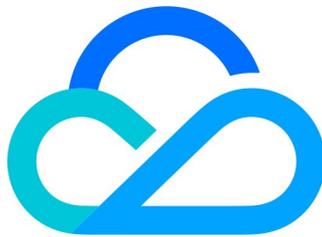


CODING DevOps

实践教学



腾讯云

【 版权声明 】

©2013–2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

实践教程

使用持续集成快速构建应用

自动构建微信小程序

自动化发布 AI 应用

将项目发布至集群

实践教程

使用持续集成快速构建应用

自动构建微信小程序

最近更新时间：2023-09-11 16:05:29

概括

本文将借助于 CODING 的持续集成，手把手带您实现一个微信小程序的持续集成环境，从构建、发布、通知实现自动化，帮您告别繁琐重复性的劳动，解决黯然神伤的烦恼。

整个实现流程大致如下：

1. 创建 CODING DevOps 项目。
2. 创建构建计划，配置微信小程序代码上传白名单。
3. 配置微信小程序代码上传私钥到环境变量中。
4. 配置企业微信的 webhook 地址到环境变量中。
5. 配置构建计划，分为 4 个步骤（检出、编译、上传新版本、发送通知）。

下面我们来一步步实现它。

前置准备

- CODING DevOps 项目。
- 具有管理员权限的微信小程序账号。
- 企业微信机器人 WebHook 地址。
- 将 [示例仓库](#) 导入至 CODING 代码仓库中。

设置小程序白名单

此过程需要将构建任务的网络出口 IP 添加至小程序开发白名单中，您可以在构建计划的基本信息中获取出口 IP。

← tensorflow-demo | **基础信息** | 流程配置 | 触发规则 | 变量与缓存 | 通知提醒 | 前往最新构建 | 操作 | **立即构建**

代码源

- CODING
- GitHub.com
- GitLab.com
- 私有 GitLab
- 码云
- 工蜂
- 通用 Git 仓库
- 不使用

代码仓库

配置来源

- 使用代码库中的
- 使用静态配置的 Jenkinsfile

节点池配置 使用 CODING 提供的云主机进行构建 [团队 CI 构建配额信息](#)

上海 中国	香港 中国	硅谷 美国
公网出口: 111.231.92.100/32,...	公网出口: 124.156.164.25/32,...	公网出口: 170.106.136.17/32,...

- 使用自定义的构建节点进行构建

中国上海

43.143.12.0/24
43.142.234.0/24
43.142.23.0/24
124.220.180.0/24
81.68.101.0/24
111.231.92.0/24
101.33.207.0/24
43.136.72.0/24

中国香港

124.156.164.25/32
119.28.15.65/32

美国硅谷

170.106.136.17/32

170.106.83.77/32

前往微信小程序的管理后台，单击左侧菜单栏中的**开发 > 开发者设置 > 小程序代码上传 > 编辑 IP 白名单**，添加需要的出口地址。

编辑IP白名单 ✕

① 身份确认 — ② 编辑IP白名单

AppID(小程序ID)

IP白名单 +

保存

创建构建计划

在**持续集成**中新建构建计划，选择**自定义构建过程模板**。

构建计划名称 *

mini program

构建过程

1 代码仓库

代码源


 CODING


 GitHub.com


 GitLab.com


 私有 GitLab


 码云


 工蜂


 通用 Git 仓库


 不使用

代码仓库

demo

2 配置来源

使用代码库中的

使用静态配置的 Jenkinsfile

Jenkinsfile 预览

```

pipeline {
  agent any
  stages {
    stage("检出") {
      steps {
        checkout([
          $class: 'GitSCM',
          branches: [[name: GIT_BUILD_REF]],
          userRemoteConfigs: [[
            url: GIT_REPO_URL,
            credentialsId: CREDENTIALS_ID
          ]]
        ])
      }
    }
    stage('自定义构建过程') {
      steps {
        echo "自定义构建过程开始"
        // 请在此处补充您的构建过程
      }
    }
  }
}
                
```

是否前往配置详情

确定 取消

在配置详情中参考 Jenkinsfile 编写构建过程。

Jenkinsfile:

```

pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
        checkout([
          $class: 'GitSCM',
          branches: [[name: env.GIT_BUILD_REF]],
          userRemoteConfigs: [[
            url: env.GIT_REPO_URL,
            credentialsId: env.CREDENTIALS_ID
          ]]
        ])
      }
    }
    stage('构建') {
      steps {
        echo '开始安装依赖'
      }
    }
  }
}
                
```

```
    sh 'npm install'
    echo '开始构建...'
    sh 'npm run build'
    echo '构建完成'
  }
}
stage('上传新版本') {
  steps {
    withCredentials([sshUserPrivateKey(credentialsId:
"${env.privatekey}",keyFileVariable: 'identity')]) {
      sh 'node upload.js -p ${identity}'
    }
  }
}
stage('发送新版本通知') {
  steps {
    sh 'node notification.js -u ${WECHAT_WEBHOOK}'
  }
}
}
```

添加环境变量

持续集成过程中，我们总会将一些配置（例如：账号密码/版本号等）信息以环境变量的形式注入到构建过程中。在本实践中需要将以下两个凭据以环境变量的形式添加至构建计划中。

- 微信小程序代码上传私钥。
- 企业微信机器人 webhook 地址。

微信小程序代码上传私钥

前往微信管理后台：[开发 > 开发设置 > 小程序代码上传](#)获取上传私钥与 AppID。



将信息导入至 CODING 项目中的项目设置 > 开发者选项 > 凭据管理 > 录入凭据 > 选择 SSH 私钥凭据类型，复制私钥内容粘贴至凭据中。CODING 会对您的私钥进行加密保存，杜绝明文暴露在工程文件中。同时还需要勾选授权所有持续集成构建计划。

