪。

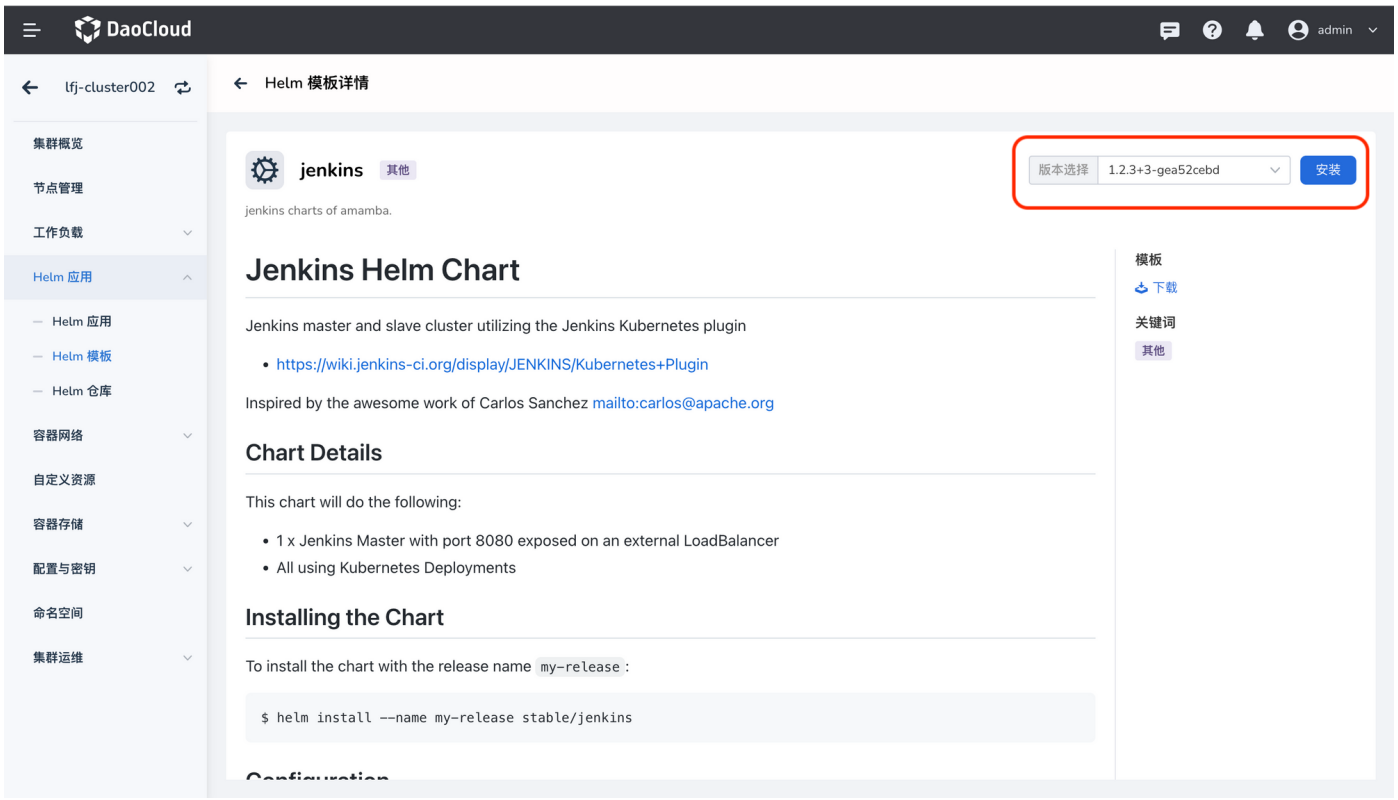


2. 在左侧导航栏中选择 `Helm 应用` -> `Helm 模板`，找到并点击 `Jenkins`。



3. 在 `版本选择` 中选择想要安装的版本，点击 `安装`。

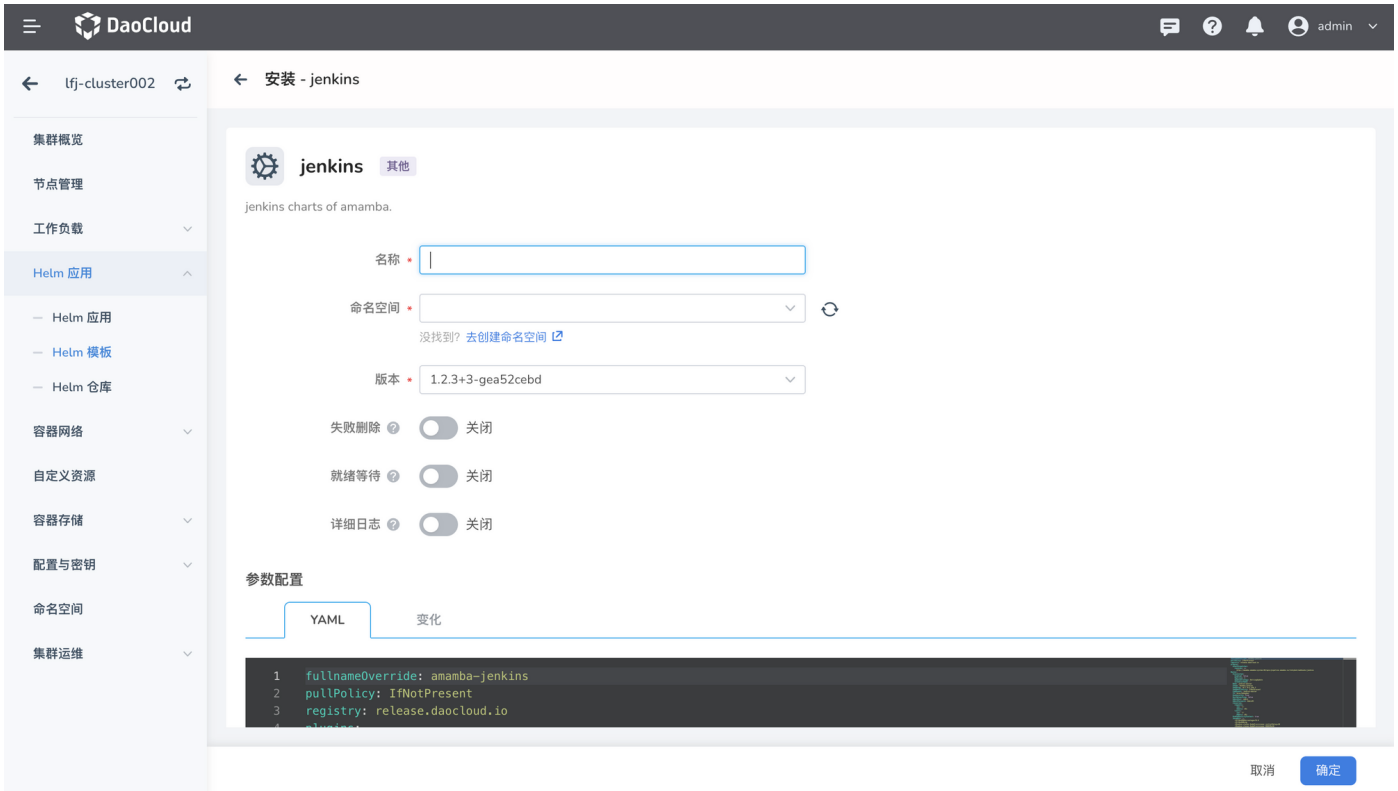
Copyright © 2016 - 2023 DaoCloud



Copyright © 2016 - 2023 DaoCloud

4. 参照下列说明填写各项配置。

- `registry`：Jenkins 镜像的存储地址
- `Master.AdminUser`：Jenkins 账户的用户名
- `Master.AdminPassword`：Jenkins 用户名对应的密码
- `Master.resources.requests`：Jenkins 的资源请求值
- `Master.resources.requests.cpu`：Jenkins master request CPU 资源，根据实际使用情况调整，一般生产级别推荐 4C8Gi 以上
- `Master.resources.limits`：Jenkins 的资源限制值
- `Master.Casc.maxParallelRuns`：最大并行流水线数，默认为 2。根据集群规划可用资源除以 jenkins agent 的平均申请资源
- `Master.JavaOpts`：Jenkins 的 Java 参数，根据实际使用情况调整。其中，`-javaagent` 是用来上报 metric 的 agent
- `Persistence.Size`：Jenkins master PV 大小，大小会影响日志和保存的制品，当存储超过这个数值，会导致 jenkins master 出现异常根据实际使用情况调整，实际生产环境推荐 200G 以上
- `Deploy.JenkinsHost`：Jenkins 的访问链接。如果选择 `Node Port`，访问地址规则为：http://{集群地址:端口}
- `eventProxy.enabled`：`true` 代表部署在 非 全局服务集群，`false` 代表部署在全局服务集群 > 如果 `eventProxy.enabled=true`，则必须填写该字段下的其他各项配置，例如 `eventProxy.image.registry`、`eventProxy.image.repository`、`eventProxy.image.tag`、`eventProxy.imagePullPolicy` 等
- 有关 `eventProxy.configMap.eventroxy.token` 字段中 token 的获取方式，可参考文档：访问密钥
- `Agent.resources.requests.cpu`：Jenkins agent request CPU 资源，默认值 200m
- `Agent.resources.requests.memory`：Jenkins agent request memory 资源，默认值 256Mi
- `Agent.resources.limits.cpu`：Jenkins agent CPU 限制，默认值 200m。需要根据实际运行流水线的情况调整，过小会导致流水线运行时间过长，或者 agent 的 Pod 出现异常
- `Agent.resources.limits.memory`：Jenkins agent memory 限制，默认为 256Mi。需要根据实际运行流水线的情况调整，过小会导致流水线运行时间过长，或者 agent 的 Pod 会出现 OOM 等异常

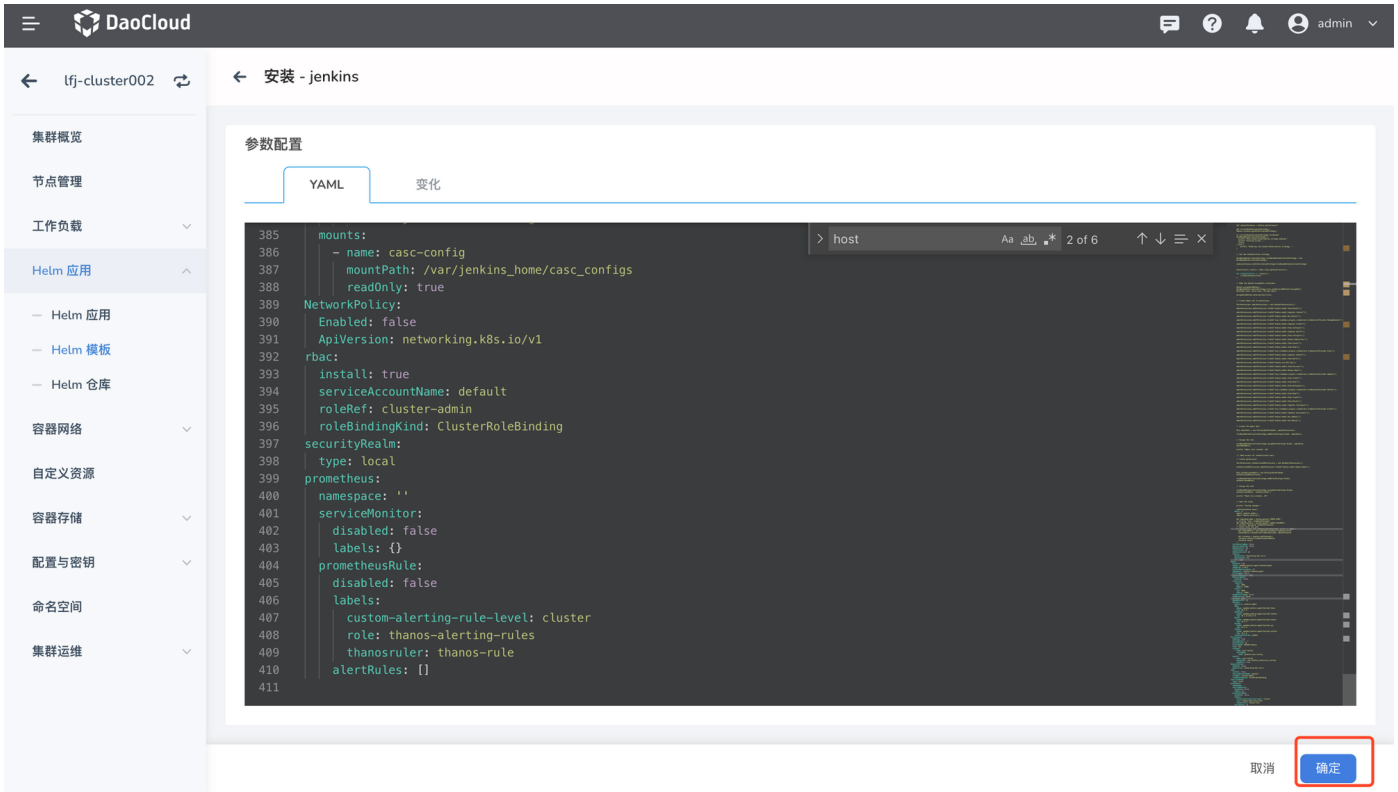


!!! info

```
如需了解 YAML 文件中的其他配置说明，可参考 Jenkins Helm 模板详情页面中 `Configuration` 部分的说明。

![配置说明(../../images/install-jenkins06.png)
```