> 演示项目 ▾

- ← 项目设置
- 👤 项目与成员
- ☑ 项目协同
- 📢 项目公告
- </> 开发者选项

项目设置 / 凭据管理 / 录入凭据

录入凭据

凭据类型

SSH 私钥
▾

凭据名称*

小程序

SSH 私钥*

```
-----BEGIN RSA PRIVATE KEY-----

-----END RSA PRIVATE KEY-----
```

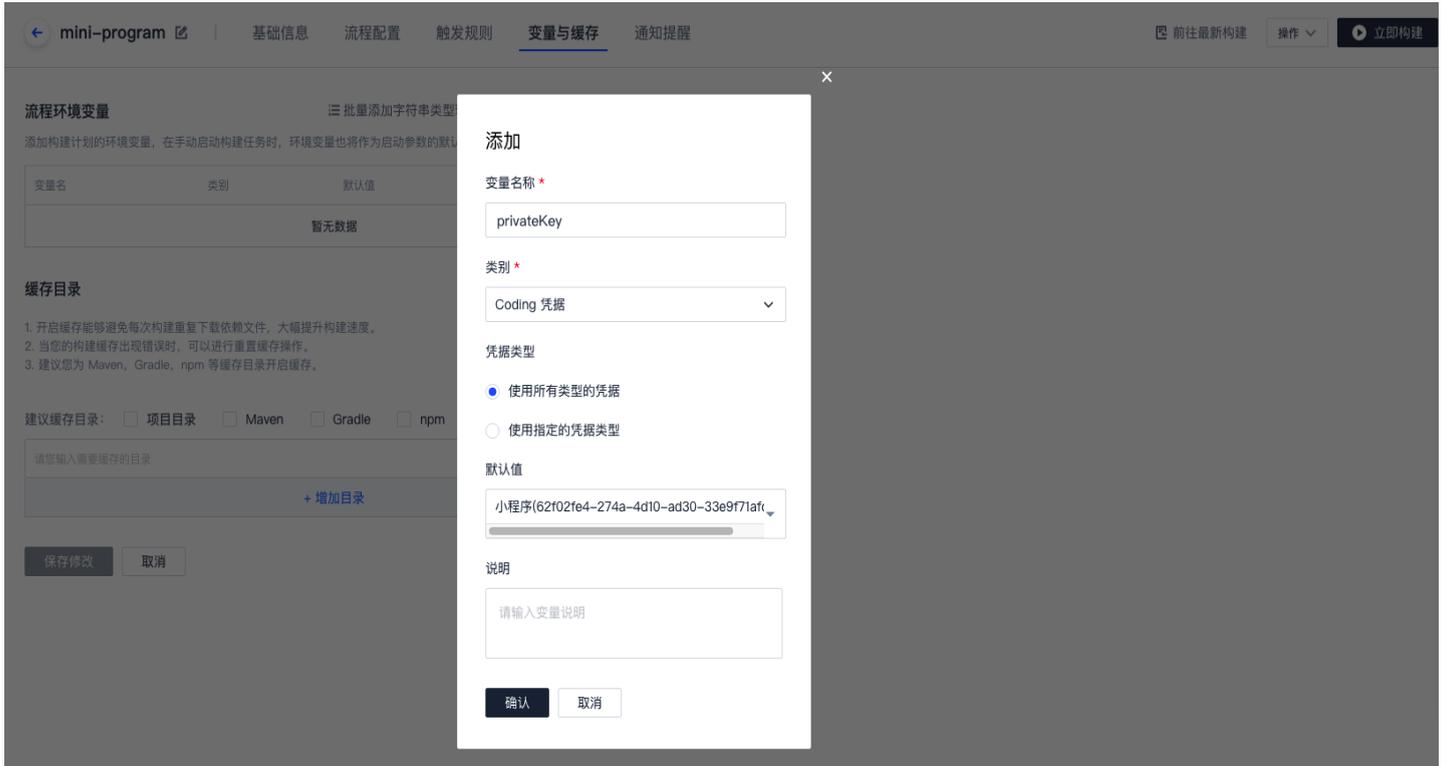
私钥口令

私钥没有口令时为空

凭据描述

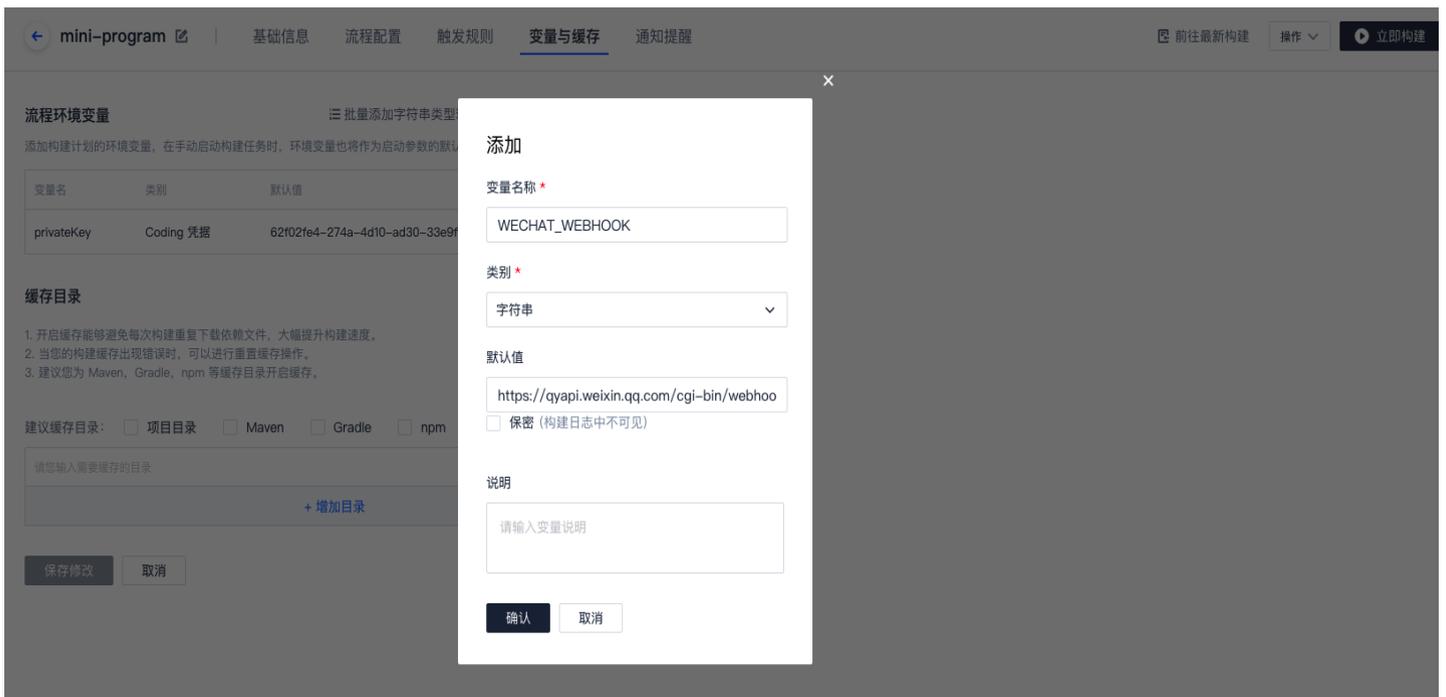
请输入凭证描述，不超过 100 个字符

创建完成后将生成一串凭据 ID，将其导入至变量与缓存中。



上传机器人 webhook

新建 [群聊机器人](#) 后，复制机器人的 webhook 地址后，以字符串的形式粘贴至**变量与缓存**中。



构建阶段细节

示例项目的代码是从微信开发者工具中抽离的关于小程序/小游戏项目代码的编译模块。开发者可不打开小程序开发者工具，独立使用已导入的示例仓库进行小程序代码的上传、预览等操作。

在上文中，我们将小程序上传代码的凭证加到环境变量，通过在 Jenkinsfile 定义 `withCredentials` 参数即可快速提取凭证。

提取到凭证后，调用了一个 `upload.js` 脚本。此部分代码涉及到了代码的上传和预览二维码的生成。

```
const ci = require('miniprogram-ci')
const path = require('path');
const fs = require("fs");
const argv = require('minimist')(process.argv.slice(2));
const package = require('./package.json')
const appDirectory = fs.realpathSync(process.cwd());
const ProjectConfig = require('./dist/project.config.json');
const previewPath = path.resolve(appDirectory, './preview.jpg');

(async () => {
  try {
    const project = new ci.Project({
      appid: ProjectConfig.appid,
      type: "miniProgram",
      projectPath: path.resolve(appDirectory, './dist'),
      privateKeyPath: argv.p,
      ignores: ["node_modules/**/*"],
    })
    await ci.upload({
      project,
      version: package.version,
      desc: package.versionDesc,
      setting: {
        ...ProjectConfig.setting
      },
      onProgressUpdate: console.log,
    })
    await ci.preview({
      project,
      version: package.version,
      desc: package.versionDesc,
      qrcodeFormat: "image",
      qrcodeOutputDest: previewPath,
      setting: {
```

```
        ...ProjectConfig.setting
    },
    onProgressUpdate: console.log,
  })
} catch (e) {
  console.error(e);
  process.exit(1);
}

})()
```

通知阶段

原理为直接发送请求，触发 webhook 后将发送预览二维码。关于企业微信 API 请参见 [群机器人配置说明](#)。

```
const md5File = require('md5-file')
const axios = require('axios');
const path = require('path');
const argv = require('minimist')(process.argv.slice(2));
const fs = require("fs");
const appDirectory = fs.realpathSync(process.cwd());

const previewPath = path.resolve(appDirectory, './preview.jpg');

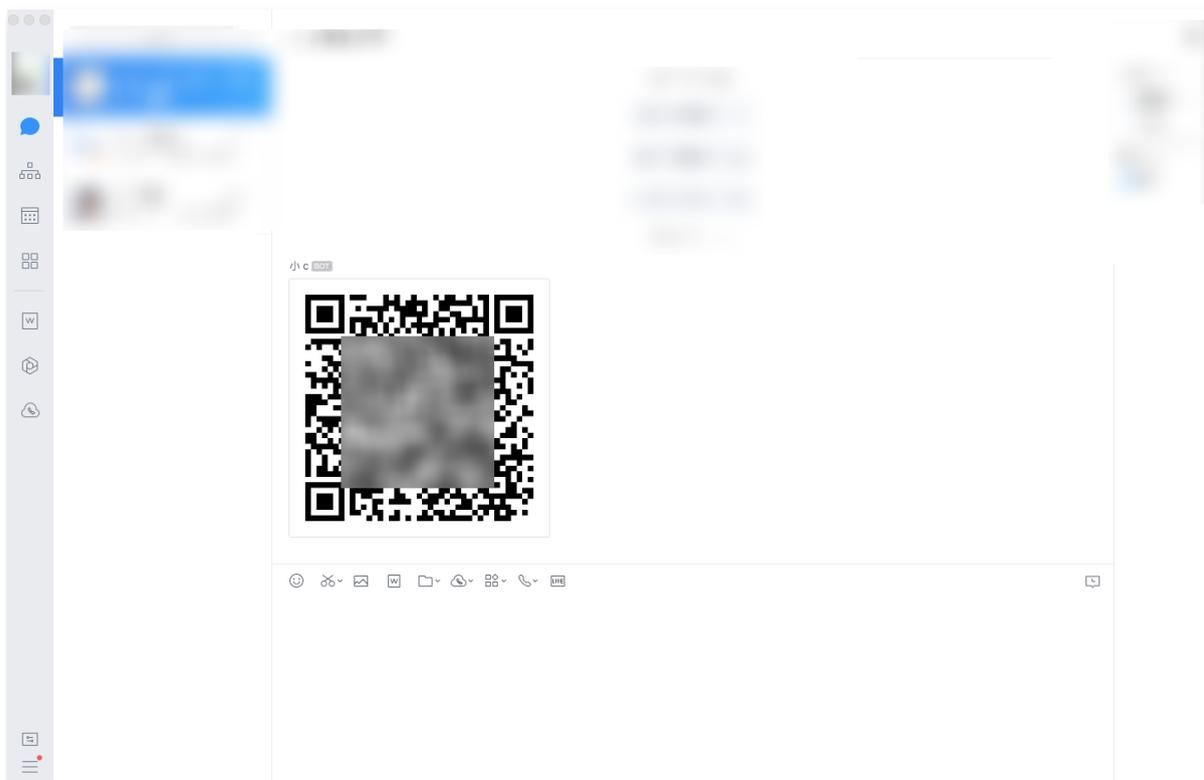
function sendQrCode (imageBase64, hash) {
  return axios({
    headers: { "Content-Type": 'application/json' },
    method: 'post',
    url: argv.u,
    data: {
      "msgtype": "image",
      "image": {
        "base64": imageBase64,
        "md5": hash
      }
    }
  });
}

(async () => {
  try {
    const imageData = fs.readFileSync(previewPath);
```

```
const hash = md5File.sync(previewPath)
const imageBase64 = imageData.toString("base64");
await sendQrCode(imageBase64, hash);

} catch(e) {
  console.error(e);
  process.exit(1);
}
})()
```

当我们把代码上传，发布新版本之后，就会往企业微信群上发送一个预览二维码，通知群上的同事进行预览体验



更多扩展

版本号和本说明没有集中管理，目前是读取 `package.json` 文件里的 `version` 和 `versionDesc` 参数。若需要进行版本控制，可以尝试通过 CODING 代码仓库的 `tag` 来管理版本，同时配置通过 `tag` 来触发构建。

自动化发布 AI 应用

最近更新时间：2023-09-11 16:05:29

本文将介绍如何基于 Tensorflow.js 开发一款 logo 识别应用，并且在此过程中演示如何通过 CODING DevOps 实现自动化构建。

前置准备

- [git](#)
- [nodejs](#)
- [yarn](#)
- [docker](#)（仅本地构建与运行此应用需要）
- [CODING 项目](#)
- [Docker 制品仓库](#)，权限需设置为公开，制品仓库命名为 `build`

初始化

在创建代码仓库时选择导入外部仓库，粘贴示例仓库地址。

← 创建代码仓库 | 普通创建 | 模板创建 | 导入外部仓库

Git 仓库 URL *

仓库名称 *

 17/100

是否开源

私有仓库（仅对仓库成员可见，仓库成员可访问仓库）

公开仓库（公开后，任何人都可以访问代码仓库，请谨慎考虑！）

代码扫描

开启代码扫描可以发现代码中的安全漏洞、功能缺陷等代码问题，结果将展示在合并请求详情中，辅助您进行代码评审。[查看详情](#)

自动匹配语言 通过模板创建 复用已有方案

初次扫描到有效代码，根据语言属性自动匹配推荐规则集。

导入成功后，将代码拉取至本地中。

```
/Volumes/CODING-Help/demo-tensorflowjs master ls
Dockerfile      build.sh        package.json    yarn.lock
Dockerfile.compile data            sample
Jenkinsfile     docker         src
README.md       package-lock.json tech.md
/Volumes/CODING-Help/demo-tensorflowjs master | 10130 19:41:32
```

接下来可以运行 `yarn install` 命令并在本地安装依赖。

```
/Volumes/CODING-Help/demo-tensorflowjs master ? yarn install
yarn install v1.22.4
warning package-lock.json found. Your project contains lock files generated by t
ools other than Yarn. It is advised not to mix package managers in order to avoi
d resolution inconsistencies caused by unsynchronized lock files. To clear this
warning, remove package-lock.json.
[1/4] 🔍 Resolving packages...
[2/4] 📦 Fetching packages...
info There appears to be trouble with your network connection. Retrying...
info There appears to be trouble with your network connection. Retrying...
info There appears to be trouble with your network connection. Retrying...
info There appears to be trouble with your network connection. Retrying...
[3/4] 🔗 Linking dependencies...
warning "@tensorflow/tfjs > @tensorflow/tfjs-data@1.7.4" has unmet peer dependen
cy "seedrandom@~2.4.3".
🔨 Building fresh packages...
* Done in 81.09s.
```

安装完成后运行 `./build.sh --local` 命令进行本地构建。应用开发完成后还需要进行容器化，以方便应用传播与测试。每次开发后都会生成一个新的制品，若要手动重复打包再上传至制品仓库，此过程未免过于繁琐。借助持续集成工具，能够在每次开发后自动触发构建并上传至制品仓库，解放生产力。并且在构建的过程中还能够配置通知机制，及时获得构建反馈。

创建构建计划

进入任一项目后，单击左侧菜单栏的**持续集成**，新建构建计划时选择**自定义过程模板**。

← 自定义构建过程

构建计划名称 *

构建过程

1 代码仓库

代码源

 CODING	 GitHub.com	 GitLab.com	 私有 GitLab
 码云	 工蜂	 通用 Git 仓库	 不使用

代码仓库

2 配置来源

- 使用代码库中的 ?
- 使用静态配置的 Jenkinsfile ?