Copyright © 2016 - 2023 DaoCloud

5. 在右下角点击 确定 按钮即可完成创建。



Jenkins 安装成功后，需要将其集成到应用工作台的工具链中。相关功能和文档正在开发中，敬请期待。

Copyright © 2016 - 2023 DaoCloud

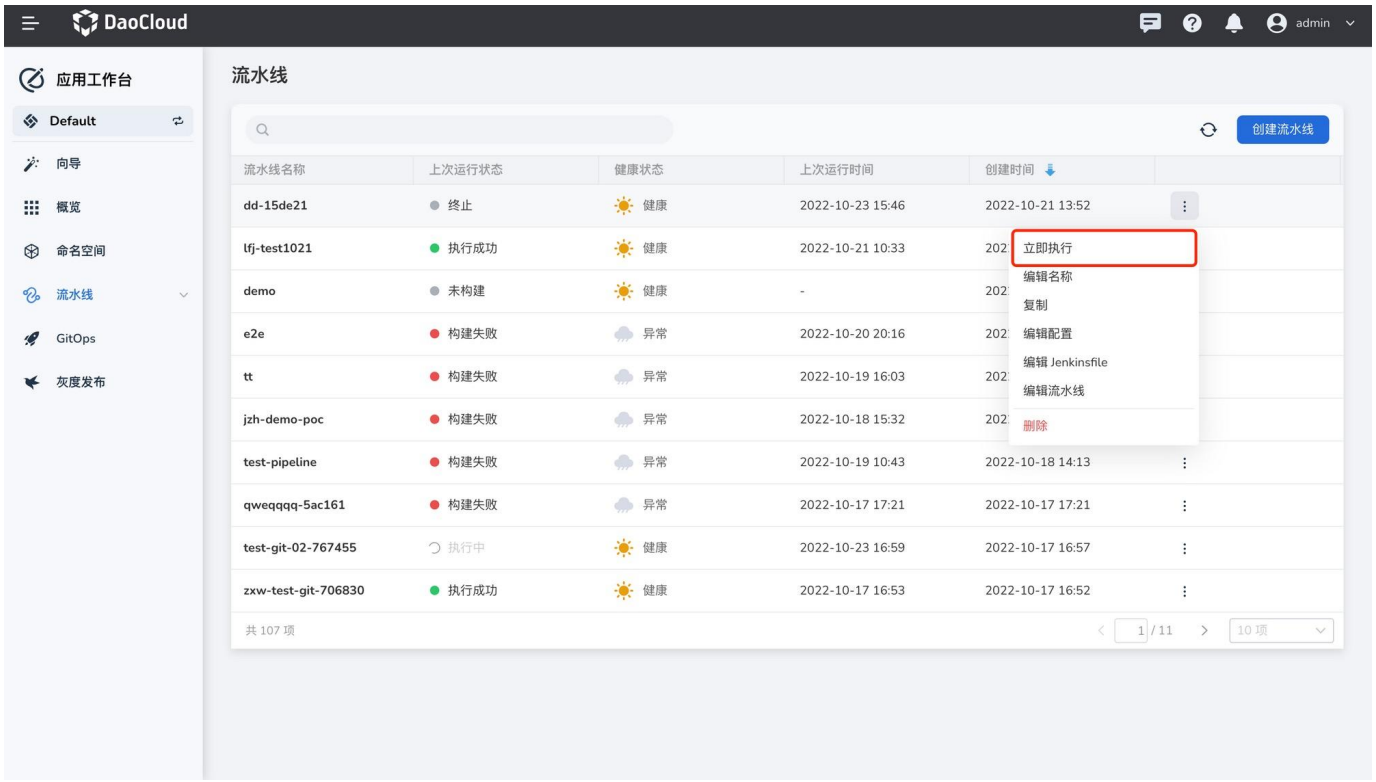
手动执行流水线

本页介绍在图形界面上手动执行流水线，包括立即执行、重新运行和取消运行。

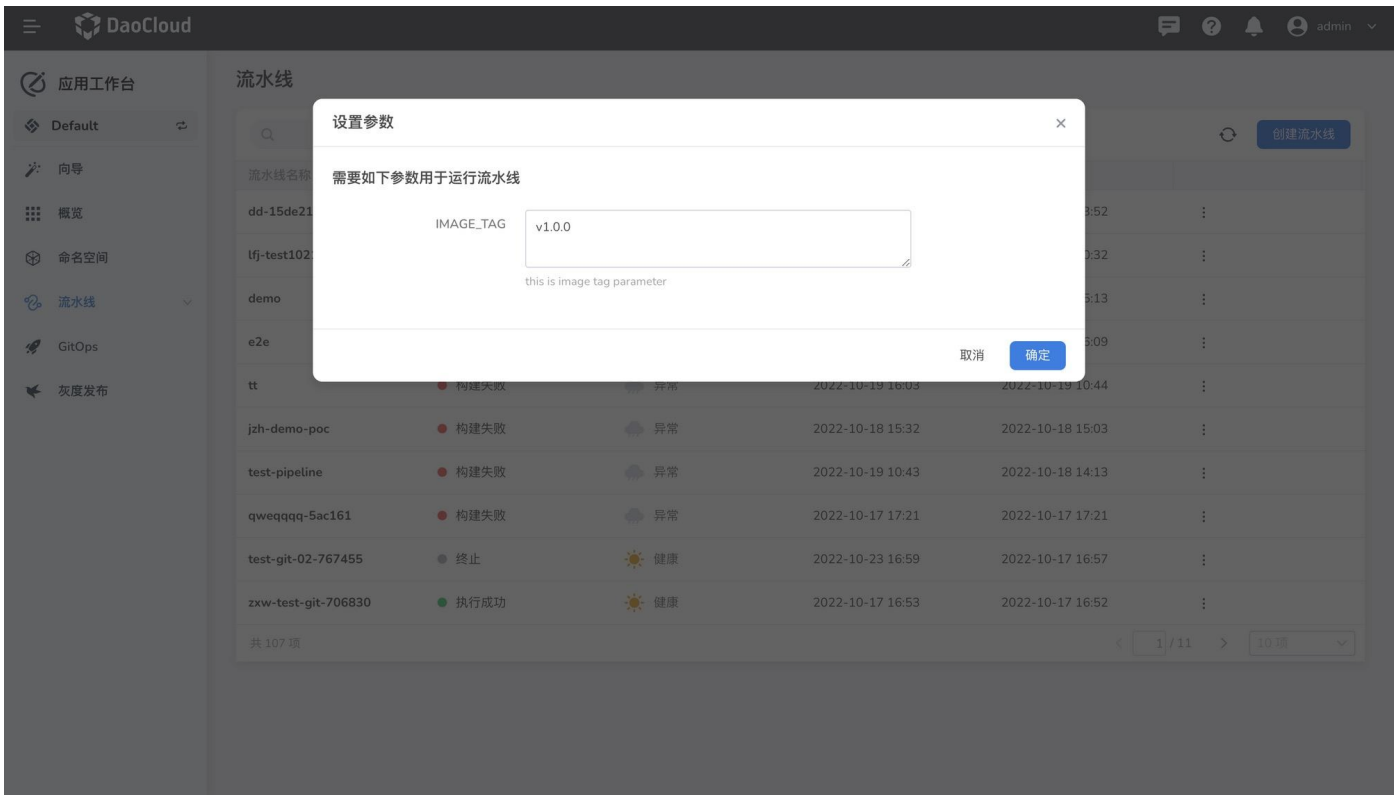
立即执行流水线

1. 在流水线列表页选择某一个流水线，点击 `⋮` ，在弹出的菜单中点击 `立即执行` 。

Copyright © 2016 - 2023 DaoCloud

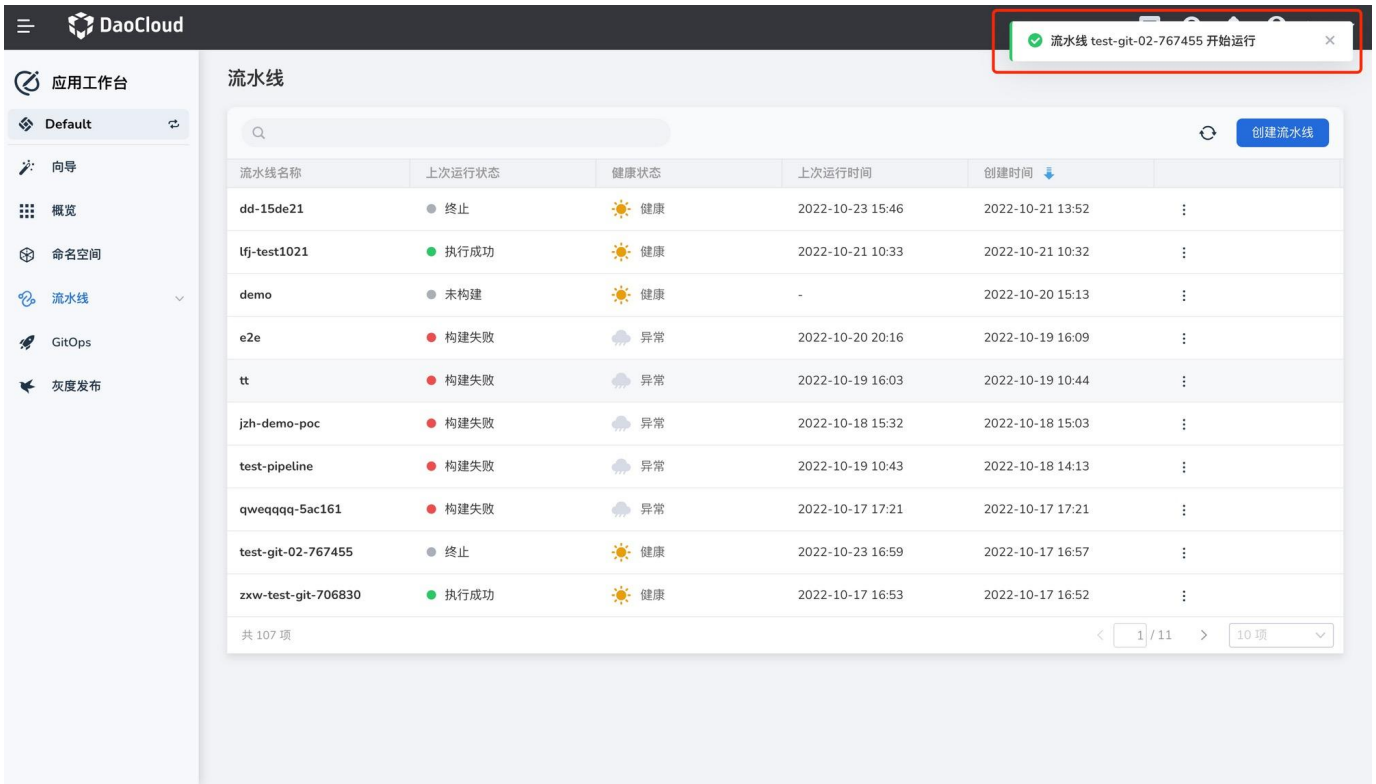


1. 根据流水线是否配置了 `构建参数`，执行后会出现一下两种情况：
2. 如果配置了 `构建参数`，则会出现对话框并展示相关内容进行参数配置。



3. 如果未配置 `构建参数`，则流水线会立即执行。
4. 流水线开始执行。

Copyright © 2016 - 2023 DaoCloud

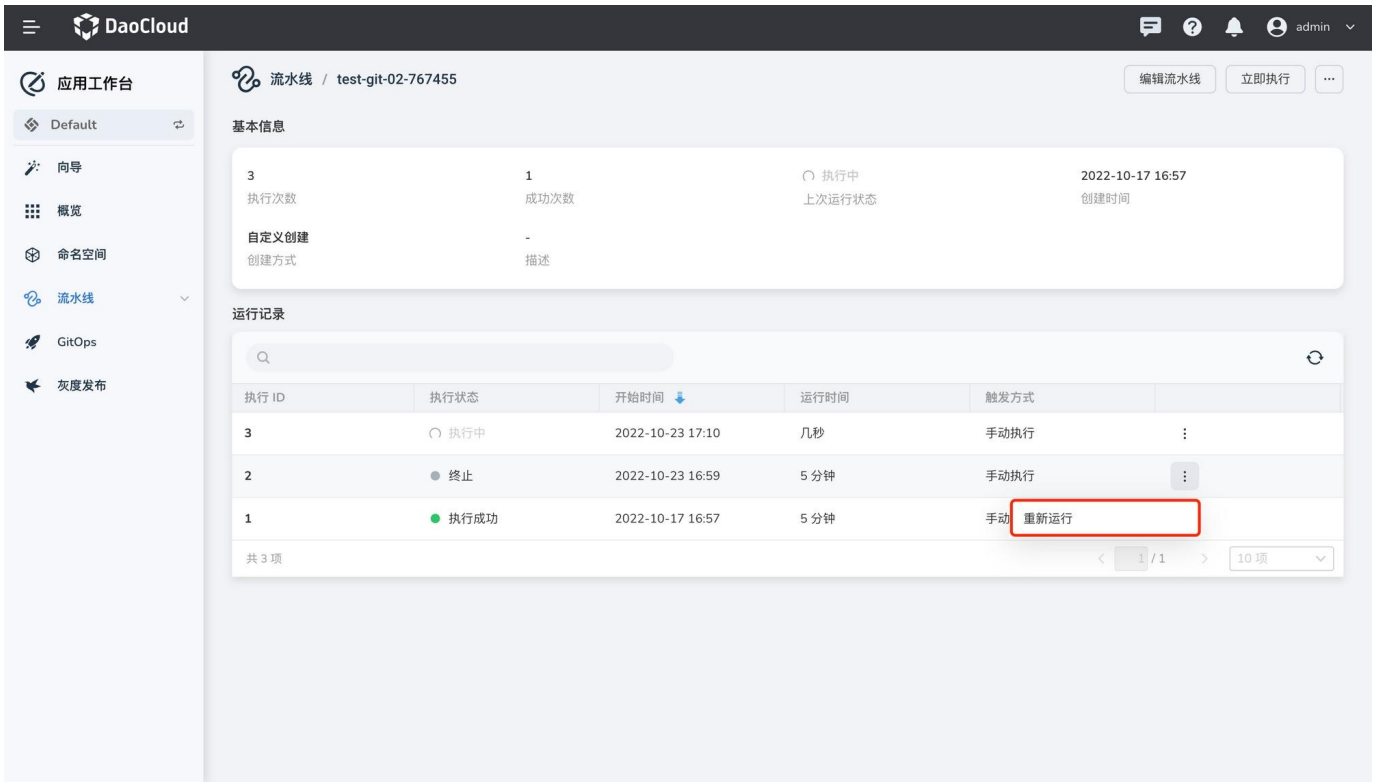


重新运行流水线

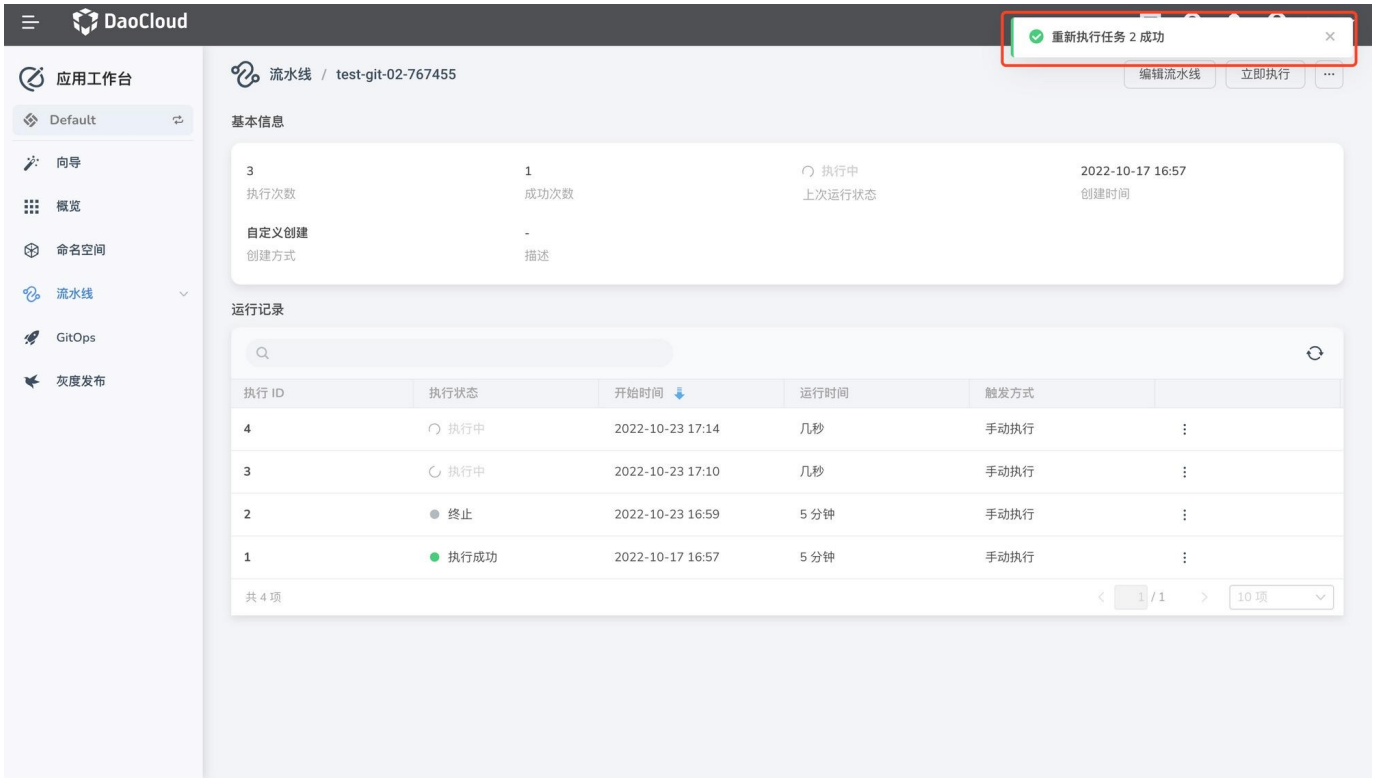
在流水线详情页面，根据流水线运行记录中的执行 ID 可以 重新运行 已运行过的流水线。

1. 在流水线列表中，点击某一个流水线的名称，进入流水线详情页面。
2. 在 运行记录 区域，找到需要重新运行的 执行 ID 。
3. 点击右侧的 ⋮ ，在弹出的菜单中点击 重新运行 。

Copyright © 2016 - 2023 DaoCloud



1. 流水线重新运行成功。

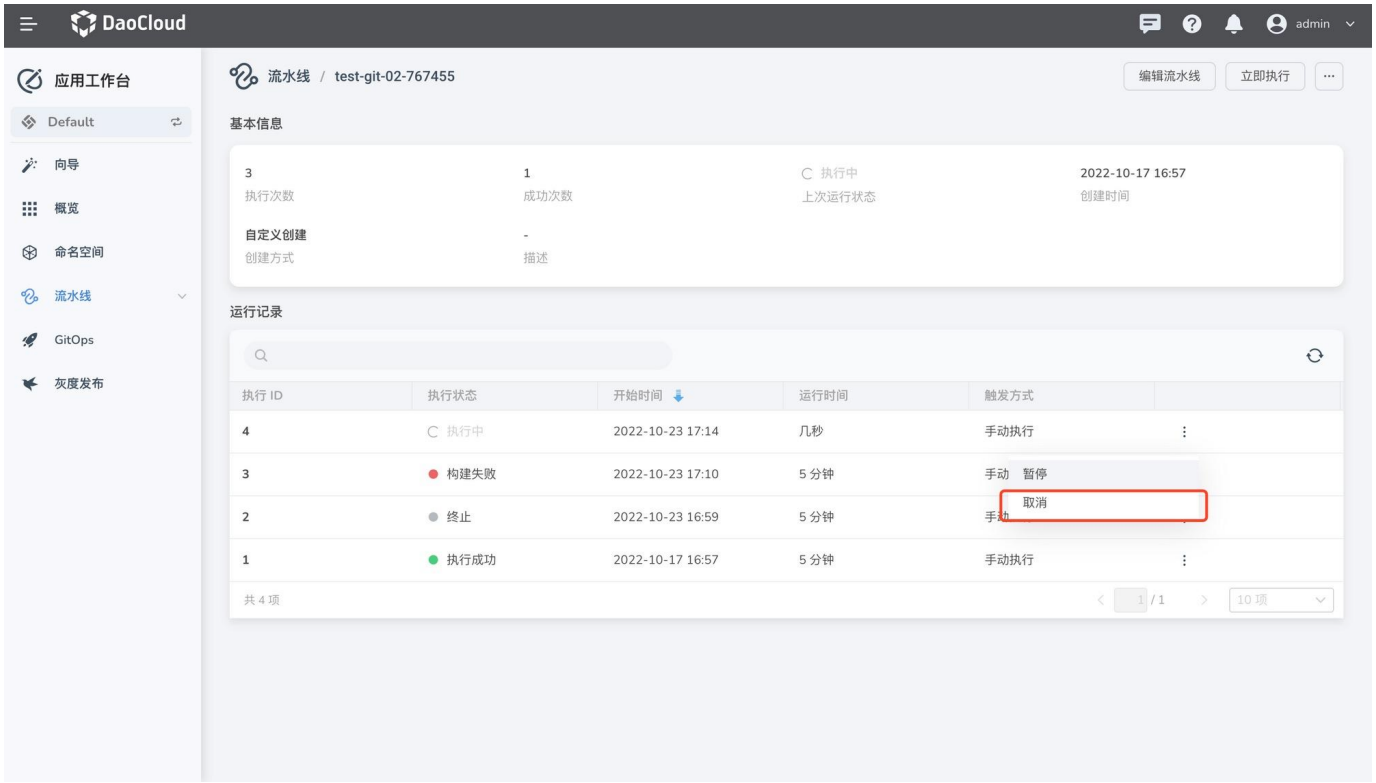


Copyright © 2016 - 2023 DaoCloud

取消运行流水线

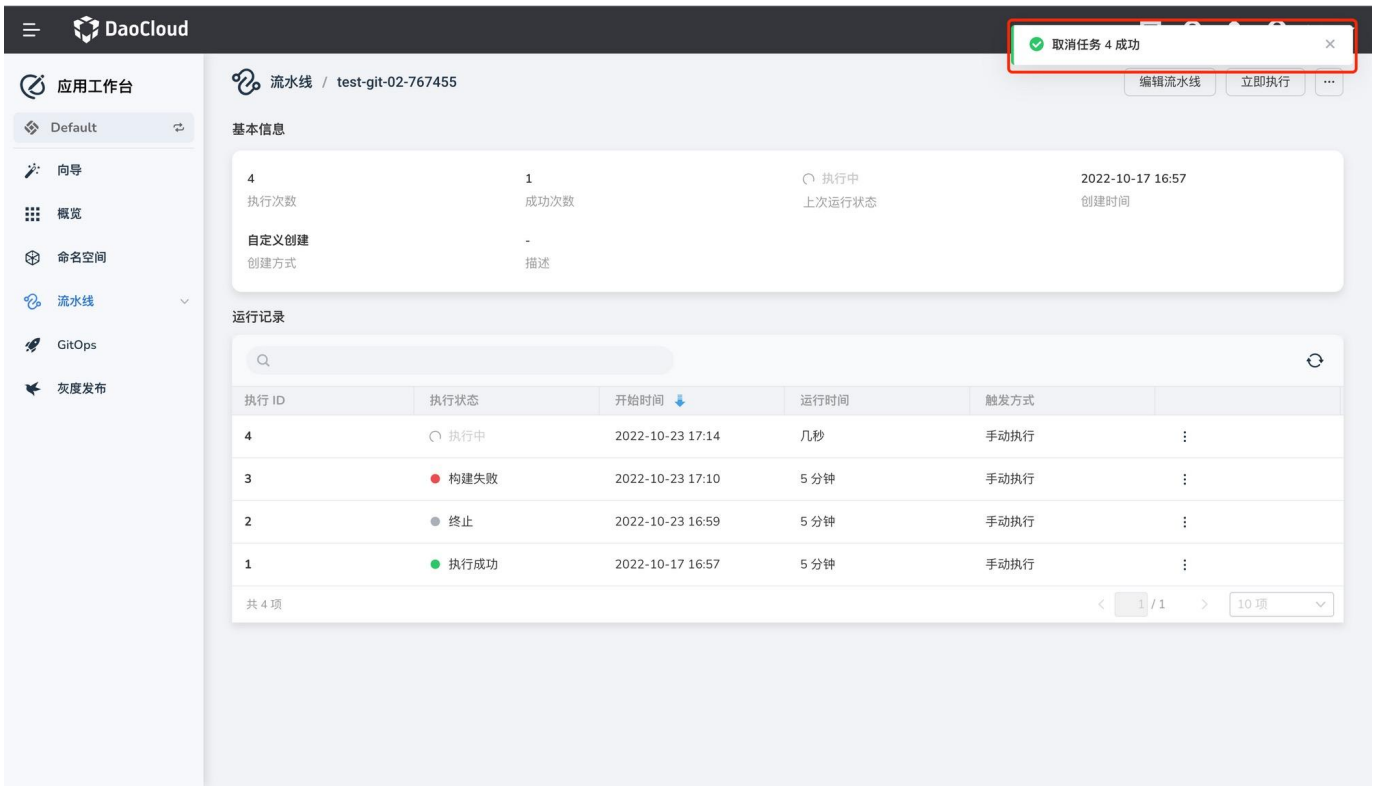
在流水线详情页面，根据流水线运行记录中的 `执行 ID` 可以 `取消` 正在执行的流水线。

1. 在流水线列表中，点击某一个流水线的名称，进入流水线详情页面。
2. 在 `运行记录` 区域，找到需要取消的 `执行 ID`。
3. 点击右侧的 `⋮`，在弹出的菜单中点击 `取消`（适用于状态为 `执行中` 的流水线）。



1. 流水线取消运行成功。

Copyright © 2016 - 2023 DaoCloud



Copyright © 2016 - 2023 DaoCloud

触发器触发流水线

为流水线配置触发器，周期性地自动触发流水线的执行。在编辑流水线时点击 编辑配置 ，即可配置构建触发器。

触发器类型

触发器类型	说明
代码源触发器	在指定的时间检查代码仓库是否有变更。例如填写 `H/15 * * * *` 表示每隔 15 分钟系统会自动检查代码仓库是否有新提交的变更，若有新提交的变更则会执行流水线。
定时触发器	在指定的时间触发流水线，开启定时触发器后，支持直接选择预设的规则，也可以选择自定义 CRON 表达式，输入自定义触发器规则。

触发规则

触发规则语法遵循 CRON 的语法（但有细微差别）。具体来说，每行由 5 个 Tab 或空格分隔的字段组成：`MINUTE HOUR DOM MONTH DOW`

字段	说明	取值范围
MINUTE	分钟	0 ~ 59
HOUR	小时	0 ~ 23
DOM	天	1 ~ 31
MONTH	月	1 ~ 12
DOW	星期	1 ~ 6 代表周一至周六，0、7 代表周日

Copyright © 2016 - 2023 DaoCloud

要为一个字段指定多个值，可以按照优先顺序使用以下运算符：

运算符	说明
*	表示匹配取值范围内的所有值
M-N	表示指定范围内的所有值
M-N/X 或 */X	表示在指定范围或整个有效范围内每隔 X 触发一次
A,B,...,Z	表示匹配多个值

为了使定期调度的任务在系统上产生均匀的负载，应尽可能使用符号 H（代表“哈希”）。 例如，对十几个日常工作使用 `0 0 * * *` 将导致午夜时出现较大的峰值，可能会导致资源紧张。 相比之下，使用 `H H * * *` 仍然会每天执行一次每个作业，但不是同时执行一次，这样能够更好地使用有限的资源。

H 可以与范围一起使用。例如，`H H(0-7) * * *` 表示 00:00 到 7:59 之间的某个时间。

由于不同的月份天数不同，所以当 `DOM` 中出现 `/3` 或 `H/3` 等短周期时，在大多数月份结束时不会触发。 例如，`*/3` 将在每个月的第 1 天、第 4 天、……、第 31 天触发任务。如果下个月有 30 天，最后一次触发任务的时间为第 28 天。

另外，@yearly、@annually、@monthly、@weekly、@daily、@midnight 和 @hourly 作为方便的别名。 这些使用哈希进行自动匹配。例如，@hourly 与 `H * * * *` 相同，可以表示一小时内的任何时间。 @midnight 代表每天 0:00 ~ 2:59 之间的一段时间。

规则举例

规则	说明
H/15 * *	表示每隔 15 分钟触发一次，例如每小时的 07 分、22 分、37 分和 52 分
H(0-29)/10 * *	表示在每个小时的前半小时中，每 10 分钟触发一次，例如每小时的 04 分、14 分和 24 分
45 9-16/2 * * 1-5	表示每个工作日从 9 点 45 分开始下午 15 点 45 分结束，每隔 2 小时，在第 45 分钟触发一次
H H(8-15)/2 * * 1-5	表示每个工作日从 8 点到 16 点这段时间内，每隔 2 小时之间的某刻触发一次，例如在 9:38、11:38、13:38、15:38
H H 1,15 1-11	表示每月 1 日和 15 日每天一次，除 12 月外