是否前往配置详情

Jenkinsfile 预览

```

pipeline {
  agent any
  stages {
    stage("检出") {
      steps {
        checkout([
          $class: 'GitSCM',
          branches: [[name: GIT_BUILD_REF]],
          userRemoteConfigs: [[
            url: GIT_REPO_URL,
            credentialsId: CREDENTIALS_ID
          ]]
        ])
      }
    }
    stage('自定义构建过程') {
      steps {
        echo "自定义构建过程开始"
        // 请在此处补充您的构建过程
      }
    }
  }
}
    
```

跳转至配置详情后参考以下 Jenkinsfile 修改构建过程。

Jenkinsfile:

```

pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
    
```

```
checkout([$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[
    url: env.GIT_REPO_URL,
    credentialsId: env.CREDENTIALS_ID
]]])
}
}
stage('构建') {
    steps {
        echo '显示环境变量'
        sh 'printenv'
        echo '构建中...'
        sh 'docker version'
        sh './build.sh'
        echo '构建完成.'
    }
}
stage('推送到 CODING Docker 制品库') {
    steps {
        script {
            docker.withRegistry(
                "${CCI_CURRENT_WEB_PROTOCOL}://${env.CODING_DOCKER_REG_HOST}",
                "${env.CODING_ARTIFACTS_CREDENTIALS_ID}"
            ) {
                docker.image("${env.CODING_DOCKER_IMAGE_NAME}:${env.GIT_COMMIT}").push()
            }
        }
    }
}
environment {
    CODING_DOCKER_REG_HOST = "${env.CCI_CURRENT_TEAM}-
docker.pkg.${env.CCI_CURRENT_DOMAIN}"
    CODING_DOCKER_IMAGE_NAME =
"${env.PROJECT_NAME.toLowerCase()}/${env.DOCKER_REPO_NAME}/${env.DOCKER_
IMAGE_NAME}"
}
}
```

此流水线脚本大致分为三个阶段，检出阶段为拉取代码，构建阶段为运行构建脚本，推送阶段为把 docker 制品推送到制品库。构建阶段主要运行脚本文件 `build.sh` 主要内容如下：

```
#!/bin/bash

docker build -t compiler -f Dockerfile.compile .

if [ "$1" = "--local" ]
then
    docker build -t logo-reg .
else
    docker build -f $DOCKERFILE_PATH -t
$CODING_DOCKER_IMAGE_NAME:$GIT_COMMIT $DOCKER_BUILD_CONTEXT
fi
```

此脚本的设计思路为分阶段构建方案。以 `Dockerfile.compile` 作为构建基础镜像的构建文件、`Dockerfile` 作为实际运行镜像的构建文件。使用这种方法可以使得实际运行的镜像不包含构建应用所需要的环境，大大减少镜像体积，构建后的镜像仅为 149 Mb。此外，此脚本还可以通过 `--push` 参数来灵活的区分云上构建环境与本地构建环境。

配置触发规则

持续集成支持多种触发方式，例如代码源触发、定时触发、API 触发及手动触发。其中代码源触发又可配置为推送到指定分支或标签触发，触发方式多样，可满足绝大部分场景需要。

如前言中所说，我们希望把更多的精力放在代码开发上，尽量减少构建所带来的干扰。因此可以设置触发规则，例如通过配置如下正则表达式，当分支名满足规则后即可自动触发构建。

```
^refs/(heads/(release|release-.*|build-.*|feat-.*|fix-.*|test-.*|mr/.*))
```

← tensorflow-demo
基础信息 | 流程配置 | 触发规则 | 变量与缓存 | 通知提醒

CODING 持续集成支持通过多种方式来触发构建计划, [查看完整帮助文档](#)

代码源触发 代码更新时自动执行

选择需要触发持续集成的事件

- 推送到 master 时触发构建
- 推送新标签时触发构建
- 推送到分支时触发构建
- 符合分支或标签规则时构建 ?

`^refs/(heads/(release|release-`

合并请求

合并请求触发会构建源分支与目标分支合并后的结果, 能够尽可能早地发现集成中的错误, [查看完整帮助文档](#)

- 创建合并请求时触发构建
- 合并合并请求时触发构建
- 源分支变更时触发构建
- 目标分支变更时触发构建
- 自动取消相同合并请求 ?

定时触发

分支	执行时间	操作
暂无内容 +添加		

API 触发

触发地址
https://straybirds.coding.net/ap...
📄
生成 curl 命令触发示例

需使用具有持续集成 API 触发权限的 [项目令牌](#) 触发

手动触发 指定 [立即构建](#) 的默认构建目标 master

设置变量与缓存

持续集成过程中, 我们总会将一些配置 (例如: 账号密码/版本号等) 信息以环境变量的形式注入到构建过程中。

← tensorflow-demo | 基础信息 | 流程配置 | 触发规则 | **变量与缓存** | 通知提醒

流程环境变量 批量添加字符串类型环境变量 | [+ 添加环境变量](#)

添加构建计划的环境变量，在手动启动构建任务时，环境变量也将作为启动参数的默认值，[查看完整帮助文档](#)

变量名	类别	默认值	操作
DOCKER_IMAGE_NAME	字符串	logo-reg	
DOCKER_BUILD_CONTEXT	字符串	.	
DOCKERFILE_PATH	字符串	Dockerfile	
DOCKER_IMAGE_VERSION	字符串	\${GIT_LOCAL_BRANCH:-branch}-\${GIT_...	
DOCKER_REPO_NAME	字符串	build	

所涉及的环境变量如下：

变量名	默认值
DOCKER_IMAGE_NAME	logo-reg
DOCKER_BUILD_CONTEXT	.
DOCKERFILE_PATH	Dockerfile
DOCKER_IMAGE_VERSION	\${GIT_LOCAL_BRANCH:-branch}-\${GIT_COMMIT}
DOCKER_REPO_NAME	build

执行构建

触发持续集成后，您可以在**构建过程**中看到各步骤的运行情况。

← 构建记录#1
构建过程
构建快照
改动记录
测试报告
通用报告
构建产物

构建中

主账号 手动触发
触发于 2 分钟前, 持续时长 1 分钟 59 秒

demo-tensorflowjs master → 340275d
feat: add tech tutorial

构建过程

执行 Shell 脚本 1 分钟 44 秒

```

8 [2021-11-26 11:09:07] 932265d85c04: Pulling fs layer
9 [2021-11-26 11:09:08] 932265d85c04: Download complete
10 [2021-11-26 11:09:08] 09c7373d031e: Verifying Checksum
11 [2021-11-26 11:09:08] 09c7373d031e: Download complete
12 [2021-11-26 11:09:08] df20fa9351a1: Verifying Checksum
13 [2021-11-26 11:09:08] df20fa9351a1: Download complete
14 [2021-11-26 11:09:08] bb3593db620e: Verifying Checksum
15 [2021-11-26 11:09:08] bb3593db620e: Download complete
16 [2021-11-26 11:09:08] df20fa9351a1: Pull complete
17 [2021-11-26 11:09:10] bb3593db620e: Pull complete
18 [2021-11-26 11:09:10] 09c7373d031e: Pull complete
19 [2021-11-26 11:09:10] 932265d85c04: Pull complete
20 [2021-11-26 11:09:10] Digest:
   sha256:a08531aa9537754563abbdbcc1da008e2527a72a128c448ecd8f9b6a4ea4be67
21 [2021-11-26 11:09:10] Status: Downloaded newer image for node:12.18.1-alpine3.12
22 [2021-11-26 11:09:10] ----> 93664755c9e2
23 [2021-11-26 11:09:10] Step 2/4 : COPY . /compile_source
24 [2021-11-26 11:09:11] ----> f02397bf1gef
25 [2021-11-26 11:09:11] Step 3/4 : WORKDIR /compile_source
26 [2021-11-26 11:09:11] ----> Running in 86c5d2ca7494
27 [2021-11-26 11:09:11] Removing intermediate container 86c5d2ca7494
28 [2021-11-26 11:09:11] ----> da6095d1ea47
29 [2021-11-26 11:09:11] Step 4/4 : RUN yarn && yarn build
30 [2021-11-26 11:09:11] ----> Running in 501b3e396548
31 [2021-11-26 11:09:12] yarn install v1.22.4
32 [2021-11-26 11:09:12] [1/4] Resolving packages...
33 [2021-11-26 11:09:12] [2/4] Fetching packages...
34 [2021-11-26 11:09:39] info fsevents@1.2.13: The platform "linux" is incompatible with
   this module.
35 [2021-11-26 11:09:39] info "fsevents@1.2.13" is an optional dependency and failed
   compatibility check. Excluding it from installation.
36 [2021-11-26 11:09:39] [3/4] Linking dependencies...
37 [2021-11-26 11:09:39] warning "@tensorflow/tfjs > @tensorflow/tfjs-data@1.7.4" has
   unmet peer dependency "seedrandom@~2.4.3".
38 [2021-11-26 11:09:44] [4/4] Building fresh packages...
39 [2021-11-26 11:09:51] Done in 37.79s.
40 [2021-11-26 11:09:51] yarn run v1.22.4
41 [2021-11-26 11:09:51] $ node_modules/.bin/parcel build src/index.html
42 [2021-11-26 11:09:51] /bin/sh: lscpu: not found
43 [2021-11-26 11:09:51] Browserslist: caniuse-lite is outdated. Please run:
44 [2021-11-26 11:09:51] npx browserslist@latest --update-db
                
```

下载制品

构建完成后，可以看到在 build 制品仓库中已有新的制品，可以根据操作指引拉取至本地中。

操作指引

拉取

输入以下拉取相关信息，生成拉取命令：

制品名称：

制品版本：

请在命令行执行以下命令进行拉取：

```
docker pull StrayBirds-docker.pkg.coding.net/demo/build/logo-reg:340275df1ec
```

推送信息

推送人 项目助手

推送时间 2021-11-26 11:11:42

其他

大小 61.22 MB

hash sha256
bf6492
6613cc
d830f

系统架构 linux/amd64

运行应用

使用以下命令，运行已拉取的制品，即可开始通过机器学习以辨别 CODING、GitHub、GitLab Logo。

将命令中的仓库地址替换为自己制品仓库的地址。

```
docker run -p 8080:80 StrayBirds-docker.pkg.coding.net/demo/build/logo-reg:340275df1ecf1b2e7800a237ebceb10cee7161c
```

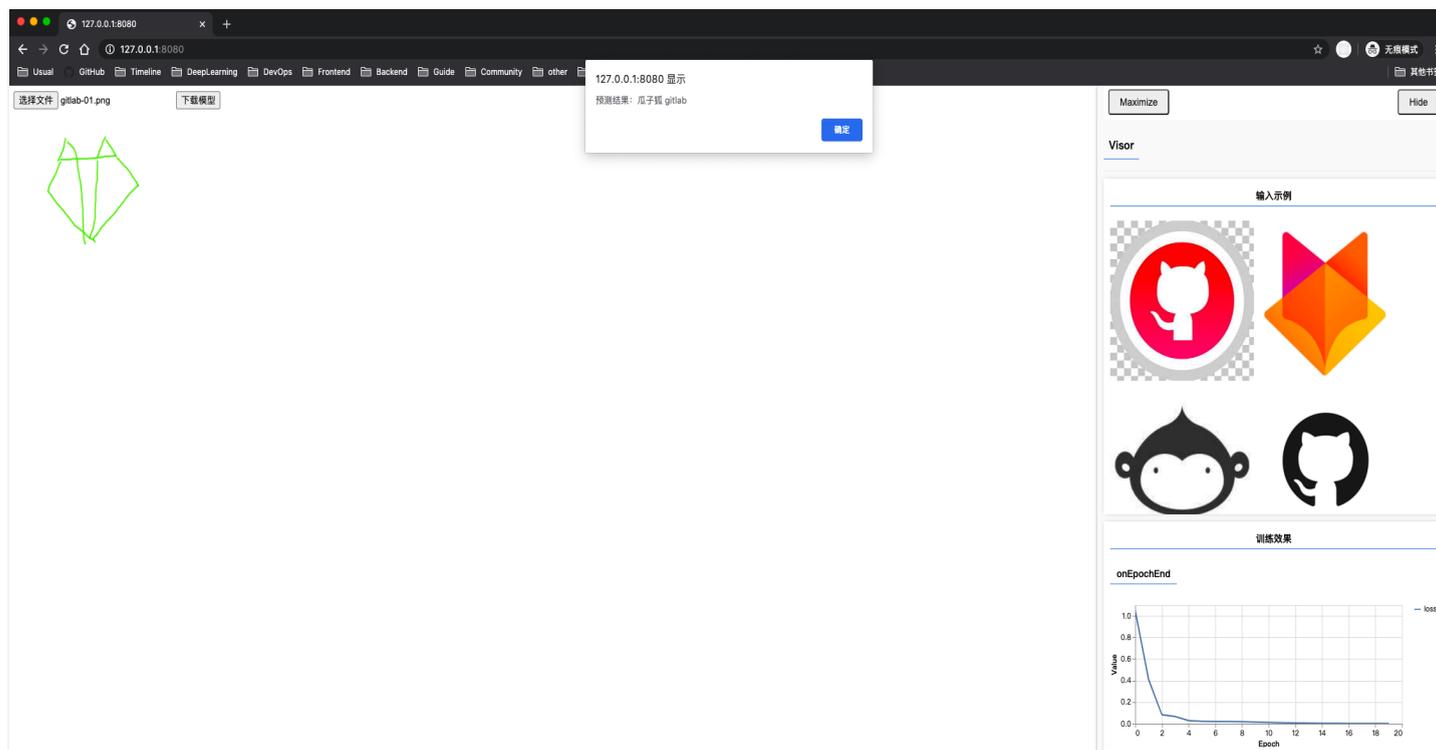
```
/Volumes/CODING-Help/demo-tensorflowjs ➤ master docker run -p 8080:80 StrayBirds-docker.pkg.coding.net/demo/build/logo-reg:340275df1ecf1b2e7800a237ebce
b10ceee7161c
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to per
form configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-def
ault.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf
.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/co
nf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates
.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes
.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/11/25 09:43:41 [notice] 1#1: using the "epoll" event method
2021/11/25 09:43:41 [notice] 1#1: nginx/1.21.4
2021/11/25 09:43:41 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-
6)
2021/11/25 09:43:41 [notice] 1#1: OS: Linux 5.10.25-linuxkit
2021/11/25 09:43:41 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2021/11/25 09:43:41 [notice] 1#1: start worker processes
2021/11/25 09:43:41 [notice] 1#1: start worker process 31
```