Copyright © 2016 - 2023 DaoCloud

Config

流水线节点（Agent）

Agent 描述了整个 流水线 执行过程或者某个 阶段 的执行环境，必须出现在 描述文件 顶部或每一个 阶段 。

本文基于 Kubernetes plugin for Jenkins 插件描述如何扩展 Kubernetes 中运行的 Jenkins Agent。

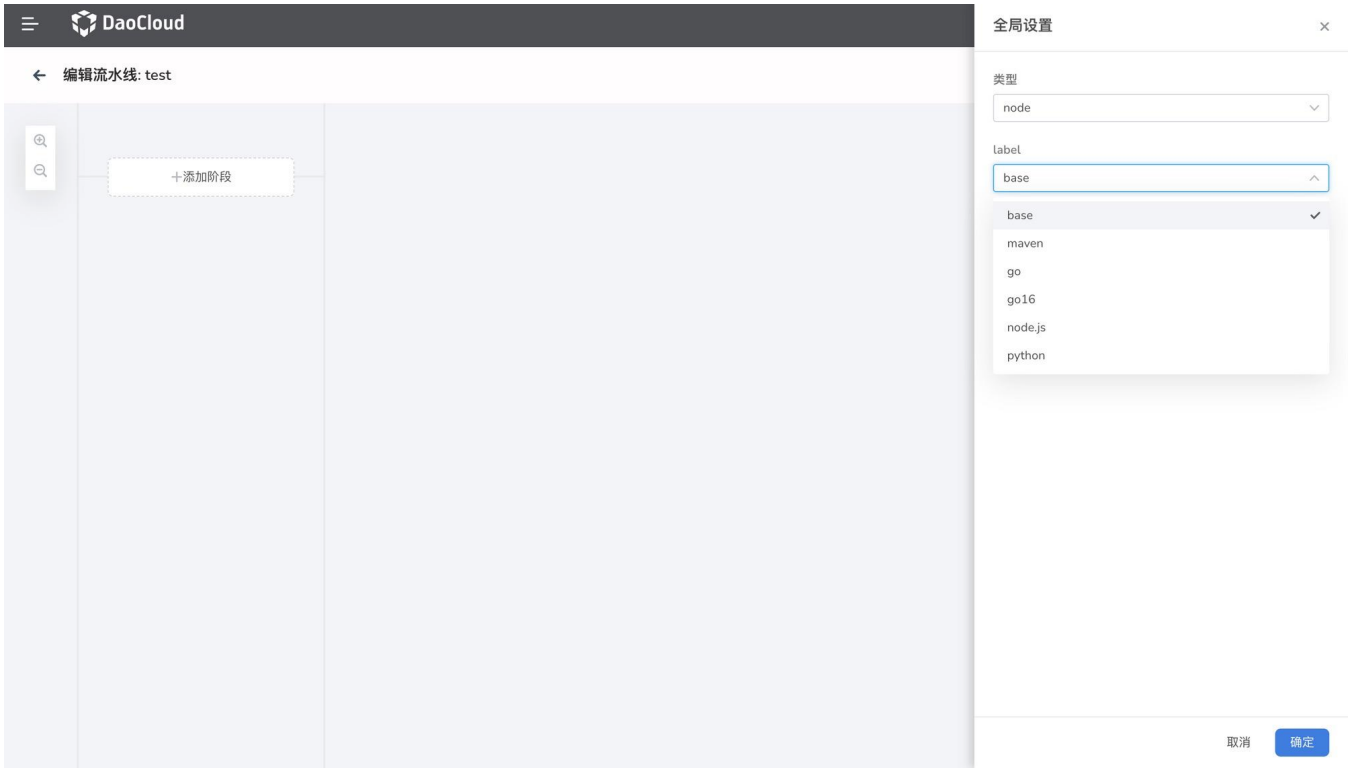
KUBERNETES POD 模板介绍

这个 Kubernetes 插件会在 Jenkins Agent Pod 中运行一个特殊的容器 `jnlp`，目的是为了在 Jenkins Controller 和 Jenkins Agent 之间进行通信，所以需要定义其他容器来运行流水线步骤，并且可以通过 `container` 命令来切换不同的容器。

使用内置 LABEL

应用工作台通过 podTemplate 能力声明了 6 个 label：`base`、`maven`、`go`、`go16`、`node.js` 和 `python`。您可以指定具体的 Agent 标签来使用对应的 podTemplate。

- 可以在 Jenkinsfile 中通过 `node('go')` 使用 go 的 podTemplate。

  ```bash pipeline { agent { node { label 'go' } }

  stages { stage('go') { steps { container('go') { sh 'go version' } } } } } ```

- 也可以在 编辑流水线 页面上选择类型为 `node` 且 label 为 `go` 的 Agent。





Copyright © 2016 - 2023 DaoCloud
  ```

内置 Label 环境说明

Jenkins Agent Label: base

名称	类型/版本
容器名称	base
操作系统	centos-7 (7.9.2009)
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: maven

名称	类型/版本
容器名称	maven
操作系统	centos-7 (7.9.2009)
Jdk	openjdk-1.8.0_322
Maven	3.5.3
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: go

名称	类型/版本
容器名称	go
操作系统	centos-7 (7.9.2009)
Go	1.12.10
GOPATH	/home/jenkins/go
GOROOT	/usr/local/go
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Copyright © 2016 - 2023 DaoCloud

Jenkins Agent Label: go16

名称	类型/版本
容器名称	go
操作系统	centos-7 (7.9.2009)
Go	1.16.8
GOPATH	/home/jenkins/go
GOROOT	/usr/local/go
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: node.js

名称	类型/版本
容器名称	nodejs
操作系统	centos-7 (7.9.2009)
Node	v10.16.3
Yarn	1.16.0
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: python

名称	类型/版本
容器名称	python
操作系统	centos-7 (7.9.2009)
Python	3.7.11
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

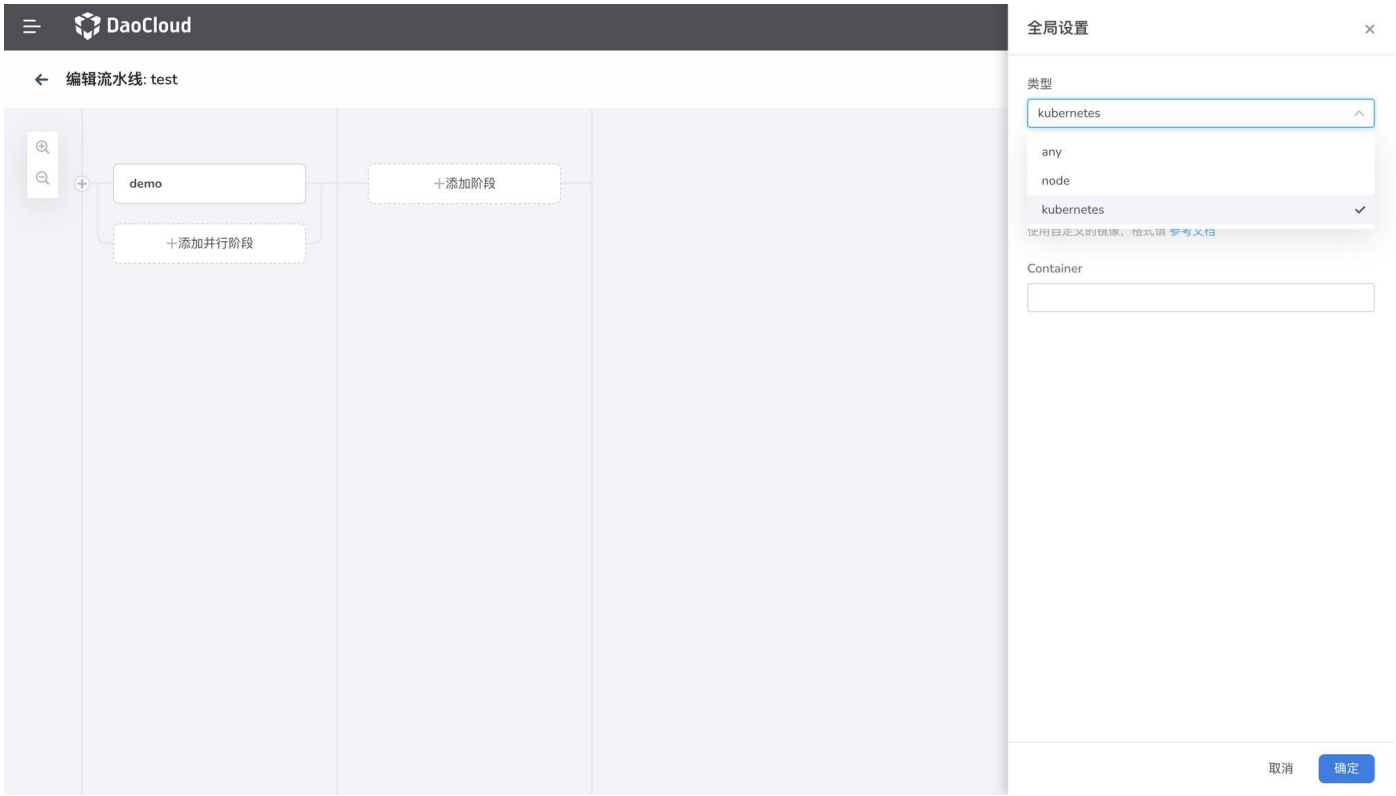
Copyright © 2016 - 2023 DaoCloud

使用 YAML 自定义 PODTEMPLATE

如果需要运行特定环境的 Jenkins Agent，可以在流水线上自定义 Jenkins Agent。

Copyright © 2016 - 2023 DaoCloud

1. 在 `编辑流水线` 页面上选择 Agent 类型为 `kubernetes`。

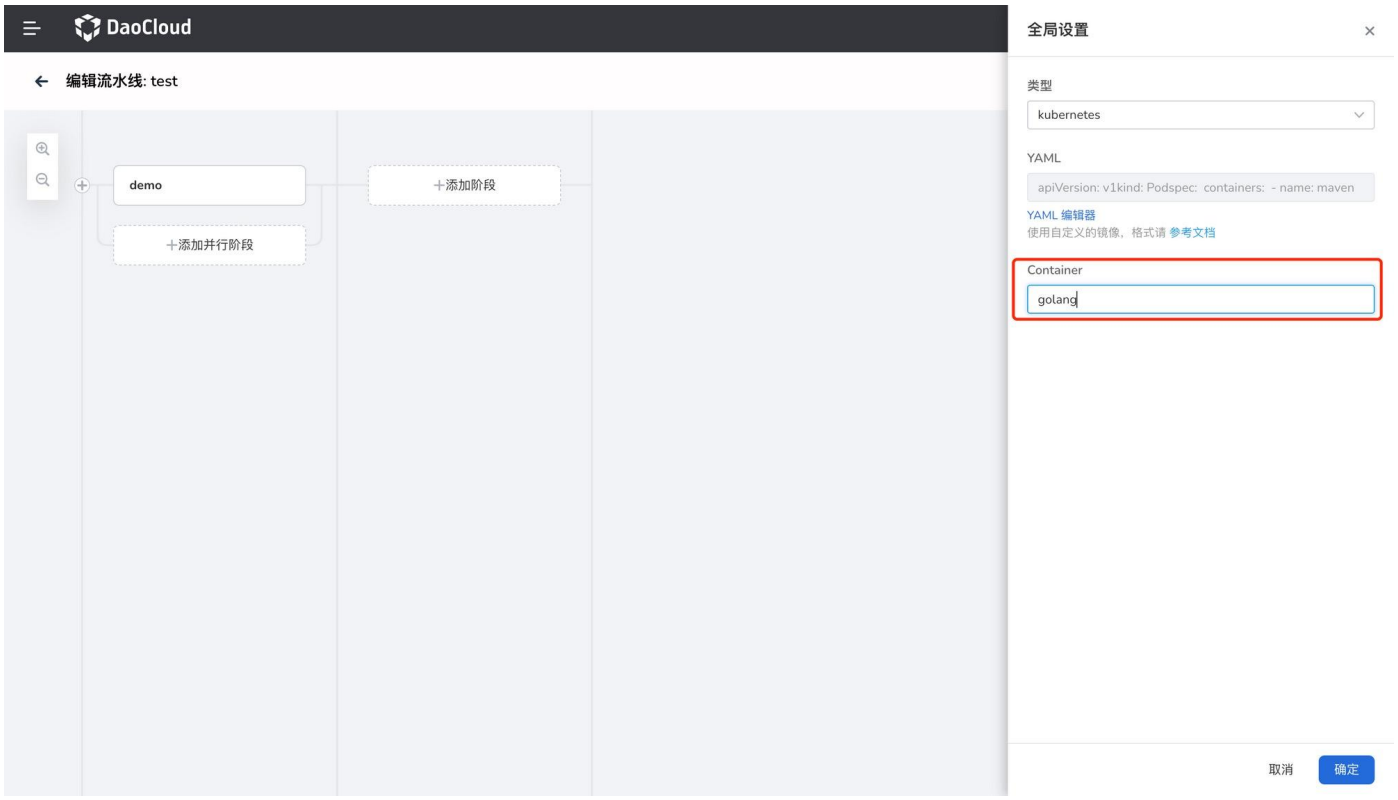


2. 点击 `YAML 编辑器`，在对话框中填写 YAML 语句，请参考以下示例：

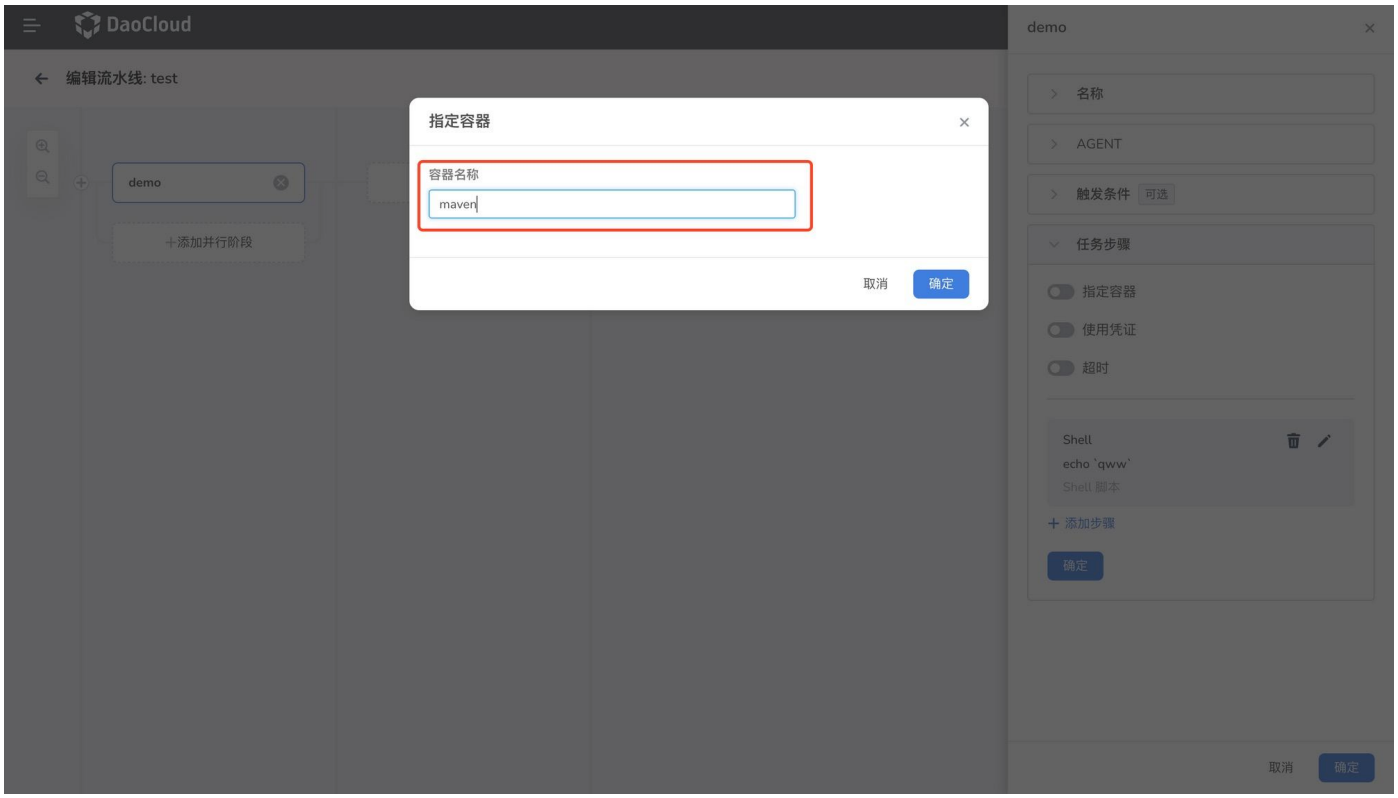
```bash
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: maven
    image: maven:3.8.1-jdk-8
    command:
    - sleep
    args:
    - 99d
  - name: golang
    image: golang:1.16.5
    command:
    - sleep
    args:
    - 99d
```

3. 在 Container 中输入 `golang` 作为流水线运行的默认容器。

Copyright © 2016 - 2023 DaoCloud



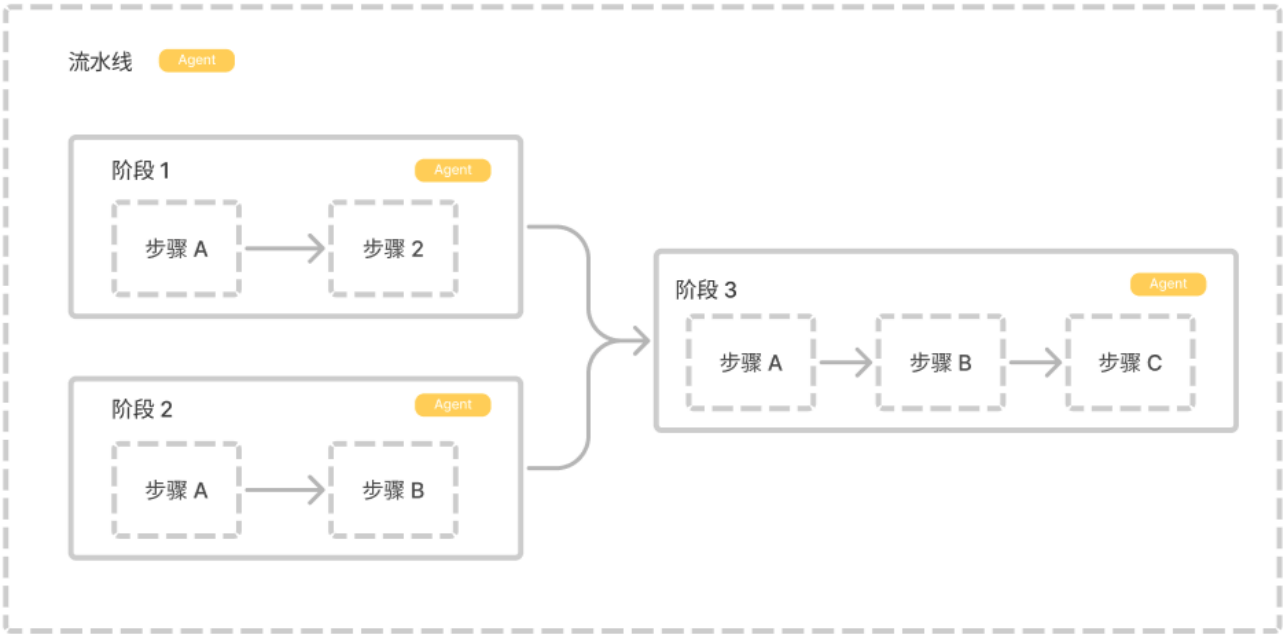
4. 要在流水线其他步骤中使用上述示例的其他容器，可以选择 指定容器 填写所需要的容器名称。



Copyright © 2016 - 2023 DaoCloud

流水线中的概念

不论图形化或文本类型，编辑器本质上都是用来方便用户查看与编辑构建流程的核心：Jenkinsfile（过程描述文件）。因此在讨论编辑器之前需理解「过程描述文件」的几个重要概念。



- 流水线

  `流水线` 是用户定义的一个工作模型，流水线的代码定义了软件交付的完整流程，一般包含构建、测试和交付应用程序等阶段。 有关流水线语法，请参见 Jenkins 官方文档。

- Agent

  Agent 描述了整个 `流水线` 执行过程或者某个 `阶段` 的执行环境，必须出现在 `描述文件` 顶格或者每一个 `阶段` 。 有关更多信息，请参见选择 Jenkins Agent。

- 阶段

  一个 `阶段` 定义了一系列紧密相关的 `步骤` 。每个 `阶段` 在整条流水线中各自承担了独立、明确的责任。 比如 “Build”、“Test” 或 “Deploy” 阶段。通常来讲，所有的实际构建过程都放置在 `阶段` 里面。 有关更多信息，请参见选择 Jenkins Stage。

- 并行阶段

  并行用来声明一些并行执行的 `阶段` ，通常适用于 `阶段` 与 `阶段` 之间不存在依赖关系的情况下，用来加快执行速度。 有关更多信息，请参见选择 Jenkins Agent。

- 步骤

  `步骤列表` 描述了一个 `阶段` 内具体要做什么事，具体要执行什么命令。比如有一个 `步骤（step）` 需要系统打印一条 `构建中…` 的消息，即执行命令 `echo '构建中...'` 。 有关更多信息，请参见选择 Jenkins Step。

Copyright © 2016 - 2023 DaoCloud

图形化任务模板参数说明

以下为图形化流水线任务模板参数说明。

GIT CLONE

参数	说明
代码仓库	填写远程代码仓库的地址。
分支	填写您想要被构建的代码分支，默认为 master。
凭证	如果您的远程仓库是私有的，请提前创建凭证并在使用时选择。

SHELL

如果要执行 shell 命令，请选择此模板，支持多行。

打印消息

如果需要在终端输出一些消息，请选择此模板。

保存制品

参数	说明
用于归档的文件	请填写期望存档文件路径的正则表达式，例如：`module/dist/*/.zip`，多个文件请用英文逗号隔开。

审核

如果添加此步骤，流水线运行至任务将会暂停，创建人和被 @ 的人可以选择继续或终止。

参数	说明
消息	流水线运行状态中将会展示此消息，支持在输入 @ + 用户的方式来选择可审核的人。

使用凭证

使用凭证组件在图形化编辑中是一个较为特殊的步骤，如果选择开启后，该阶段定义的所有步骤都将会被嵌套在使用凭证步骤中。

类型	说明
用户名和密码	选择用户名和密码类型后，您需要填写以下参数： 用户名变量：在流水线构建期间用户名的环境变量的名称。 密码变量：在流水线构建期间密码的环境变量的名称。
访问令牌	选择用访问令牌类型后，您需要填写以下参数： 文本变量：在流水线构建期间文本的环境变量的名称。
kubeconfig	选择用 kubeconfig 类型后，您需要填写以下参数： kubeconfig 变量：在流水线构建期间 kubeconfig 环境变量的名称。