浏览器打开 <http://127.0.0.1:8080>，等待数分钟后，右下角训练损失几乎将为 0 即为训练完毕（若不为 0，说明训练过程收到不可逆干扰，请刷新页面即可重新训练）。上传任一 CODING、GitHub、GibLab 的图标文件，此应用可准确的预测出图片属于哪个 logo。

预测 CODING logo:

The screenshot shows a web browser window with the address bar set to 127.0.0.1:8080. A file named 'coding-01.png' is selected in the browser's file picker. A modal dialog box displays the prediction result: '预测结果: 洋葱猴 coding'. The application interface includes a 'Visor' section with '输入示例' (Input Examples) showing GitHub and GitLab logos, and '训练效果' (Training Results) showing a line graph of 'onEpochEnd' loss over 20 epochs, which drops to near zero.

预测 GitLab logo:



可以看出，此应用具有相当高的准确率！

总结

本文通过一个基于 Tensorflow.js 开发的 AI 应用项目讲解了如何使用持续集成与制品仓库。借用 CODING DevOps 平台的这些功能，我们解放本地算力，省去了人为的不必要劳动，提高了生产力。

除此之外，持续集成可以构建任何应用（无论是终端、后端、甚至机器学习应用）。部署与构建不再是编程中的烦恼，专注于代码，专注于业务，繁琐之事皆可放行交由平台。

将项目发布至集群

最近更新时间：2023-09-11 16:05:29

本文将通过示例项目，演示如何从代码仓库开始，最终将项目自动化发布至集群中。

操作步骤

步骤1：导入示例代码仓库

先在团队创建全功能 DevOps 项目。

STEP 1/2
选择项目模板

全功能 DevOps 项目

开启一个全功能的 CODING 项目，包含代码托管、项目协同、持续集成、制品管理、持续部署、测试管理、知识管理在内的全部能力，促进研发团队的 DevOps 成熟度提升。

[选择](#) [了解更多](#)

按需选择

按团队当前实际所需开启所需功能。

[代码托管](#) [需求搜集 & 事项跟踪 & 迭代管理](#)

[构建流水线 & 自动化测试](#) [制品管理](#) [自动化部署](#)

[测试用例管理](#) [知识管理](#)

[选择](#) [了解更多](#)

您还可以 [从我们的教学项目开始](#)。

在代码仓库中导入开源 [示例代码仓库](#)。

← 导入外部仓库 / URL 导入

Git 仓库 URL *

https://e.coding.net/codingtest-cd/k8sdemo/k8sDemo.git

仓库名称 *

k8sDemo

7/100

是否开源

- 私有仓库（仅对仓库成员可见，仓库成员可访问仓库）
- 公开仓库（公开后，任何人都可以访问代码仓库，请谨慎考虑！）

代码扫描



开启代码扫描可以发现代码中的安全漏洞、功能缺陷等代码问题，结果将展示在合并请求详情中，辅助您进行代码评审。[查看详情](#)

- 自动匹配语言 通过模板创建 复用已有方案

初次扫描到有效代码，根据语言属性自动匹配推荐规则集。

完成创建

取消

步骤2：创建持续集成任务

1. 选择项目内左侧产品栏**持续集成**，单击右上角**创建构建计划**，选择部署分类下的“推送到 Kubernetes”模板。

← 选择构建计划模版

自定义构建过程

构建计划是持续集成的基本单元，在这里你可以快速创建一个构建计划，更多内容可以到构建计划详情中进行配置。[查看帮助文档](#)

全部 团队模版 编程语言 Serverless 镜像仓库 制品库 **部署** 基础 API 文档

 **CODING Docker 镜像推送并部署到 Kubernetes**
将一个构建完毕的 Docker 镜像推送到当前项目下的 Docker 制品库中并...

若没有找到合适的模版，可选择自定义构建过程

 **自定义构建过程**
允许您根据 Jenkinsfile 的规范来随意定制持续集成流水线过程。

2. 示例仓库中已含有 `Dockerfile` 文件，因此仅需自定义镜像名称。若制品仓库中没有 Docker 类型仓库，可以在第三步单击页面上的**创建新的制品库**。

2 构建 Docker 镜像

Docker 镜像名称 *

Dockerfile 文件位置 *

Docker 构建目录 *

Docker 镜像版本 *

```

stage('构建镜像并推送到 CODING Docker 制品库') {
  steps {
    sh "docker build -t ${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION} -f ${DOCKERFILE_PATH} $
    useCustomStepPlugin(
      key: 'coding-public:artifact_docker_push',
      version: 'latest',
      params: [
        image:"${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION}",
        repo:"${DOCKER_REPO_NAME}"
      ]
    )
  }
}
            
```

3 推送到 CODING Docker 制品库

Docker 制品库 *

3. 创建完成后，前往持续集成设置的触发规则，勾选代码源触发。当 master 分支有代码变更时将自动触发持续集成任务，你也可以配合团队工作流，自定义多种自动触发方式。

代码源触发 代码更新时自动执行 ?

代码源中 同时满足 以下规则的代码提交将会触发流水线。如果您没有设置任何规则，则代码源中所有代码提交都会触发流水线。

包含 代码分支或标签 [*] ^refs/heads/master 🗑

[+添加](#)

合并请求触发

- 创建合并请求时触发构建
- 合并合并请求时触发构建
- 源分支变更时触发构建
- 自动取消相同合并请求 ?
- 目标分支变更时触发构建

在代码源中 同时满足 以下规则的合并请求事件将会触发流水线。如果您没有设置任何规则，则所有合并请求都会触发流水线。

[+添加](#)

步骤3：配置目标制品仓库

软件制品是指由源码编译打包生成的二进制文件，不同的开发语言对应着不同格式的二进制文件，这些文件通常可以直接运行在服务器上，用以支撑应用运行。持续集成任务运行成功后，前往制品仓库中就可以查看已推送的制品，您可以参见 [Docker](#) 在本地进行镜像推拉。

制品仓库 | 全部制品 | 仓库管理 创建制品仓库

k8s
Docker 仓库 | 项目内

example
npm 仓库 | 项目内

ruby
Docker 仓库 | 项目内

apk
Generic 仓库 | 项目内

build
Docker 仓库 | 公开

k8s 设置仓库 代理设置 版本覆盖策略

类型 Docker | 权限 项目内

镜像列表

发布状态 全部 + 制品属性 搜索制品名称... 操作指引

镜像名	最新推送版本	最近更新时间	版本数	操作
k8sdemo	master-27eabe61d2a26b814cb9e0ece98f8b6...	2022-03-16 17:36:42	1	...