超时

当前任务的执行时间若超过超时时间，则会中止任务，流水线变为失败状态。

参数	说明
时间	设置超时的时间。
单位	设置时间单位，支持秒、分钟、小时、天。

Copyright © 2016 - 2023 DaoCloud

5.4 Release

创建金丝雀发布任务

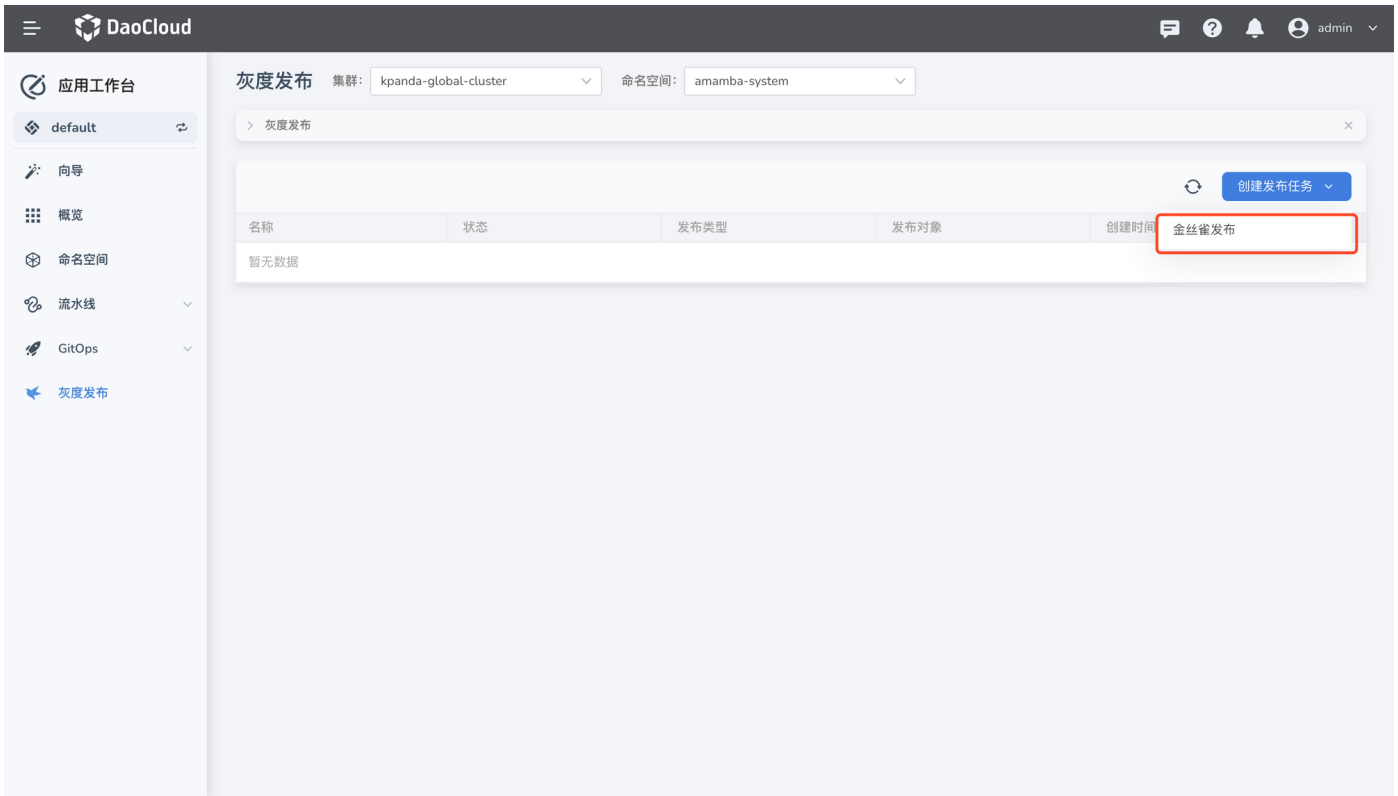
应用工作台基于开源项目 Argo Rollout 提供强大的灰度发布能力。灰度发布可以在不影响老版本的前提下，发布新的应用版本。按照预先定义的规则，渐进式地将流量切换到新版本，当新版本运行无问题后，最终自动将全部流量从老版本迁移至新版本。

前提条件

1. 创建一个工作空间和一个用户，该用户需加入该工作空间并具备 `Workspace Editor` 角色。
2. 创建一个应用并开启 灰度发布，可参考基于 Git 仓构建微服务应用、基于 Jar 包部署 Java 应用。
3. 发布对象所在的集群已经安装了 Istiod 和 Argo Rollout 两个组件。有关具体的安装方式，可参考管理 Helm 应用。

操作步骤

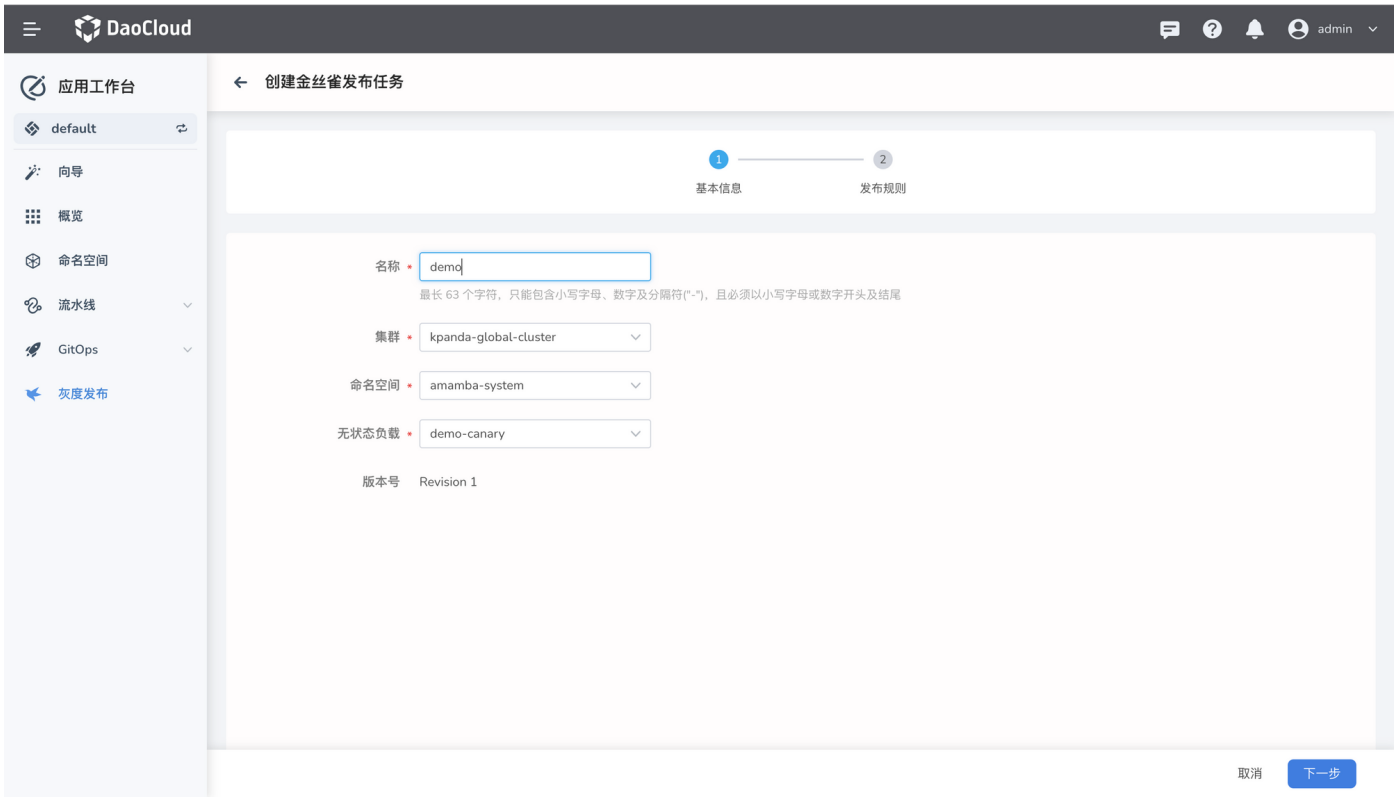
1. 进入 应用工作台 模块，在左侧导航栏点击 灰度发布，然后在页面右上角点击 创建发布任务 -> 金丝雀发布 。



Copyright © 2016 - 2023 DaoCloud

2. 参考下列要求填写基本信息，然后点击 `下一步`。

- 名称：填写发布任务的名称。最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾
- 集群：选择发布对象所在的集群。需要确保该集群已经部署了 Istio 和 Argo Rollout。
- 命名空间：选择发布对象所在的命名空间，且该命名空间中已经部署了开启了 `灰度发布` 的应用。
- 无状态负载：选择具体的发布对象。



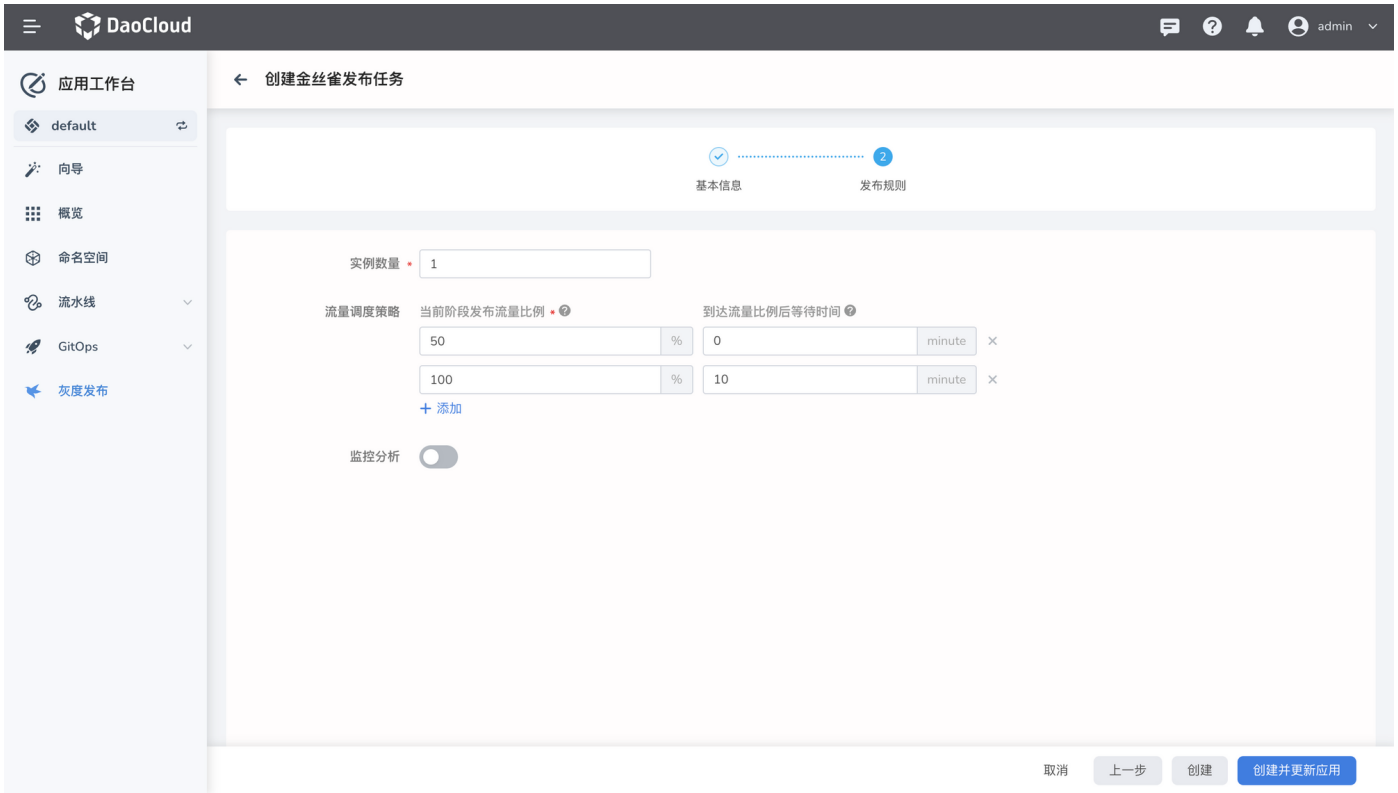
Copyright © 2016 - 2023 DaoCloud

3. 参考下列说明配置发布规则。

- 实例数量：进行灰度发布任务时应用的副本数量。
- 流量调度策略：
- 当前阶段发布流量比例：每个流量周期内，为灰度版本增加的流量比例。
- 到达流量比例后等待时间：为灰度版本增加流量周期，即等待多久后自动进入下一个灰度流量比例。

  如果设置为 0 或不填写，发布任务执行到此步骤时会永久暂停。

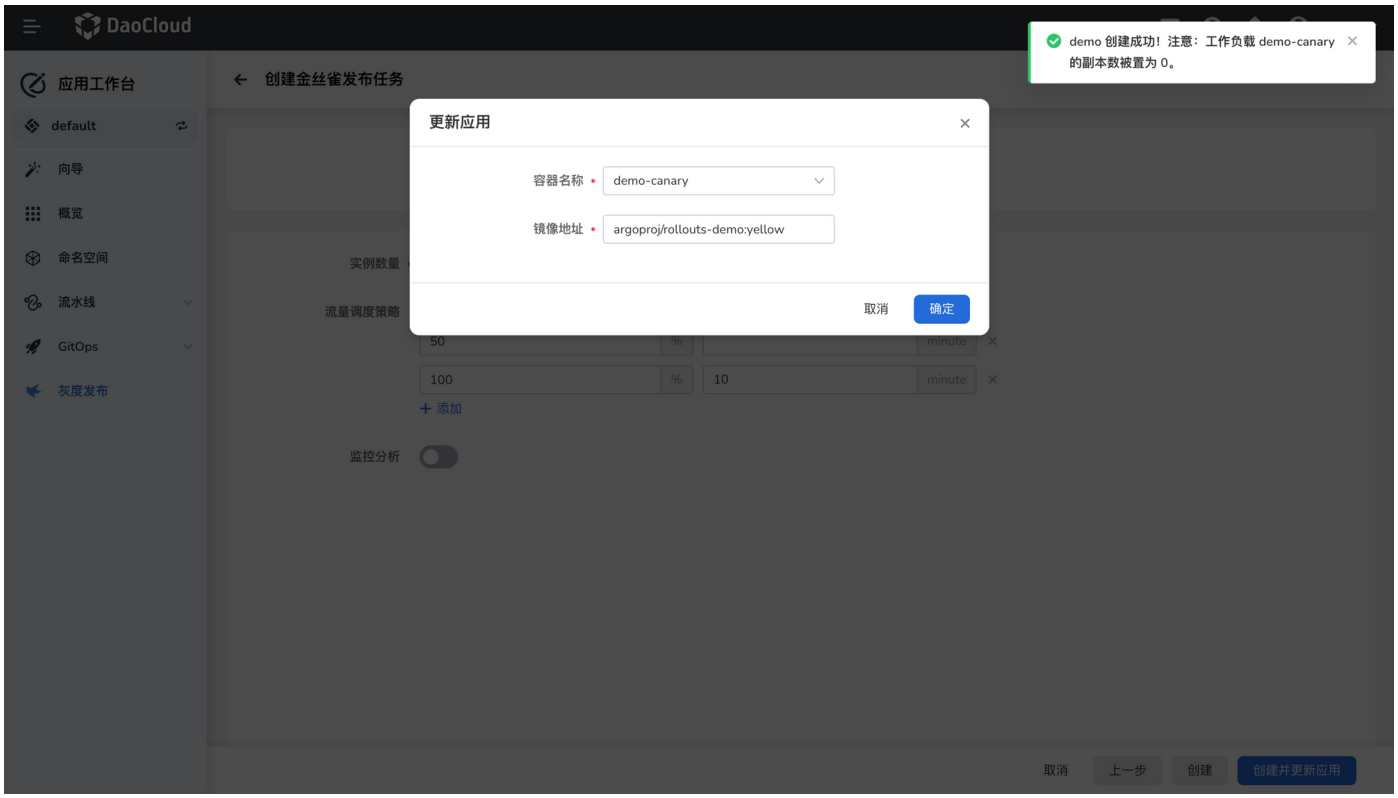
- 监控分析：开启后即可利用 Prometheus 的能力进行监控。根据预先定义的监控指标以及采集间隔，在整个发布过程中自动执行监控分析。如果不满足定义的规则，则自动回滚至旧版本，灰度发布失败。



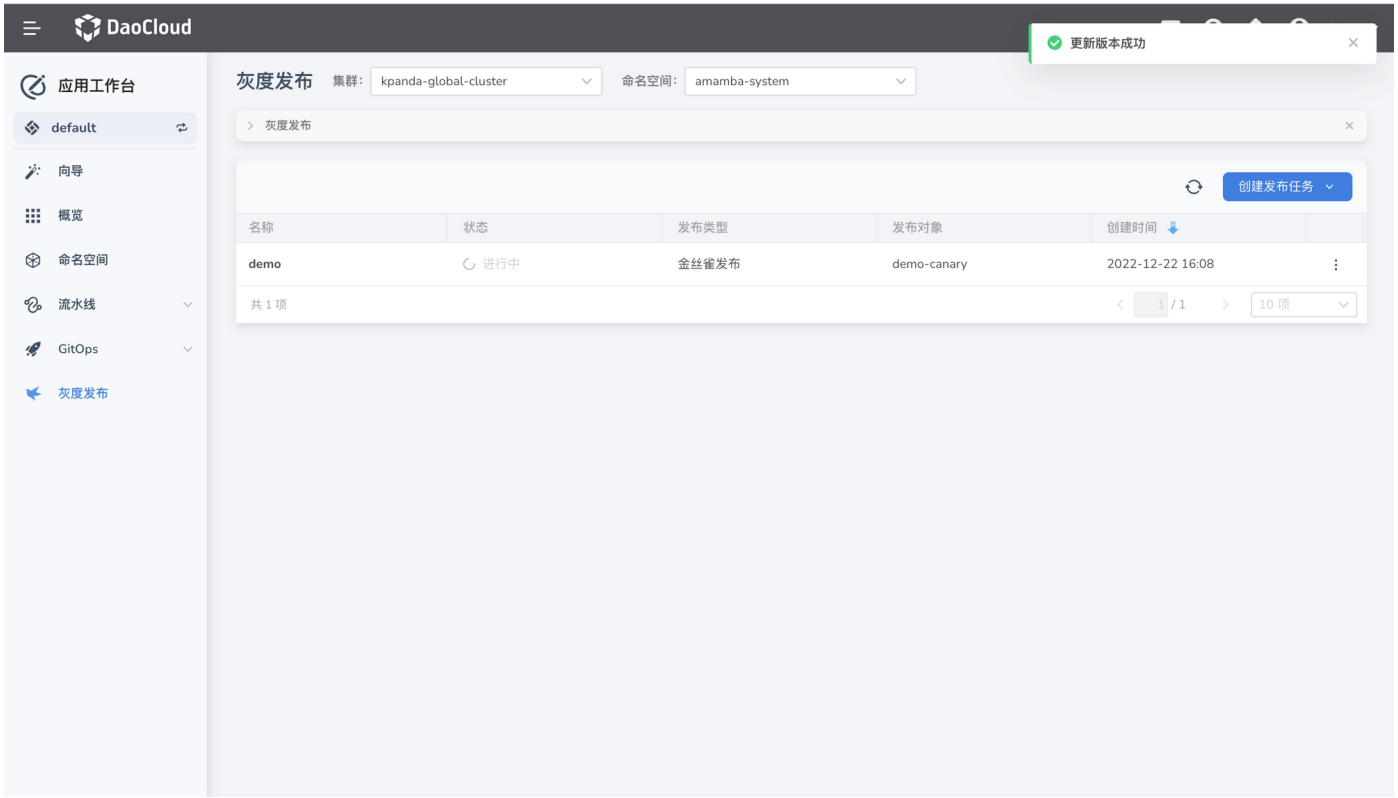
4. 在页面底部点击 创建并更新应用 ，然后在弹框中设置灰度版本的镜像地址并点击 确定 。

   此时原工作负载的副本数会被设置为 0。

Copyright © 2016 - 2023 DaoCloud



5. 系统自动跳转至灰度发布的任务列表页面，提示 更新版本成功 。



Copyright © 2016 - 2023 DaoCloud

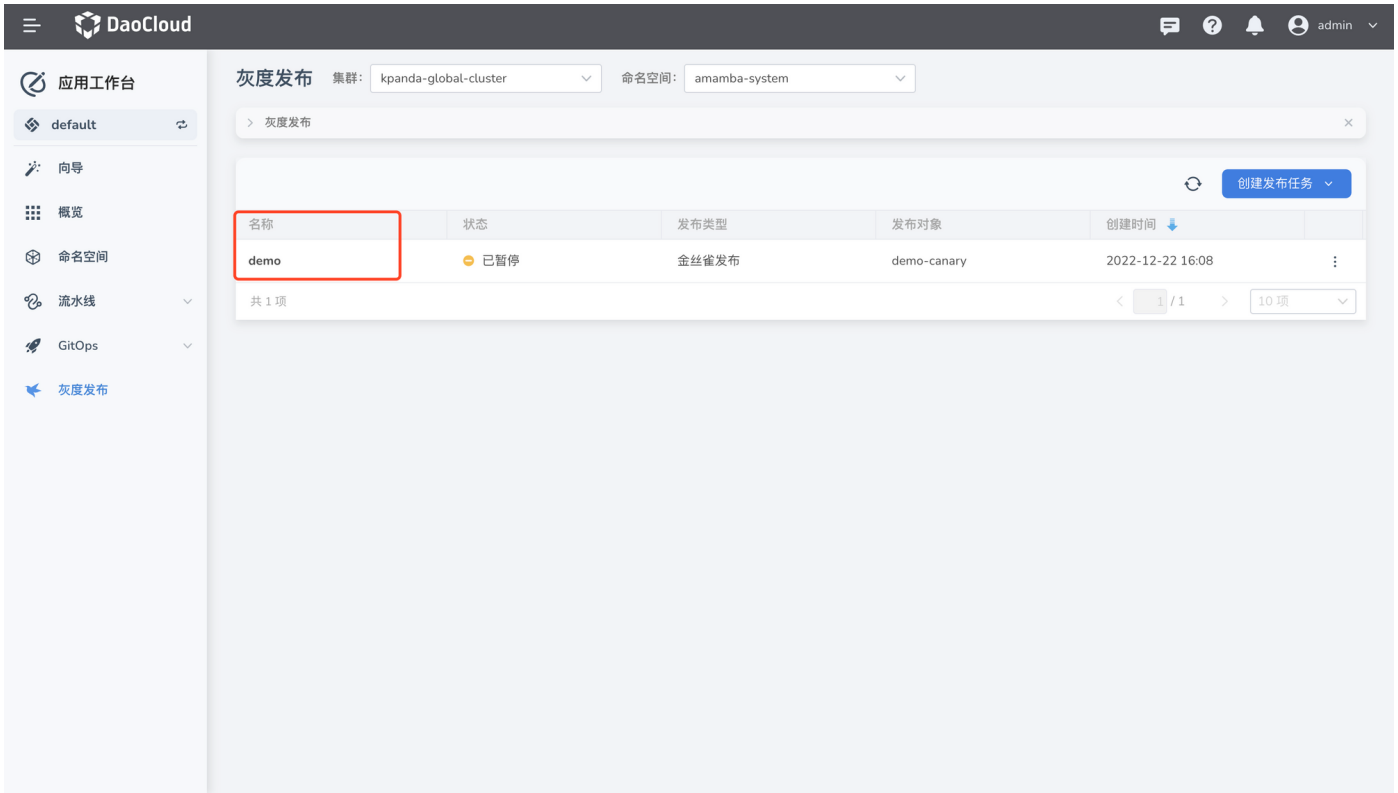
查看金丝雀发布任务

创建灰度发布任务并关联工作负载后，修改工作负载的镜像、资源配置、启动参数等使得容器组重启时，会自动触发灰度发布任务更新版本，并按照定义好的发布规则进行流量调度。

本文主要介绍查看金丝雀发布任务时涉及到的相关操作，例如查看任务详情、更新版本、更新发布任务、回滚版本等。

查看任务详情

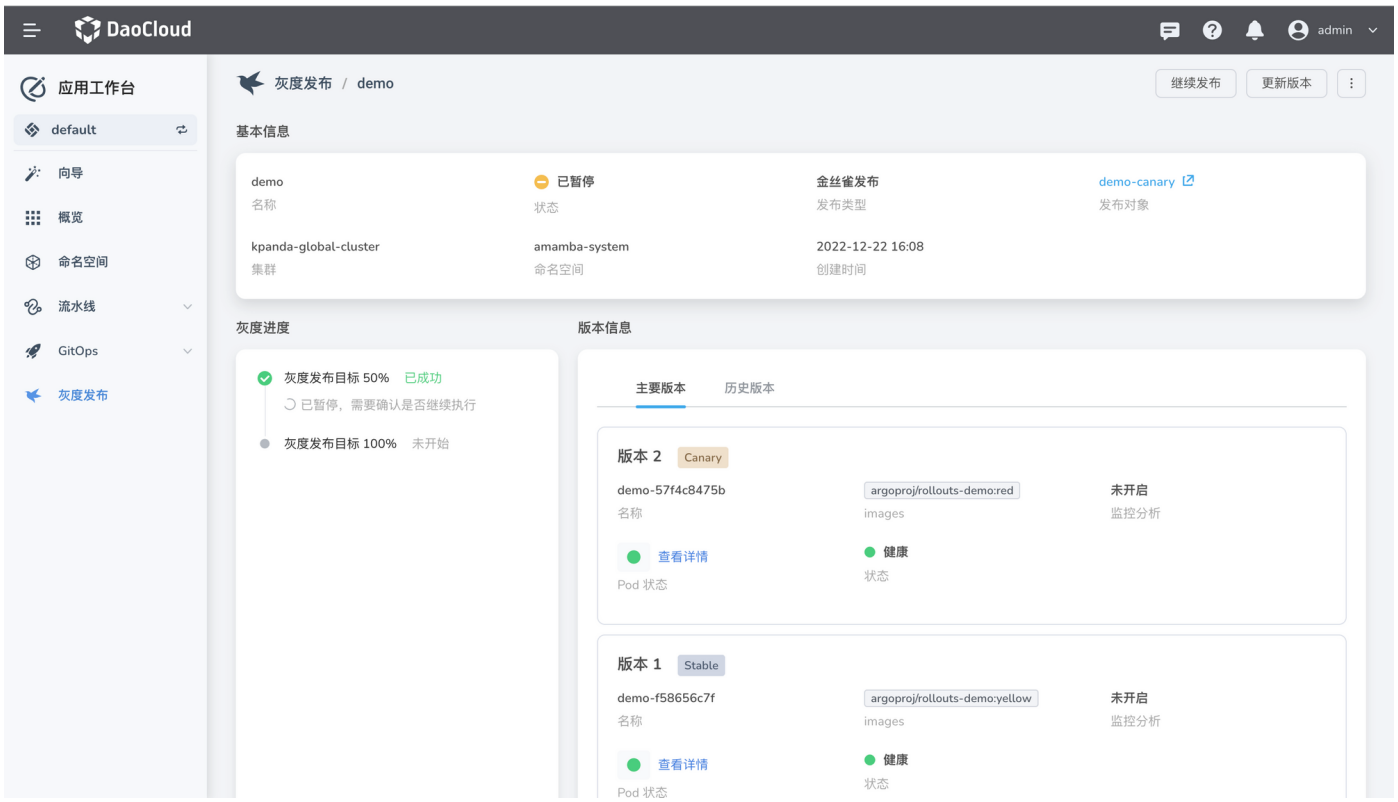
1. 进入 应用工作台 模块，在左侧导航栏点击 灰度发布 ，点击目标任务的名称。



Copyright © 2016 - 2023 DaoCloud

2. 查看灰度发布任务详情页面。

- 基本信息 区域：查看任务的名称、状态、发布类型、发布对象等信息。
- 灰度进度 区域：以可视化的形式展示灰度发布任务的执行进度，可以直观地了解任务目前执行到了哪一步，以及执行的状态如何。
- 版本信息 区域：
- 主要版本：展示当前版本以及金丝雀版本信息。
- 历史版本：展示历史版本记录。