1-1 个, 共 1 个 每页显示行数 15 1

获取制品仓库的拉取链接。

更新时间
版本号
仓库
概览
镜像历史

操作指引
配置凭据
拉取
镜像源加速

帮助中心

拉取

输入以下拉取相关信息，生成拉取命令：

制品名称:

制品版本:

请在命令行执行以下命令进行拉取：

```
pull StrayBirds-docker.pkg.coding.net/flask-demo/cd-demo/hello-world:latest
```

推送信息

推送人 账号 主账号

推送时间 2 天前

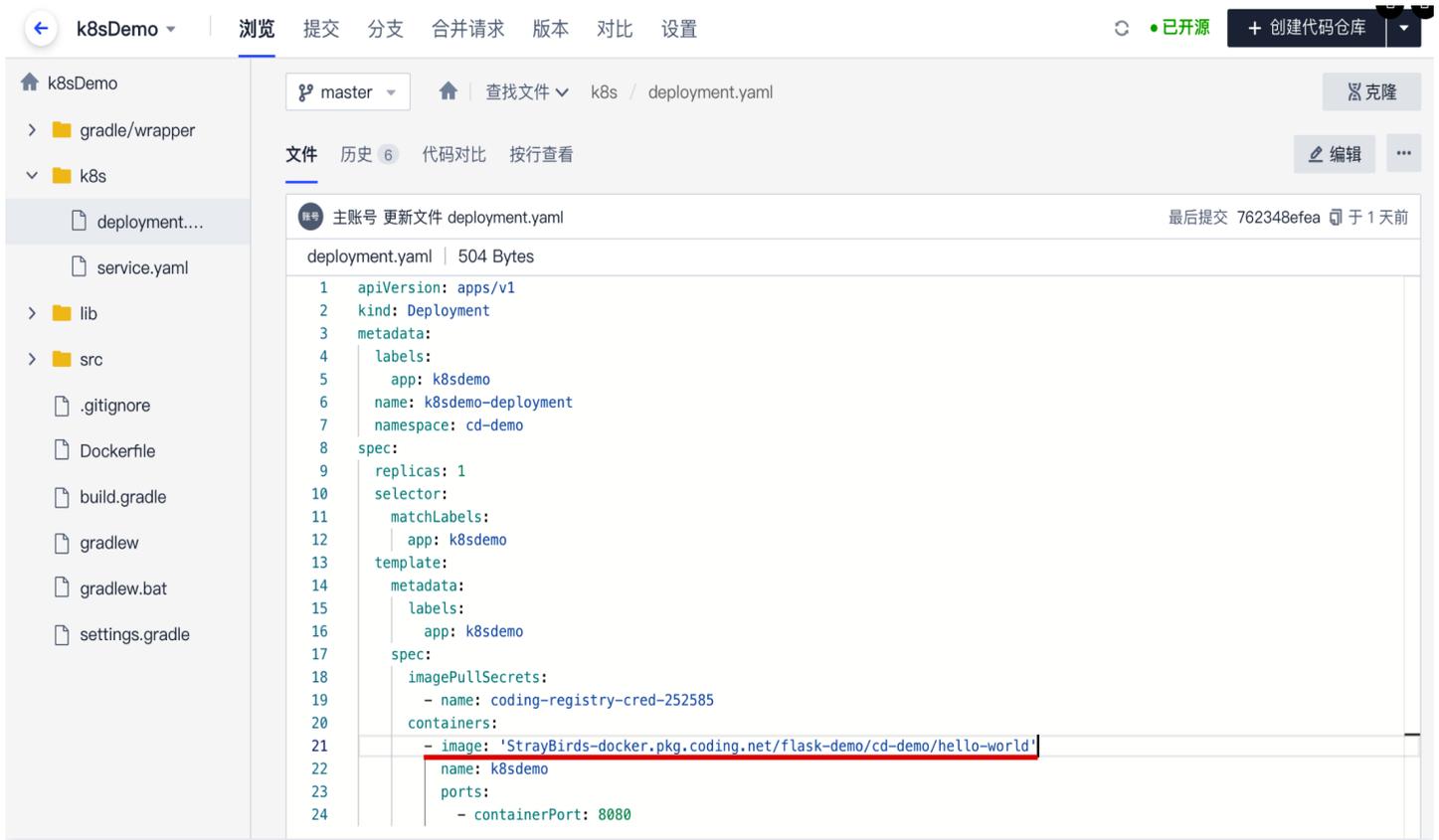
其他

大小 61.62 MB

hash sha256:44d5f1eb23a6227332267f15811f8833ecf65f3a52dd57844d30bd4353401f98

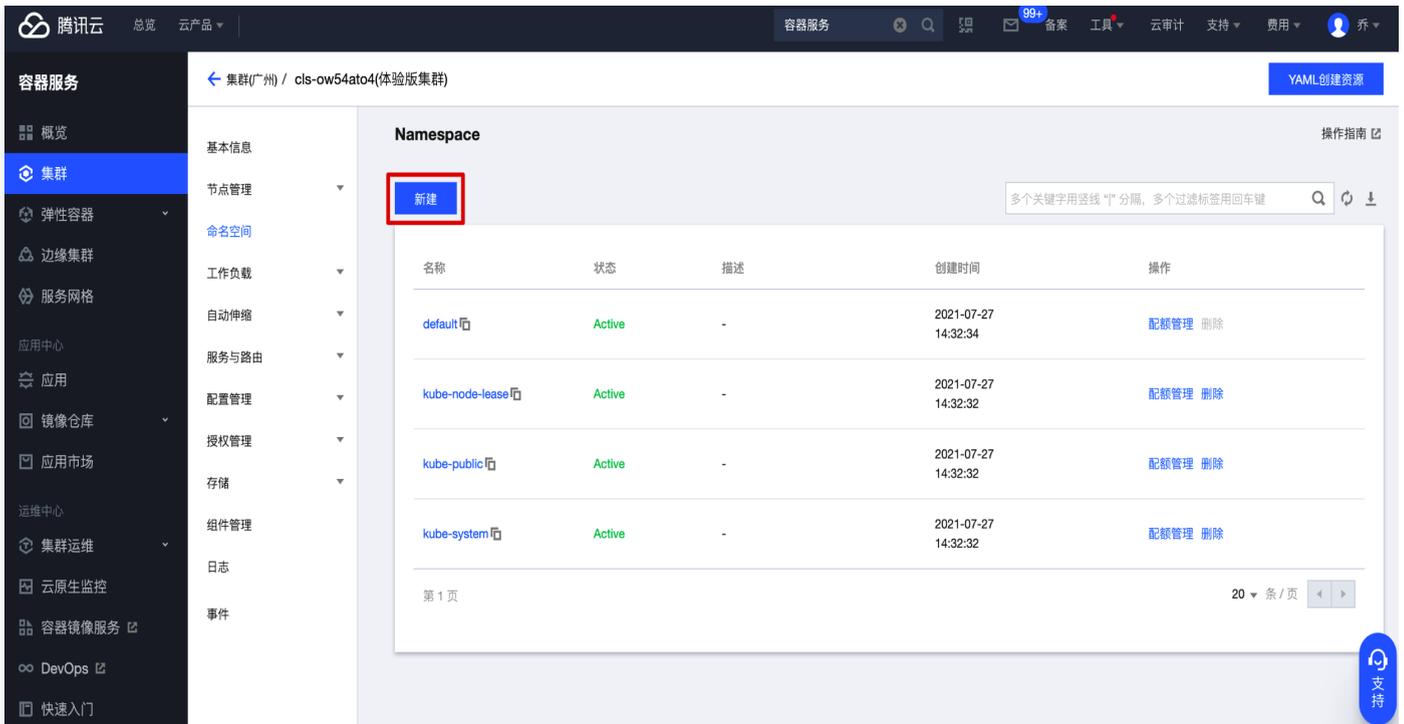
系统架构 linux/amd64

将制品的拉取地址填写至代码仓库中 `/k8s/deployment.yaml` 中的 `image` 参数中。



步骤4：创建云端容器服务

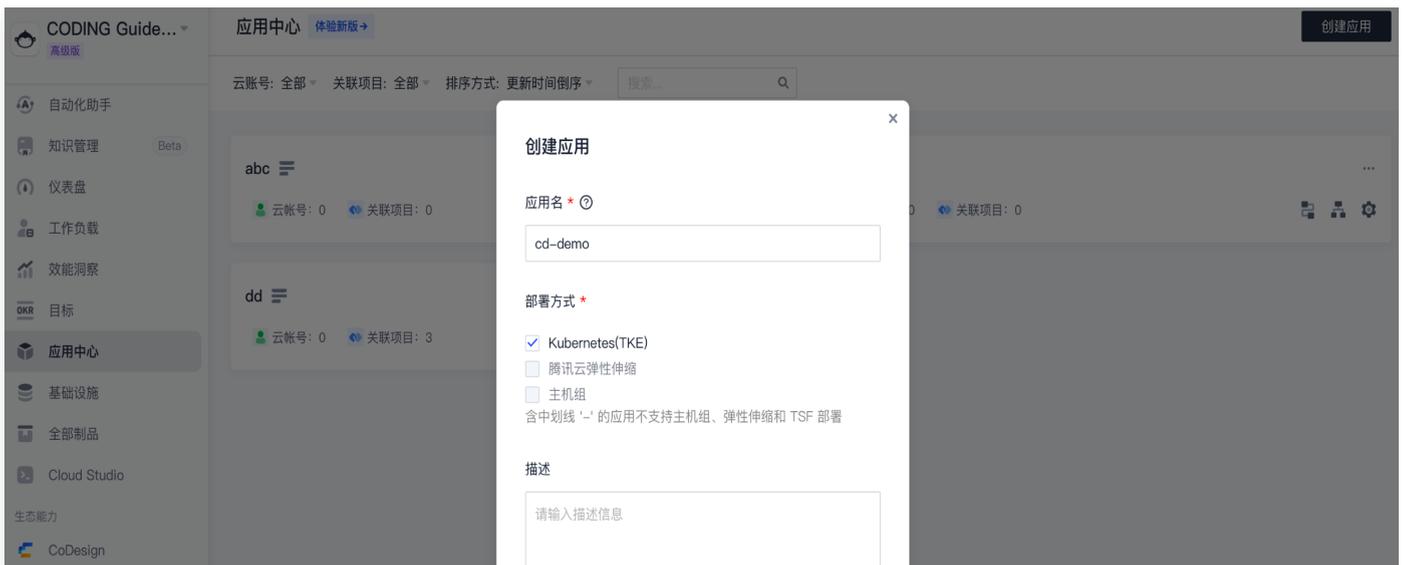
1. 前往腾讯云 [控制台](#)，单击[开通容器服务](#)。在集群中新建命名空间（Namespace）用于存储自动生成的制品仓库访问凭证，本文中所使用的集群命名为：`cd-demo`。



2. 新建命名空间后，返回 CODING 站点，单击首页左侧的[基础设施](#)，在[云账号](#)中绑定腾讯云账号。



3. 成功添加云账号后，在应用中心（旧版）中单击**创建应用**，填写应用名与选择部署方式。



4. 选择部署到 **Kubernetes 集群** 模板，填写名称与描述后完成创建。

创建部署流程

复制现有流程

Kubernetes

部署 Helm 应用到 Kubernetes 集群



部署 Deployment 和 Service 到 Kubernetes 集群



部署到 Kubernetes 集群前进行人工确认



并行部署 Deployments 和 Services



名称 *

cd-demo

描述

请输入描述

部署流程权限

- 添加用户 +
- 添加用户组 +
- 添加部门 +

名称	设置权限组 (详情)	操作
<p>暂无数据</p>		

5. 接下来需导入云账号的 `imagePullSecrets` 至代码仓库中。在基础设施 > 云账号中单击查看详情后，复制名称。

已经授权此 Kubernetes 集群访问 CODING Docker 仓库

imagePullSecrets: coding-registry-cred-252585

已授权的命名空间: cd-demo

示例用法

```
metadata:
  labels:
    app: flask-backend
    name: flask-backend-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: flask-backend
  template:
    metadata:
      labels:
        app: flask-backend
    spec:
      containers:
        - image: wzw-test-docker.pkg.coding.net/cd-show/release/flask-backend