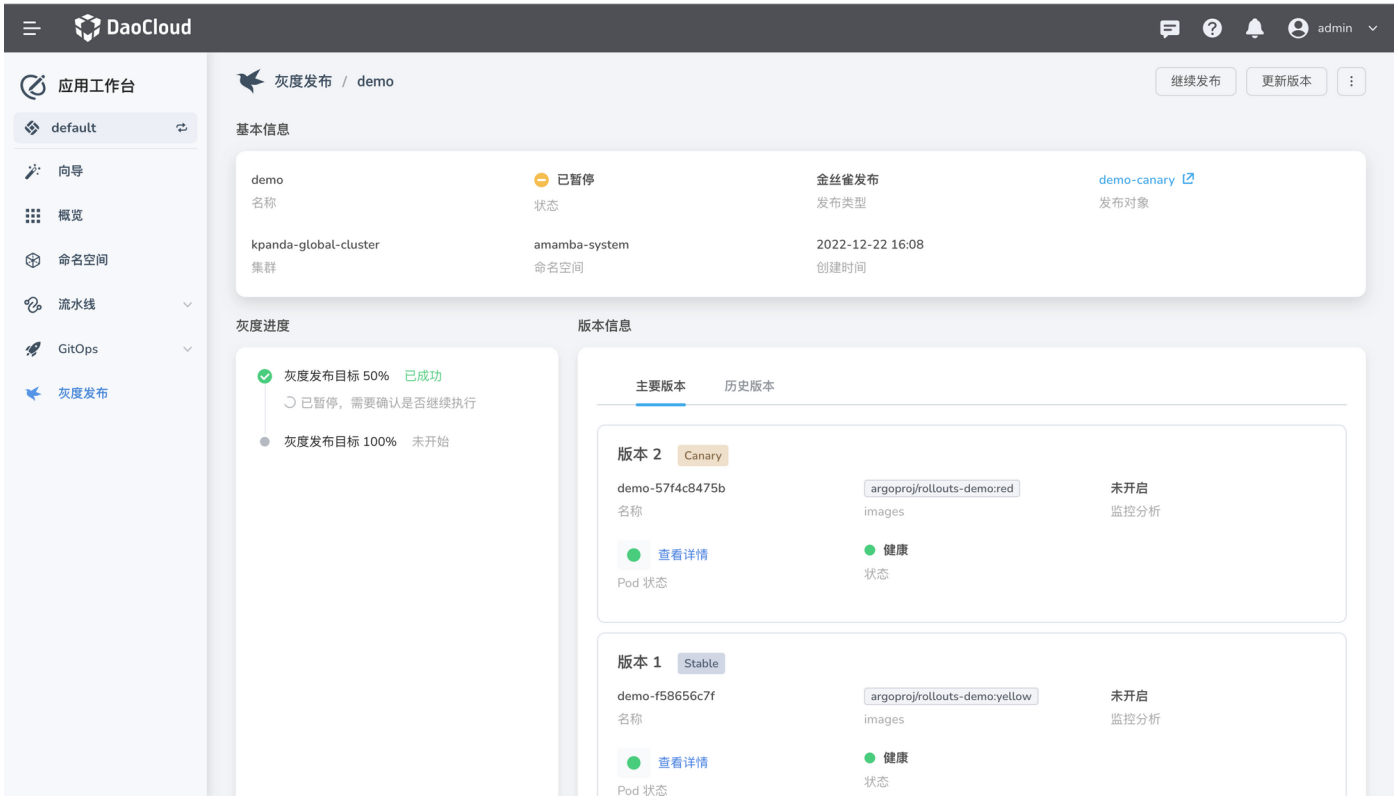
Copyright © 2016 - 2023 DaoCloud

更新版本

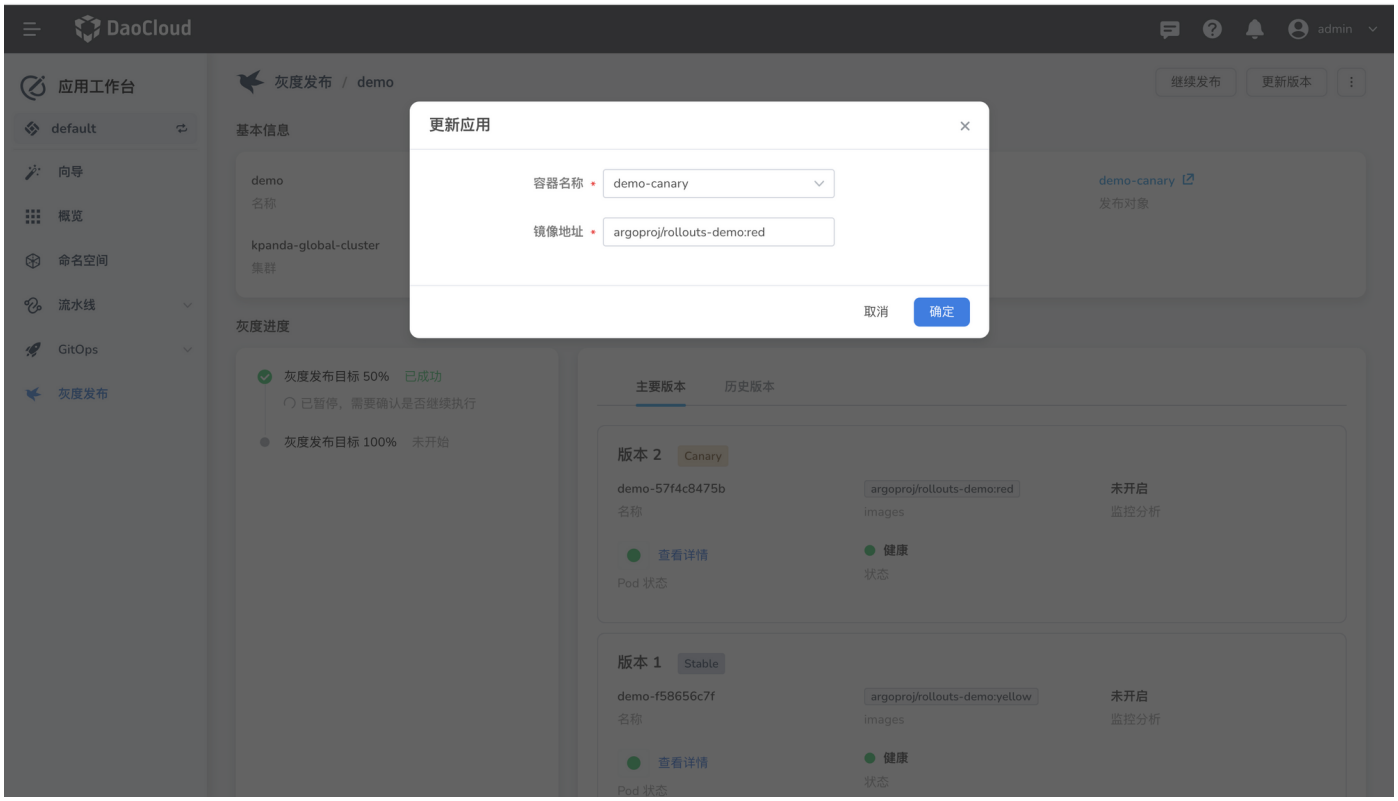
发布对象更新版本后，会自动触发灰度发布任务。

Copyright © 2016 - 2023 DaoCloud

1. 点击目标任务的名称，然后在右上角点击 更新版本 。

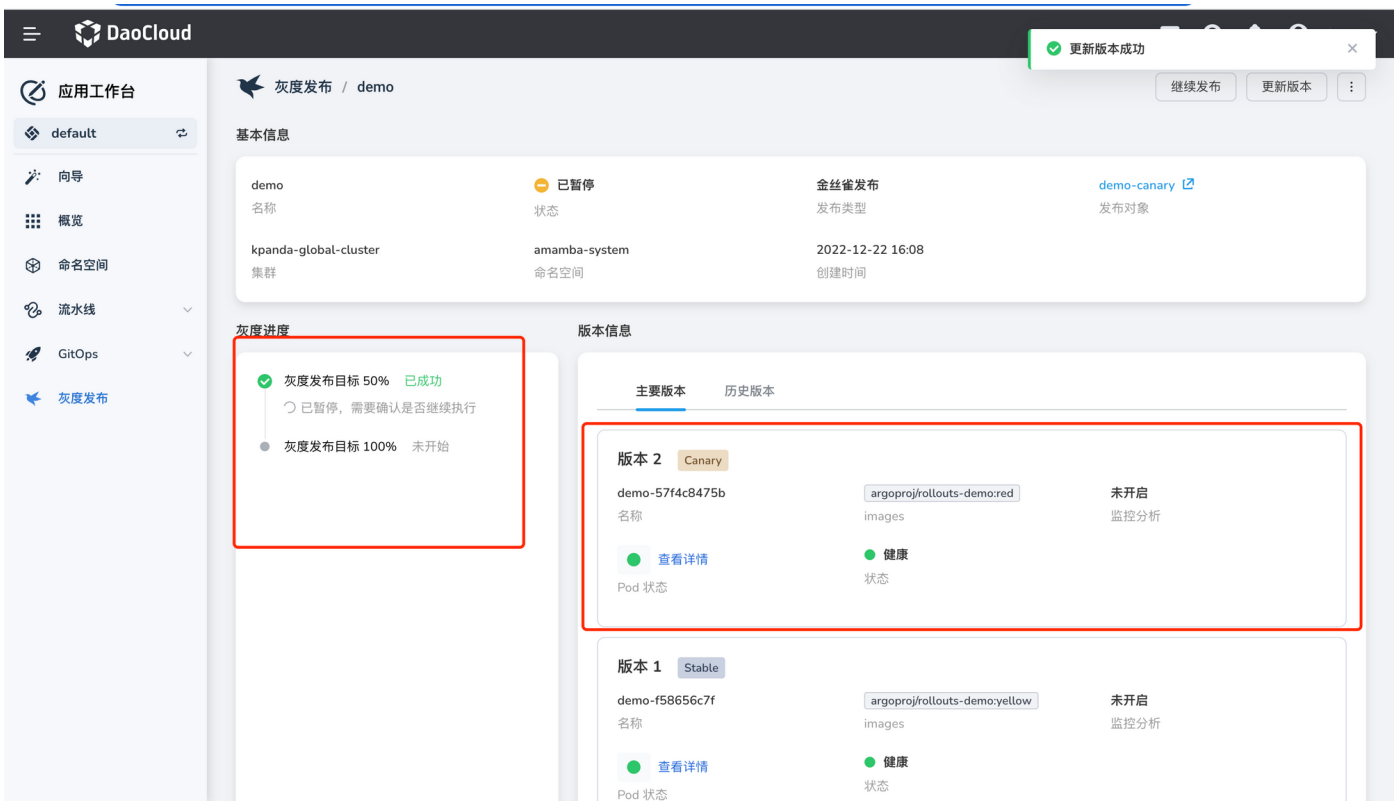


2. 设置金丝雀发布的镜像。



3. 应用更新成功后会触发一次新的灰度发布过程。

Copyright © 2016 - 2023 DaoCloud

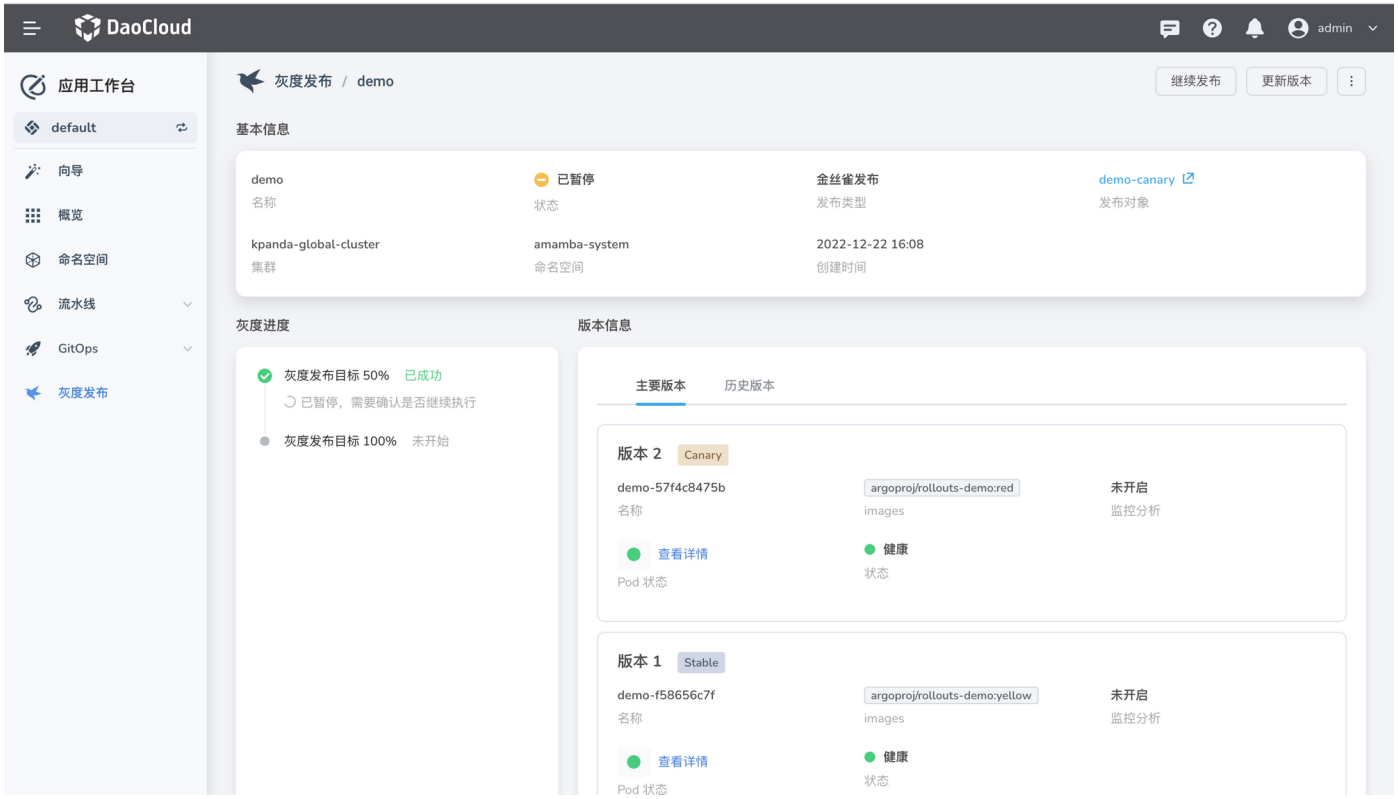


Copyright © 2016 - 2023 DaoCloud

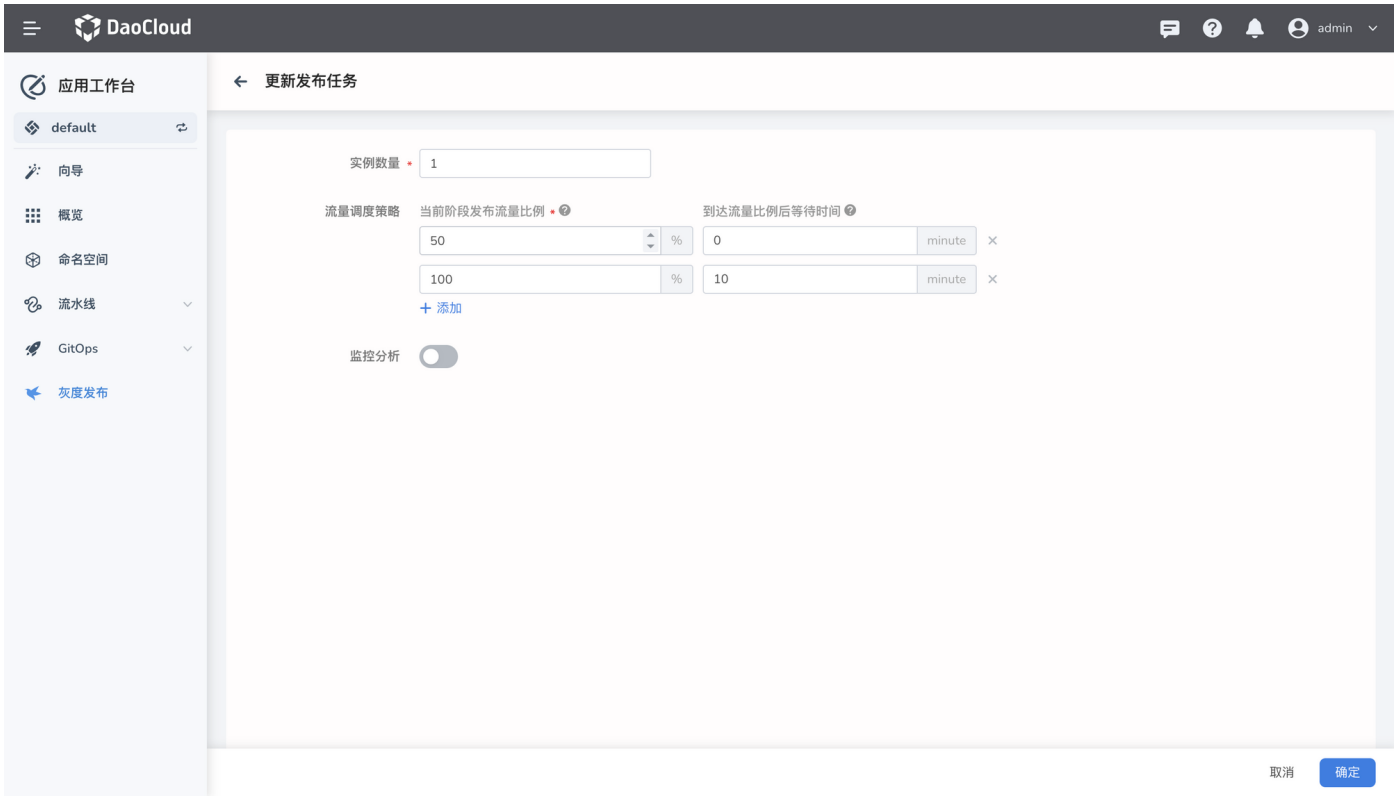
更新发布任务

通过更新发布任务，可以修改灰度发布过程的流量调度策略。

1. 在 灰度发布任务 详情页面，在页面右上角点击 ⋮ 并选择 更新发布任务 。



2. 调整发布规则并点击 确定 。



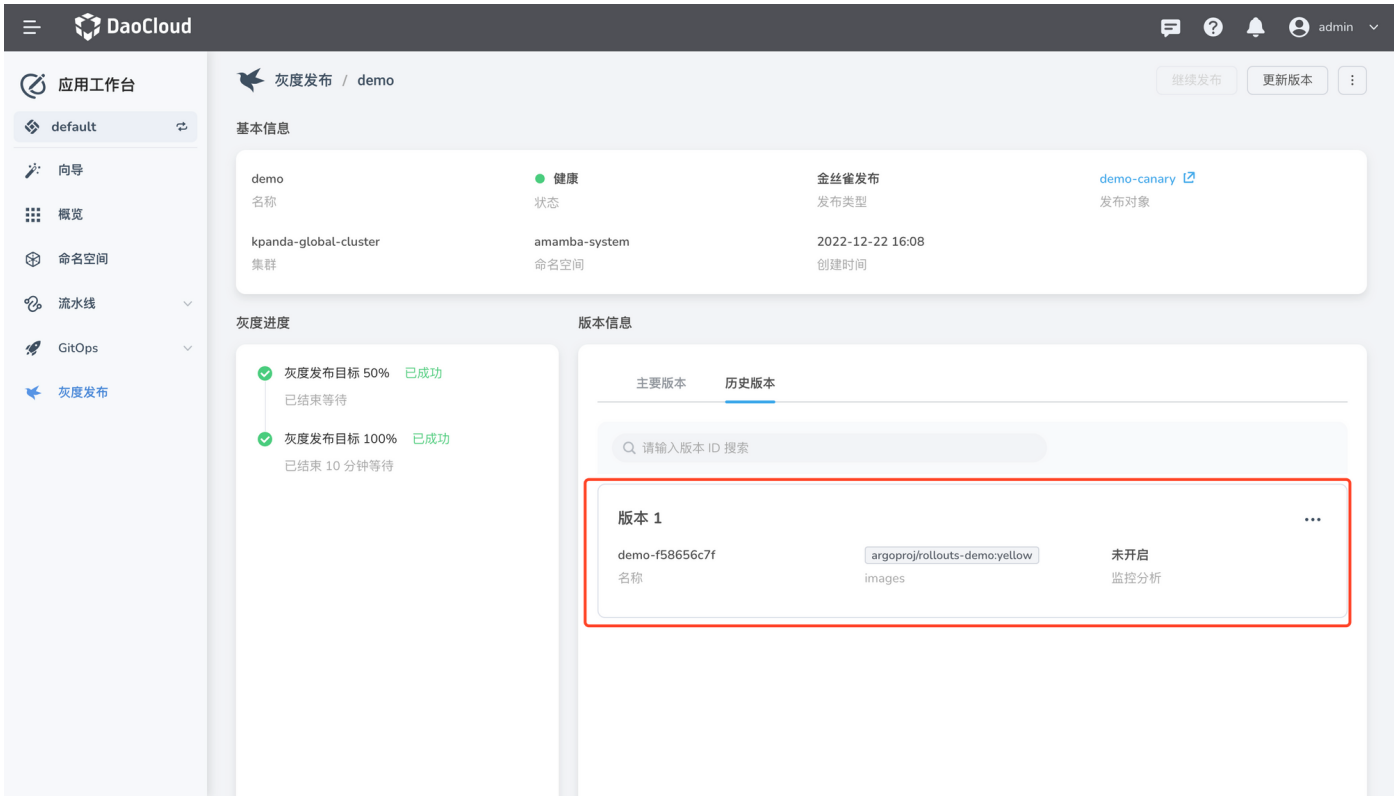
Copyright © 2016 - 2023 DaoCloud

回滚

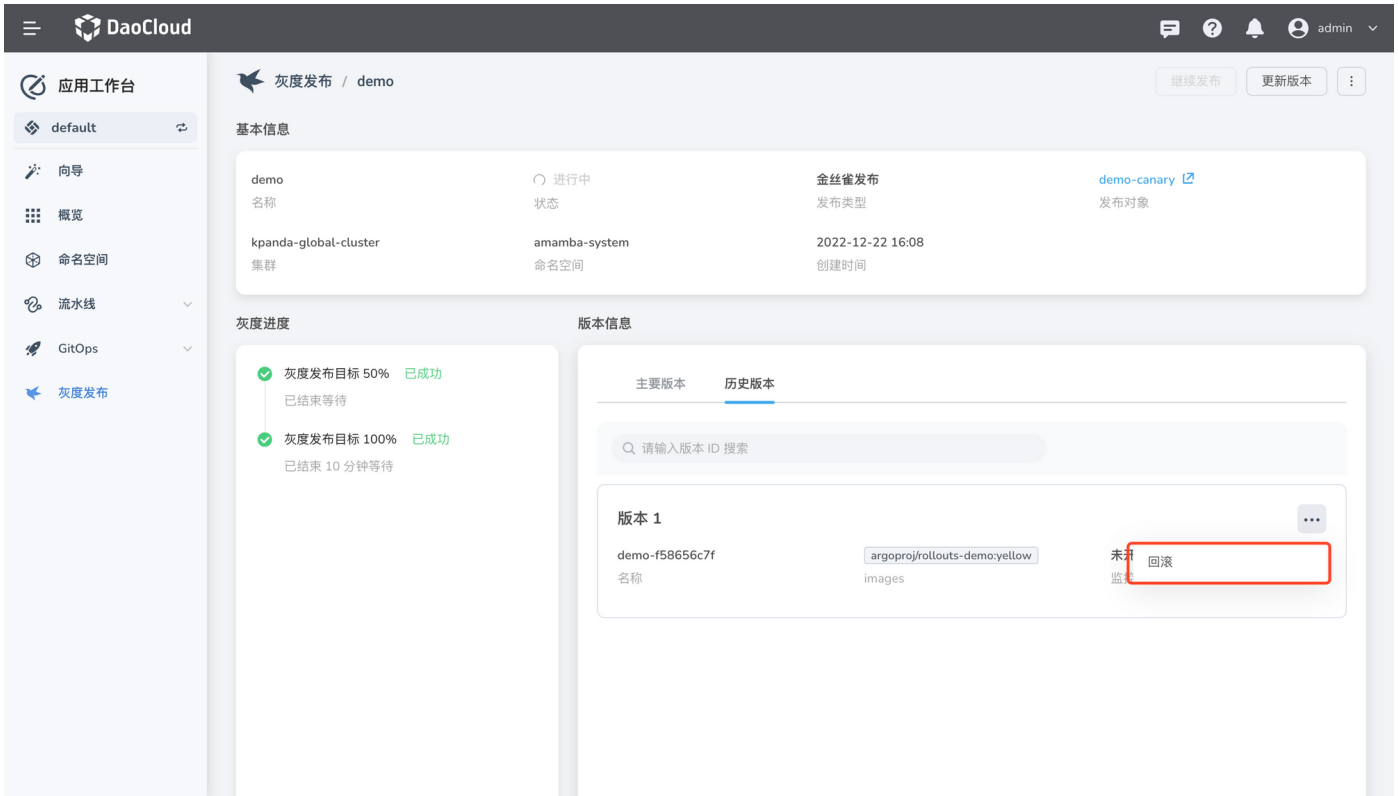
支持查看以往发布过的历史版本，可以一键回滚到之前的某个版本。

Copyright © 2016 - 2023 DaoCloud

1. 在 灰度发布任务 详情页面，点击 历史版本 标签。

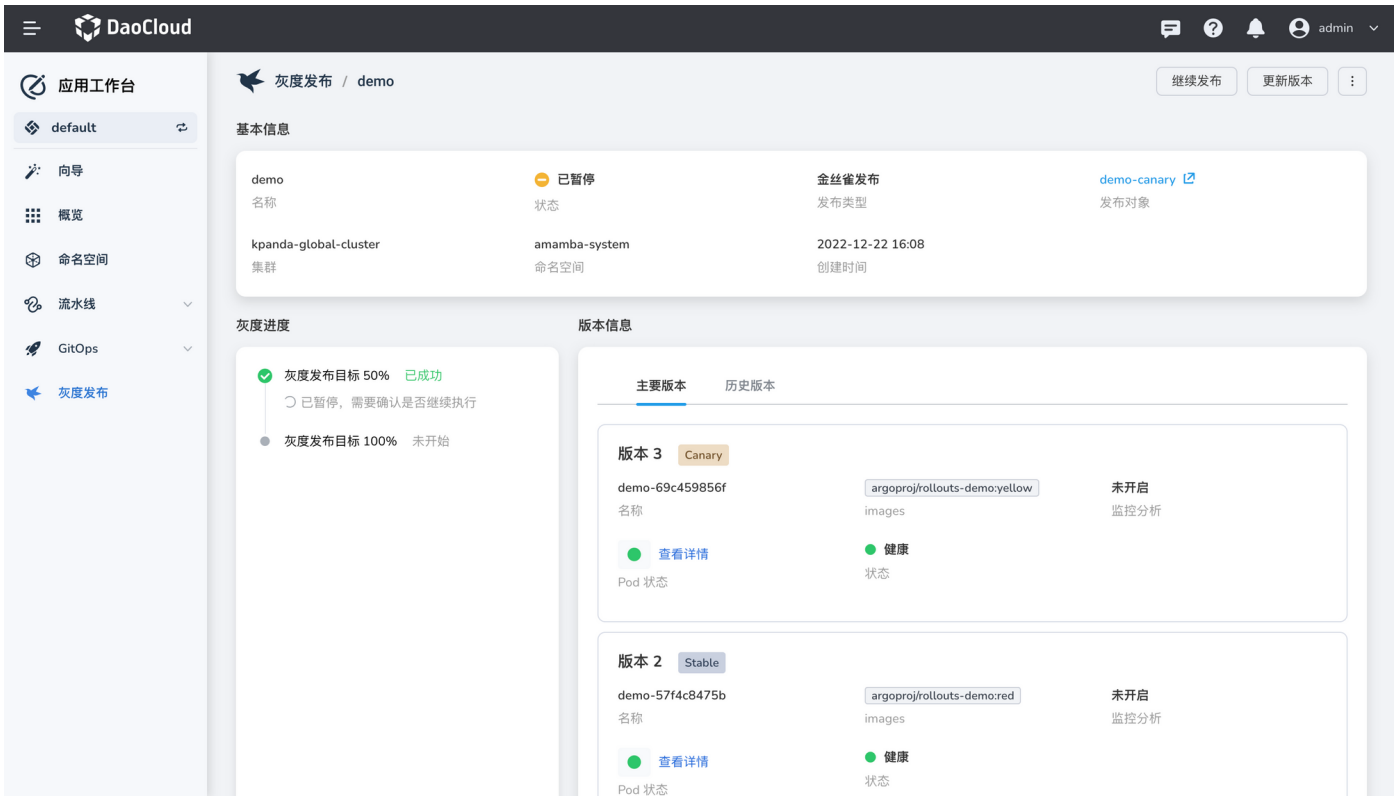


2. 选择目标版本，点击 回滚 。



3. 回滚成功后会触发一次新的灰度发布过程。

Copyright © 2016 - 2023 DaoCloud



其他操作说明

操作	
继续发布	对于正在进行中或者暂停的金丝雀发布任务，继续发布可以推动金丝雀发布过程进入到下一阶段的发布。
发布	对于正在进行中或者暂停的金丝雀发布任务，发布可以推动金丝雀发布过程直接完成发布，将金丝雀版本更新为稳定版本。
终止发布	对于正在进行中或者暂停的金丝雀发布任务，终止发布会暂停当前所有步骤，并回滚到稳定版本。

Copyright © 2016 - 2023 DaoCloud

5.5 Wizard

基于 Git 仓构建微服务应用

应用工作台支持通过 Git 仓库、Jar 包、容器镜像、Helm 模板等四种方式构建应用。本文介绍如何通过 Git 仓库源码构建传统微服务应用，从而对应用进行流量治理，查看日志、监控、链路追踪等功能。

前提条件

- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 `workspace edit` 角色。 参考创建工作空间、用户和角色。
- 创建可以访问代码仓库仓库、镜像仓库的两个凭证，参考凭证管理。
- 准备一个 Gitlab 仓库、Harbor 仓库。

创建凭证

1. 在 `凭证` 页面创建两个凭证：
- `git-credential`：用户名和密码，用于访问代码仓库
- `registry-credential`：用户名和密码，用于访问镜像仓库
2. 创建完成后，可以在 `凭证列表` 页面看到凭证信息。

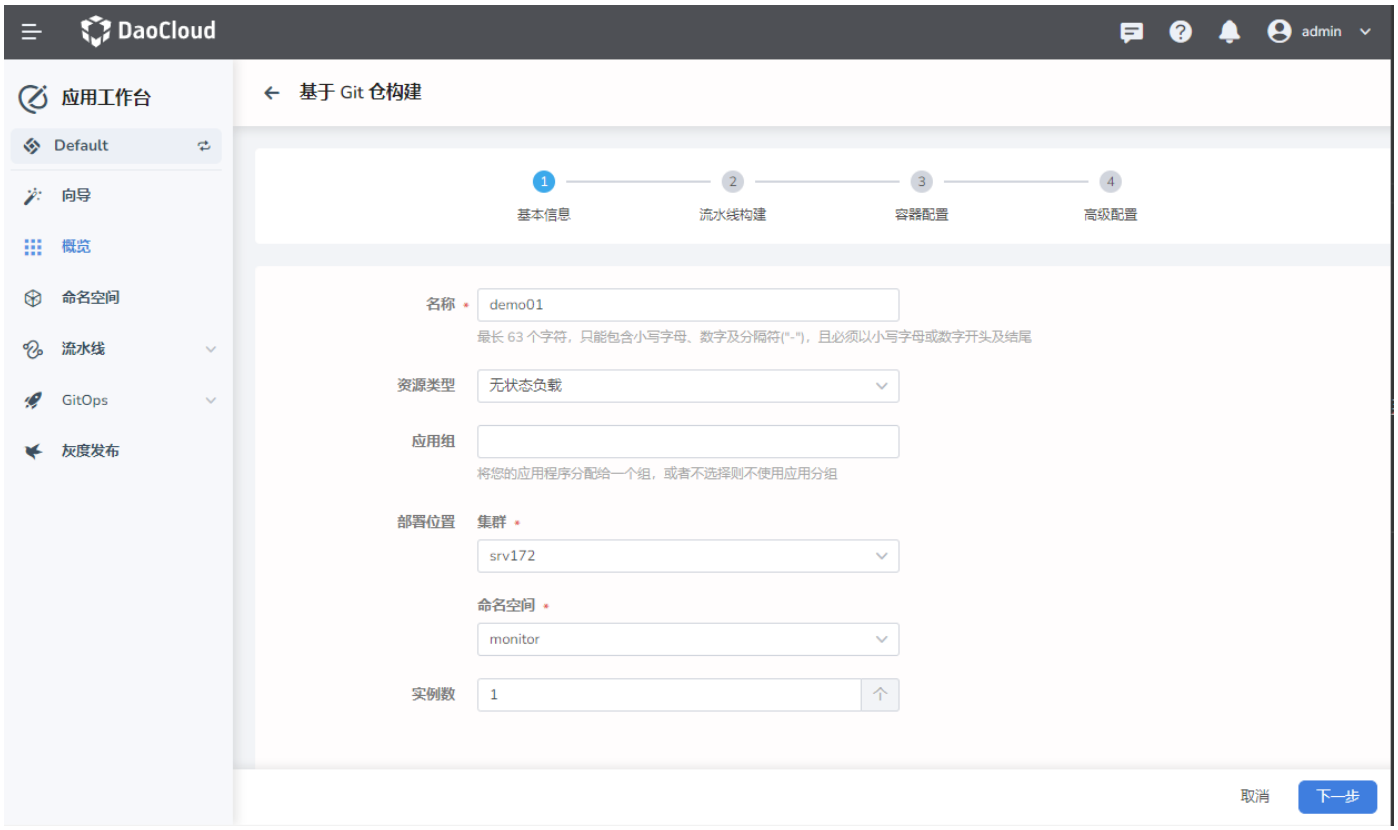
基于 Git 创建微服务应用

1. 在 `应用工作台` -> `向导` 页面中，点击 `基于 Git 仓构建` 。

Copyright © 2016 - 2023 DaoCloud

2. 参考以下说明填写基本信息，然后点击 `下一步`：

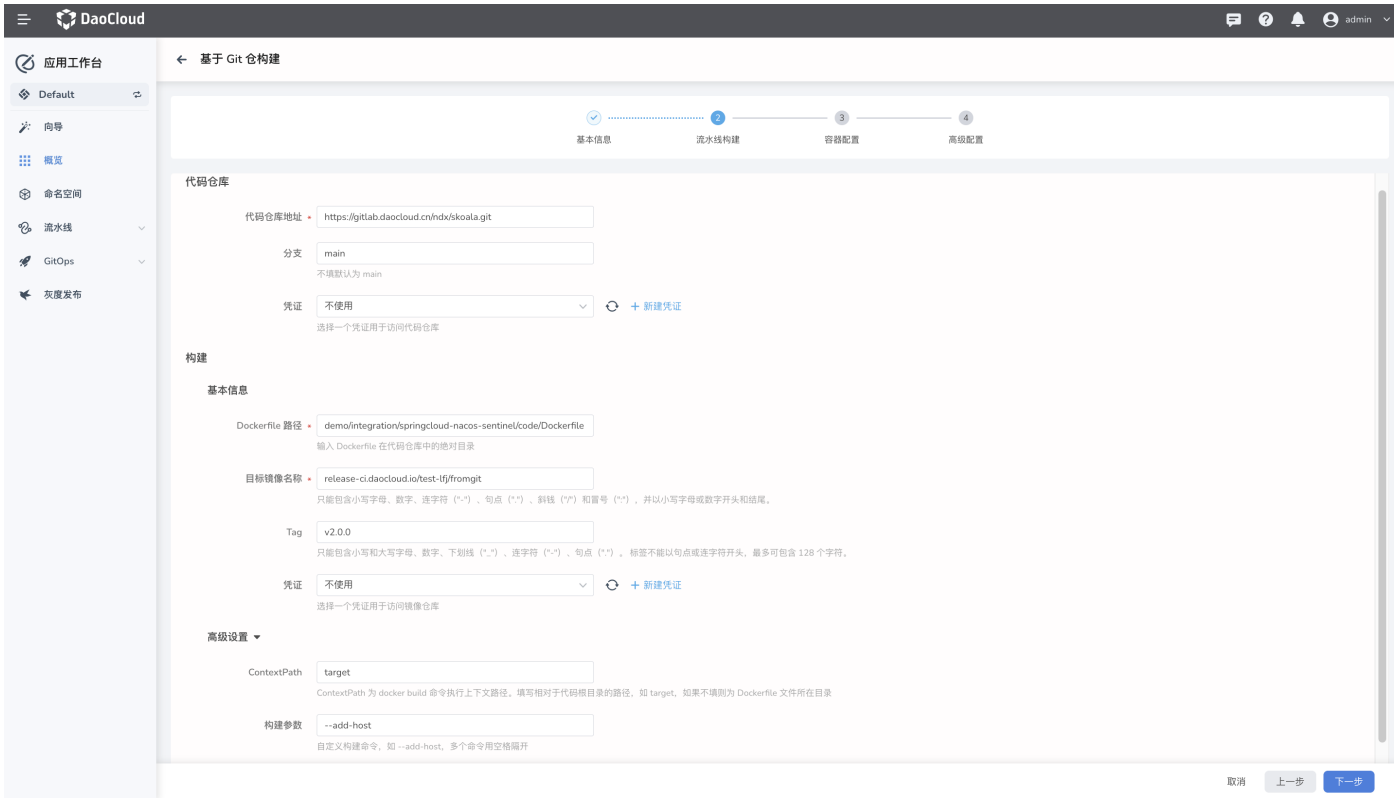
- 名称：填写应用的名称。
- 资源类型：支持无状态负载、有状态负载，本演示选择无状态负载。
- 输入或选择应用组。
- 部署位置：选择将应用部署到哪个集群下的哪个命名空间。如果要接入微服务，请确保当前工作空间下已经创建了注册中心。
- 实例数：填写实例的数量，Pod 的数量。



Copyright © 2016 - 2023 DaoCloud

3. 参考以下说明填写流水线配置，然后点击 `下一步` 。

- 代码仓库：输入 Git 仓库地址，例如 `https://gitlab.daocloud.cn/ndx/skoala.git` 。在实际操作中请使用自己的仓库地址。
- 分支：默认为 `main` ，此处为 `main` ，无需更改。
- 凭证：选择访问代码仓库的凭证 `git-credential` ，如果为公开仓库，则无需填写。
- Dockerfile 路径：输入 Dockerfile 在代码仓库中的绝对路径，例如 `demo/integration/springcloud-nacos-sentinel/code/Dockerfile` 。
- 目标镜像名称：输入镜像仓库名称，例如 `release-ci.daocloud.io/test-lfj/fromgit` 。
- Tag：输入镜像仓库版本，例如 `v2.0.0` 。
- 凭证：选择访问镜像仓库的凭证，例如 `registry-credential` 。
- ContextPath：ContextPath 为 docker build 命令执行上下文路径。填写相对于代码根目录的路径，如 target，如果不填则为 Dockerfile 文件所在目录。
- 构建参数：构建参数会以 --build-arg 的形式传递到 build 命令的参数中，支持将上游制品下载地址、上游镜像下载地址设置为参数，同时支持自定义任意参数。



Copyright © 2016 - 2023 DaoCloud

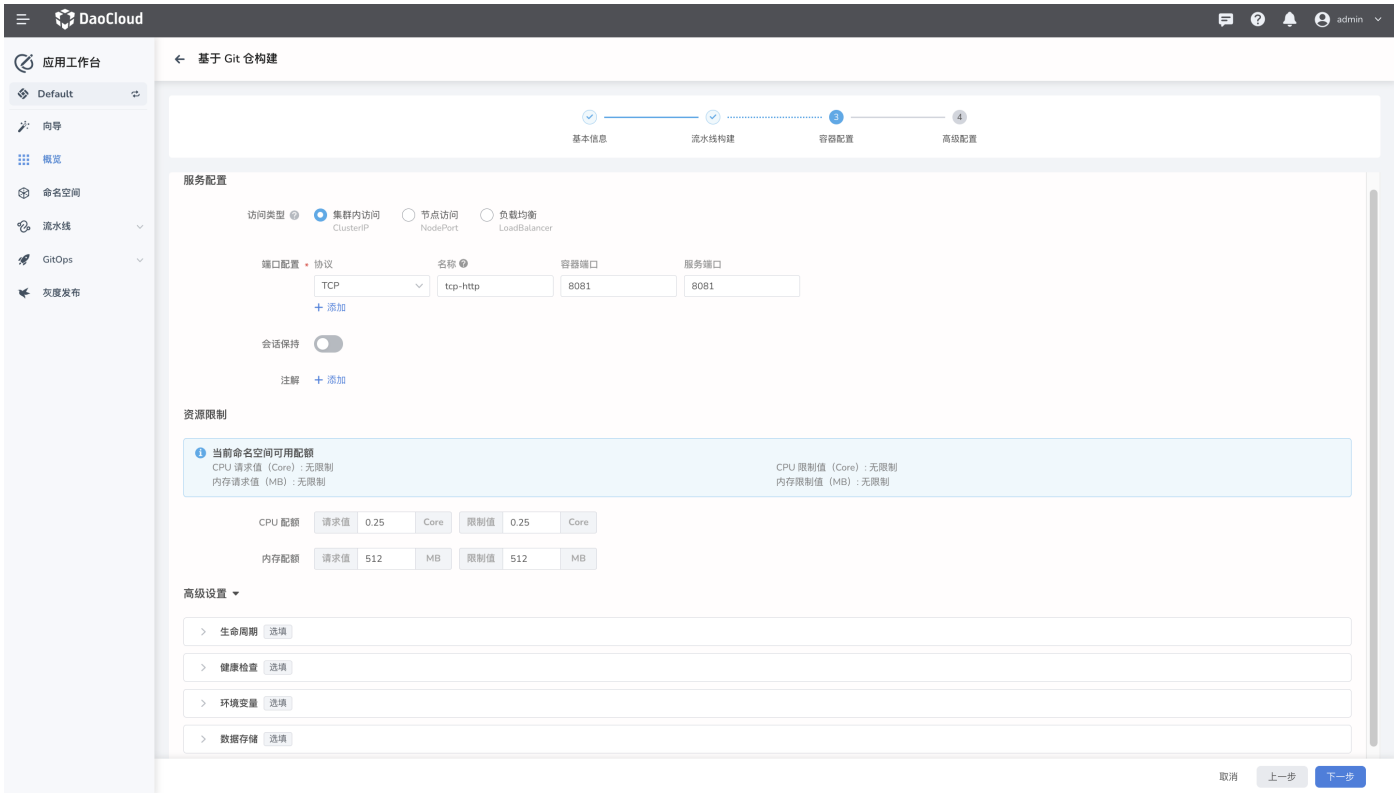
4. 参考以下说明填写容器配置，然后点击 `下一步`。

- 服务配置：支持集群内访问、节点访问、负载均衡。示例值如下：

```
- name: http            protocol: TCP      port: 8081      targetPort: 8081
- name: health-http     protocol: TCP      port: 8999      targetPort: 8999
- name: service         protocol: TCP      port: 9555      targetPort: 9555
```

  有关服务配置的更多详细说明，可参考创建服务。

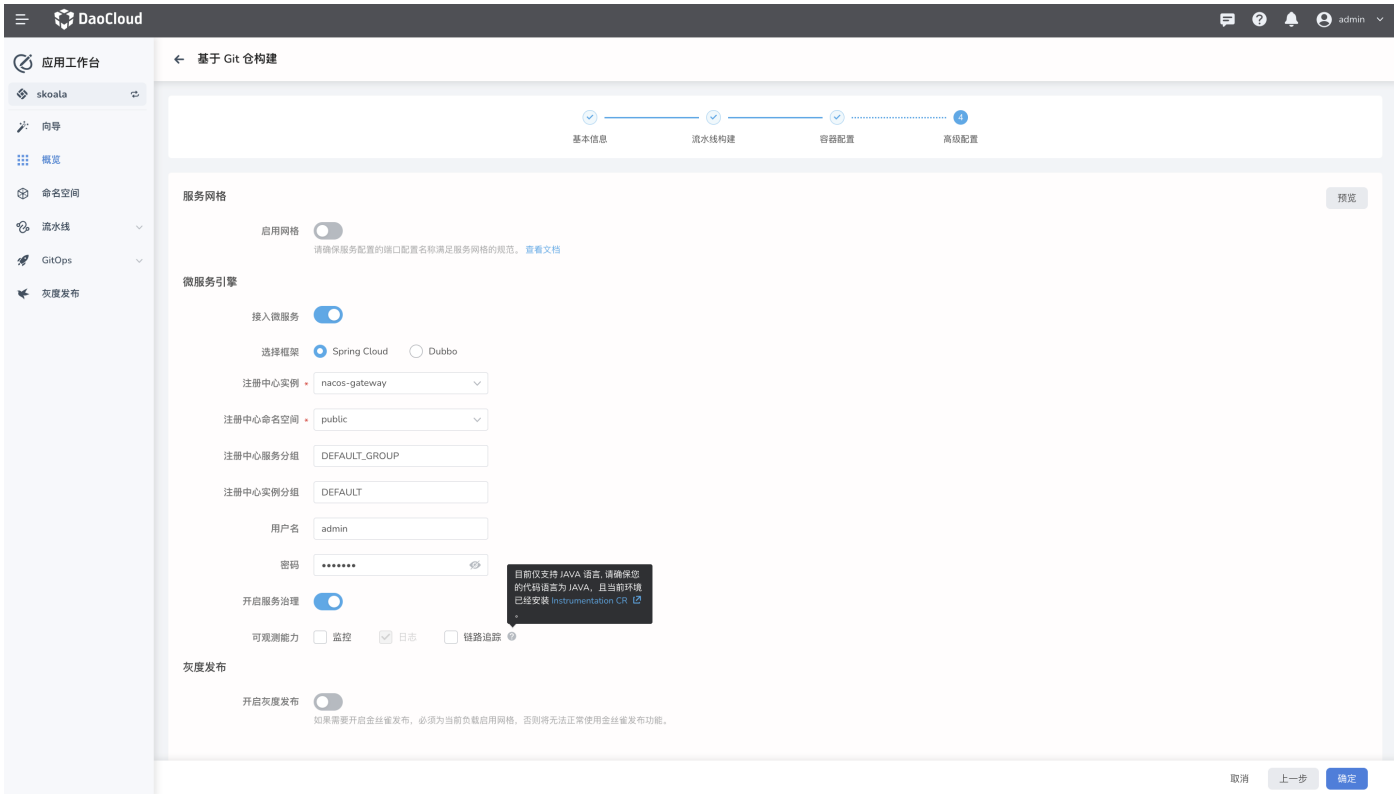
- 资源限制：指定应用能使用的资源上限，包括 CPU、内存。
- 生命周期：设置容器启动时、启动后、停止前需要执行的命令。详情可参考容器生命周期配置。
- 健康检查：用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考容器健康检查配置。
- 环境变量：配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考容器环境变量配置。
- 数据存储：配置容器挂载数据卷和数据持久化的设置。



Copyright © 2016 - 2023 DaoCloud

5. 在 高级配置 页面点击 开启接入微服务 ，参考以下说明配置参数，然后点击 确定 。

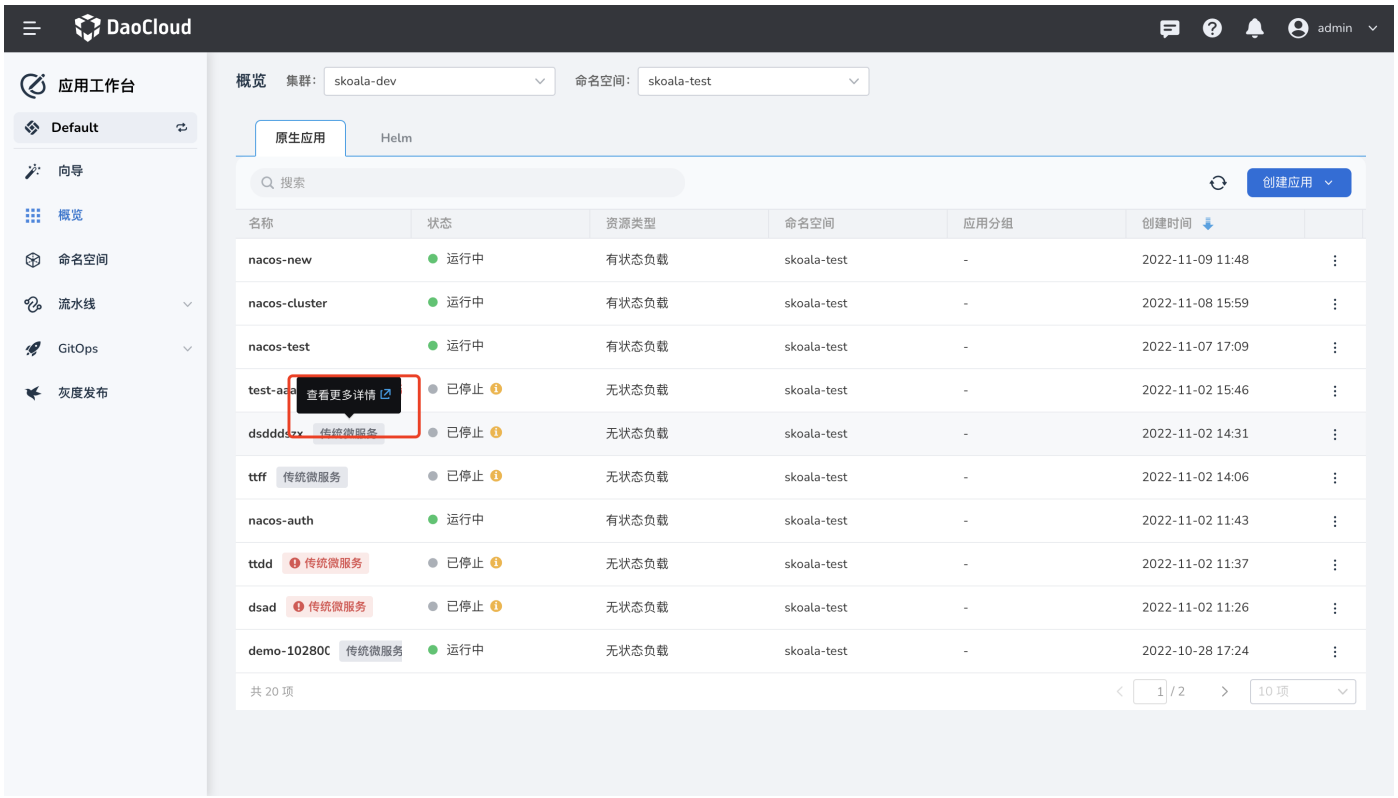
- 选择框架：支持 Spring Cloud 、 Dubbo ，此处选择 Spring Cloud 。
- 注册中心实例：目前仅支持选择微服务引擎中的托管 Nacos 注册中心实例。
- 注册中心命名空间：微服务应用的 nacos 命名空间
- 注册中心服务分组：微服务应用的服务分组
- 用户名/密码：如果该注册中心实例被认证，则需要填写用户名密码
- 开启微服务治理：所选择的注册中心实例应开启了 Sentinel 或 Mesh 治理插件
- 监控：选择开启，开启后可查看服务相关监控信息
- 日志：默认开启
- 链路追踪：开启后可查看服务的链路追踪信息，目前仅支持 Java 语言



查看并访问微服务相关信息

1. 在左侧导航栏点击 概览 ，在 原生应用 页签中，光标悬浮在某一个应用上，点击悬浮菜单 查看更多详情 。

Copyright © 2016 - 2023 DaoCloud



2. 跳转到微服务引擎，查看服务详情。

Copyright © 2016 - 2023 DaoCloud

基于 Jar 包部署 Java 应用

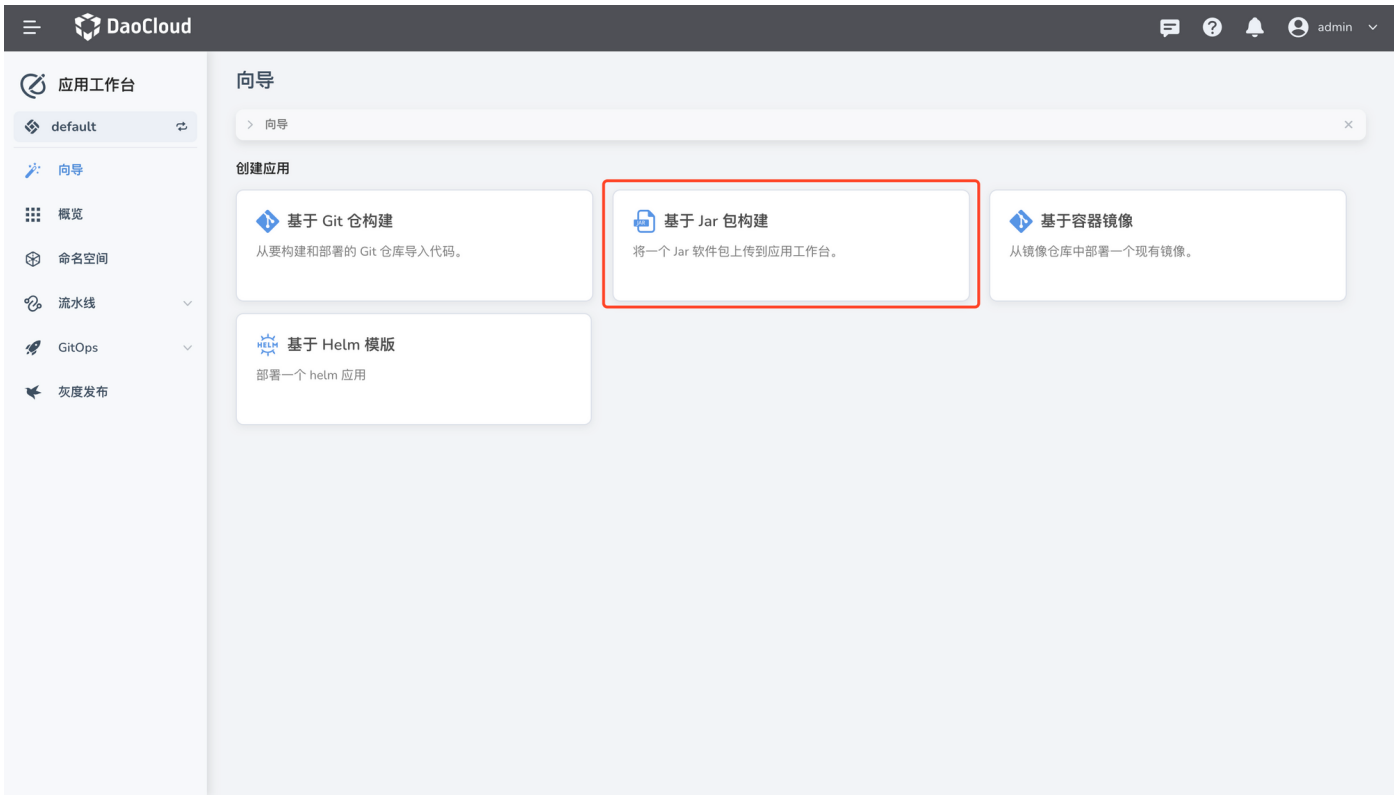
应用工作台支持通过 Git 仓库、Jar 包、容器镜像、Helm 模板等四种方式构建应用。本文介绍如何通过 Jar 文件部署 Java 应用程序。

前提条件

1. 创建一个工作空间和一个用户，该用户需加入该工作空间并具备 `Workspace Editor` 角色。
2. 创建访问镜像仓库的凭证，例如 `registry`。
3. 准备一个镜像仓库，例如 Harbor 仓库。

操作步骤

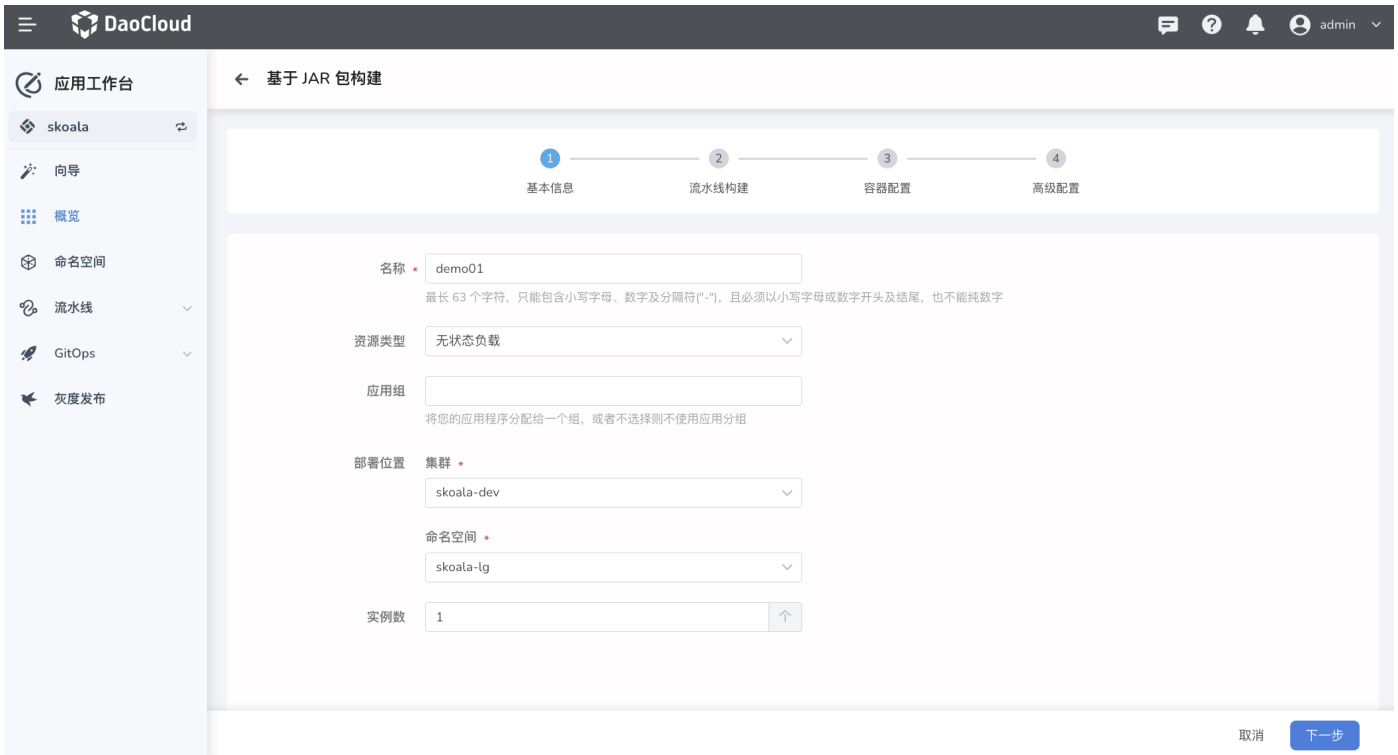
1. 在左侧导航栏点击 `向导`，然后选择 `基于 Jar 包构建`。



Copyright © 2016 - 2023 DaoCloud

2. 参考下列要求填写基本信息，然后点击 `下一步`。

- 名称：最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾。
- 资源类型：选择要创建的应用属于无状态负载，还是有状态负载。
- 应用组：为应用选择所属的分组。为空表示不对此应用进行分组。
- 部署位置：选择将应用部署在哪个集群下的哪个命名空间。只能选择当前工作空间下存在的集群。
- 实例数：为应用设置 Pod 的数量。



Copyright © 2016 - 2023 DaoCloud

3. 参考下列要求配置流水线，然后点击 `下一步` 。

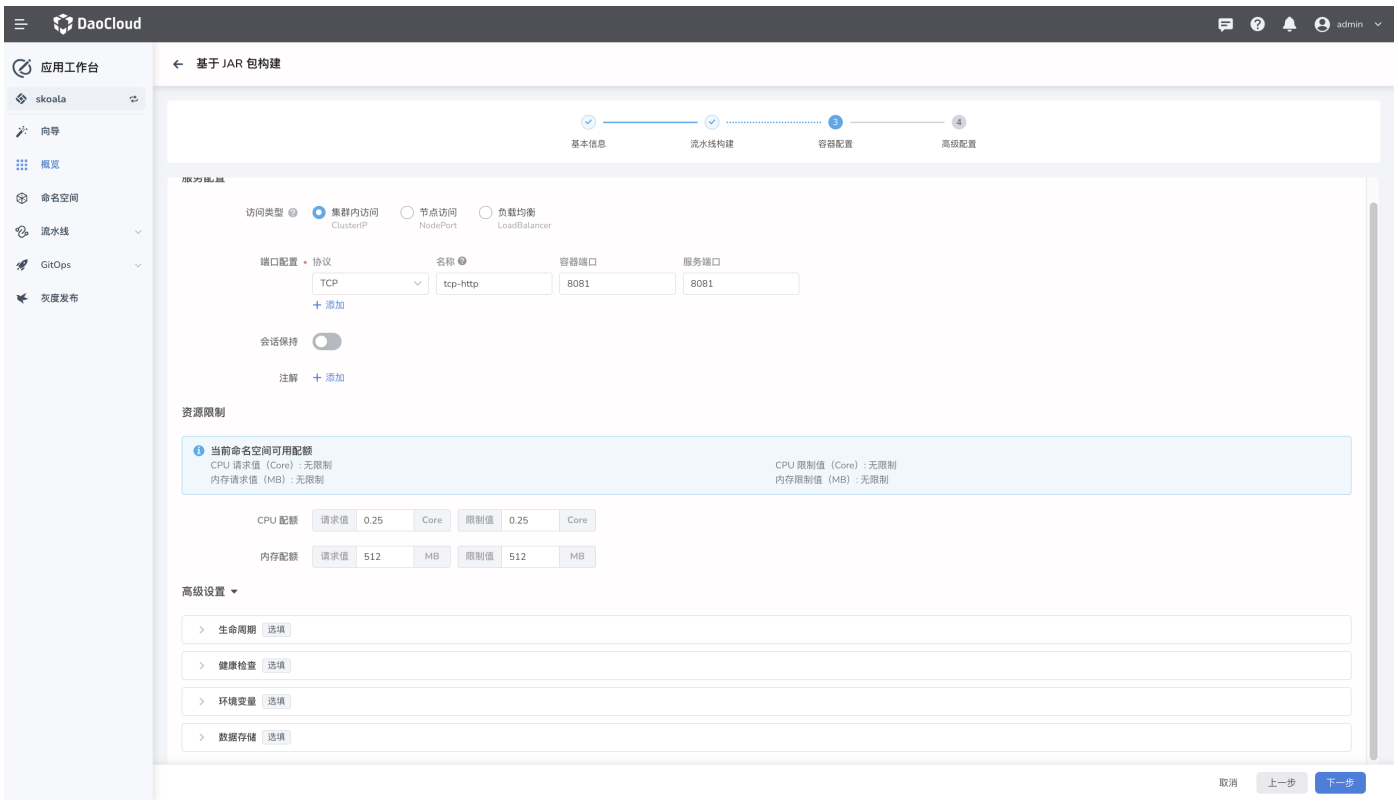
- 目标镜像名称：为目标镜像命名，需包含目标镜像的存储路径，例如 `release-ci.daocloud.io/test-lfj/fromjar` 。
- Tag：为目标镜像打标签，例如版本号 `v1.0` 。
- 凭证：选择访问镜像仓库的凭证，例如 `registry-credential` 。
- JAVA_OPTS：用来设置 JVM 相关运行参数的变量，例如 `-server -Xms2048m -Xmx2048m -Xss512k` 。
- 构建参数：构建参数会以 `--build-arg` 的形式传递到 build 命令中，支持将上游制品下载地址、上游镜像下载地址设置为参数，也支持自定义任意参数。

Copyright © 2016 - 2023 DaoCloud

4. 参考下列要求填写容器配置，然后点击 `下一步`。

- 访问类型：支持通过 clusterIP 仅允许在集群内访问该应用，或者通过 NodePort 允许在集群外部访问，或者通过负载均衡器进行访问。
- 端口配置：根据实际业务场景填写需要暴露的端口号。

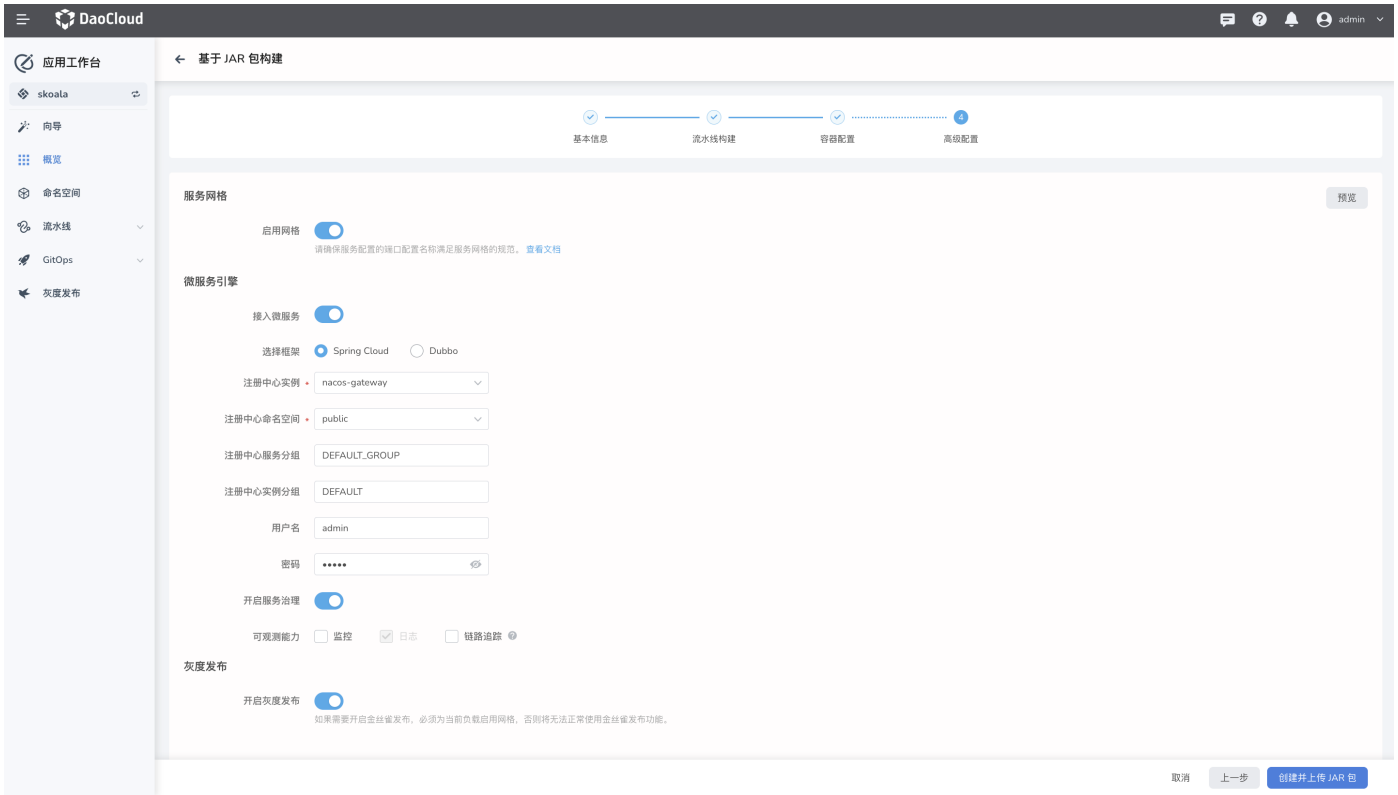
  有关服务配置的更多详细说明，可参考创建服务。
- 资源限制：CPU 和内存配额不得超出应用所在的命名空间在当前工作空间中的剩余资源。
- 生命周期：设置容器启动时、启动后、停止前需要执行的命令。详情可参考容器生命周期配置。
- 健康检查：用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考容器健康检查配置。
- 环境变量：配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考容器环境变量配置。
- 数据存储：配置容器挂载数据卷和数据持久化的设置。



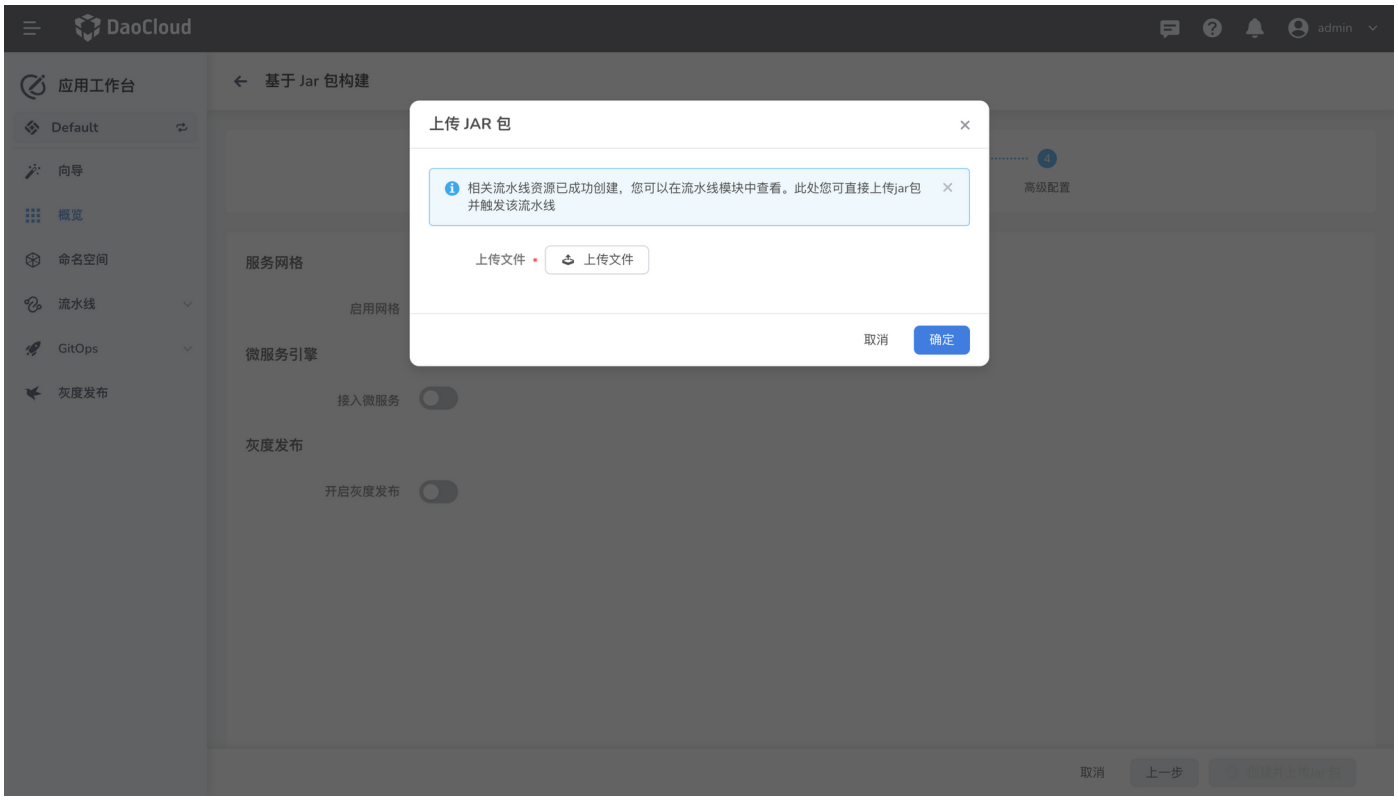
Copyright © 2016 - 2023 DaoCloud

5. 参考下列说明选择是否开启高级功能，然后点击 `创建并上传 Jar 包` 。

- 服务网格：选择是否启用 DCE 5.0 的服务网格模块来治理微服务流量。
- 微服务引擎：是否将新创建的应用接入 DCE 5.0 的微服务引擎模块。 > 有关微服务引擎的配置，可参考基于 Git 仓构建微服务应用。
- 灰度发布：选择是否开启灰度发布。有关灰度发布的更多内容，可参考金丝雀发布。



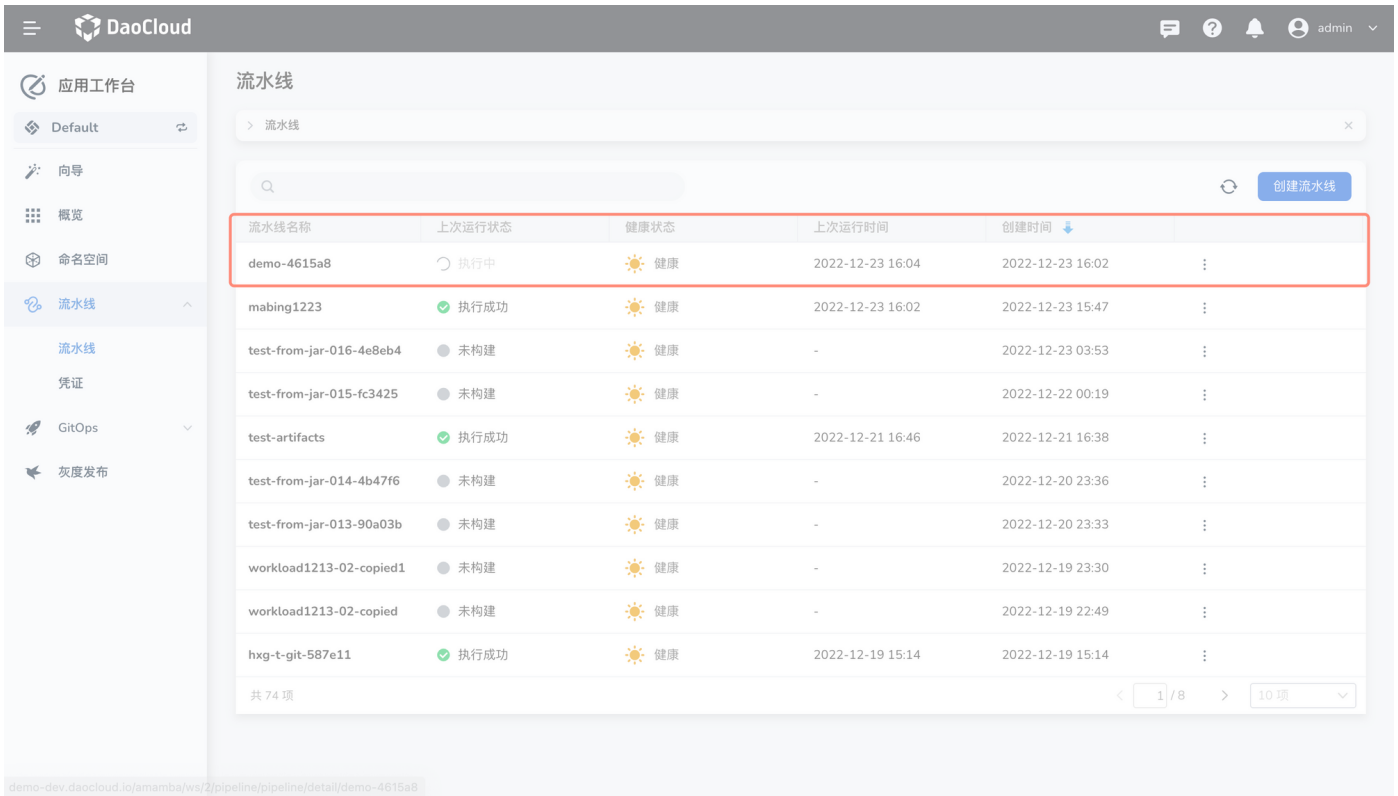
6. 选择需要上传的文件，点击 `确定` 。



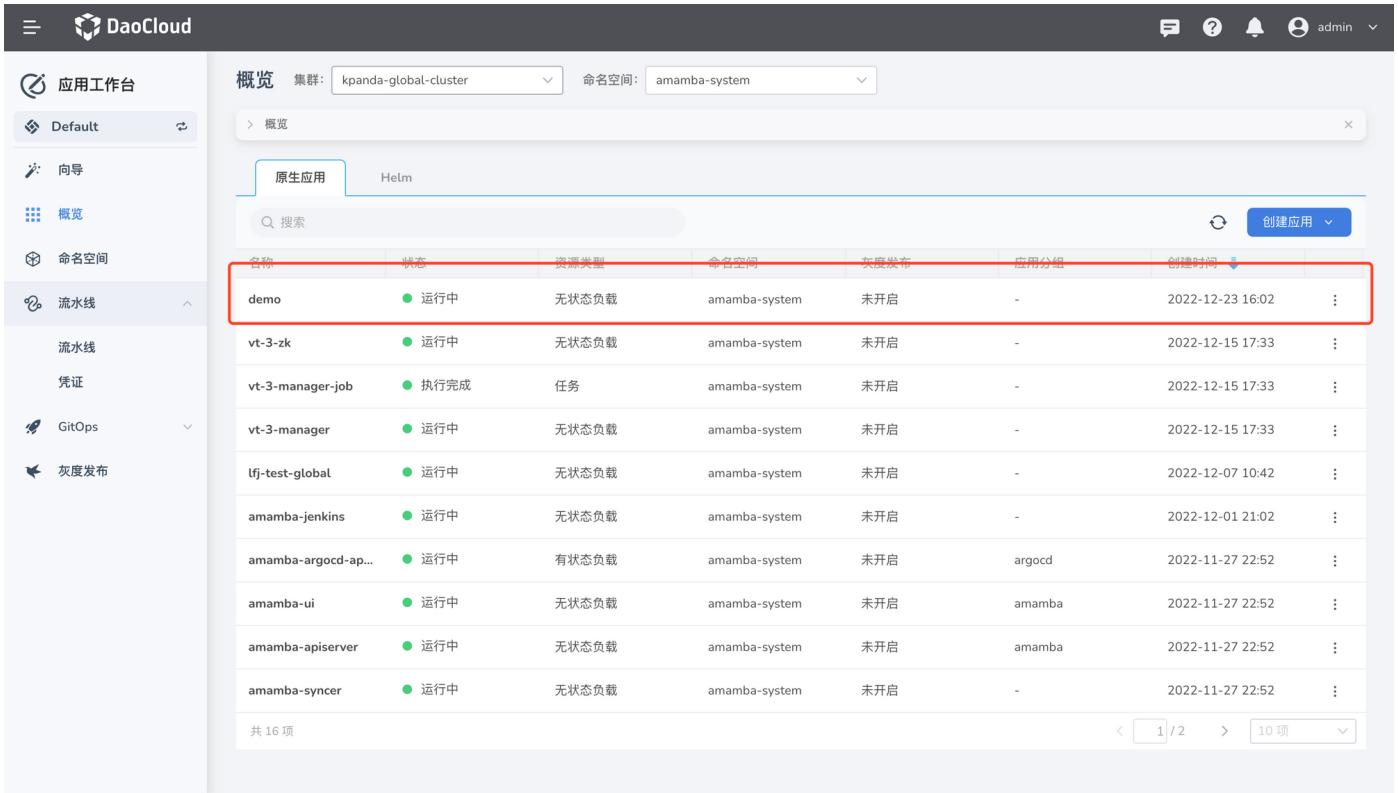
Copyright © 2016 - 2023 DaoCloud

7. 创建成功后会触发运行对应的流水线，在左侧导航栏点击 流水线 可查看其运行状态。

   流水线的命名规则为“对应的应用名称-随机数”，例如通过流水线名称 `demo-4615a8` 即可得知对应的应用名为 `demo`。



8. 等待流水线执行成功后，在左侧导航栏点击 概览，选择 原生应用 页签即可查看新建的应用。



Copyright © 2016 - 2023 DaoCloud