          name: flask-backend
          imagePullSecrets:
            - name: coding-registry-cred-252585
```

关闭

6. 粘贴至代码仓库中的 `deployment.yaml` 文件中，同时在 `namespace` 参数一栏中填写在上文中所创建的命名空间 `cd-demo`。

← k8sDemo | 浏览 提交 分支 合并请求 版本 对比 设置

• 已开源 + 创建代码仓库

🏠 k8sDemo

🔍 master | 🏠 查找文件 | k8s / deployment.yaml

👤 克隆

文件 更改对比

提交 取消

👤 主账号 更新文件 deployment.yaml 最后提交 762348efea 于 1 天前

deployment.yaml | 504 Bytes

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    labels:
5      app: k8sdemo
6    name: k8sdemo-deployment
7    namespace: cd-demo
8  spec:
9    replicas: 1
10   selector:
11     matchLabels:
12       app: k8sdemo
13   template:
14     metadata:
15       labels:
16         app: k8sdemo
17     spec:
18       imagePullSecrets:
19         - name: coding-registry-cred-252585
20     containers:
21       - image: 'StrayBirds-docker.pkg.coding.net/flask-demo/cd-demo/hello-world'
22         name: k8sdemo
23         ports:
24           - containerPort: 8080
    
```

同一层级的 `service.yaml` 文件中的 `namespace` 内容也需保持一致。

← k8sDemo | 浏览 提交 分支 合并请求 版本 对比 设置

• 已开源 + 创建代码仓库

🏠 k8sDemo

🔍 master | 🏠 查找文件 | k8s / service.yaml

👤 克隆

文件 历史 4 代码对比 按行查看

编辑 ...

👤 管理员 更新文件 service.yaml 最后提交 01c504d016 于 3 个月前

service.yaml | 181 Bytes

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: k8sdemo
5    namespace: cd-demo
6  spec:
7    selector:
8      app: k8sdemo
9    ports:
10   - port: 8080
11     targetPort: 8080
12   type: LoadBalancer
    
```

步骤5：自动化发布至集群

进入部署流程配置页面，可以为此流程设定：

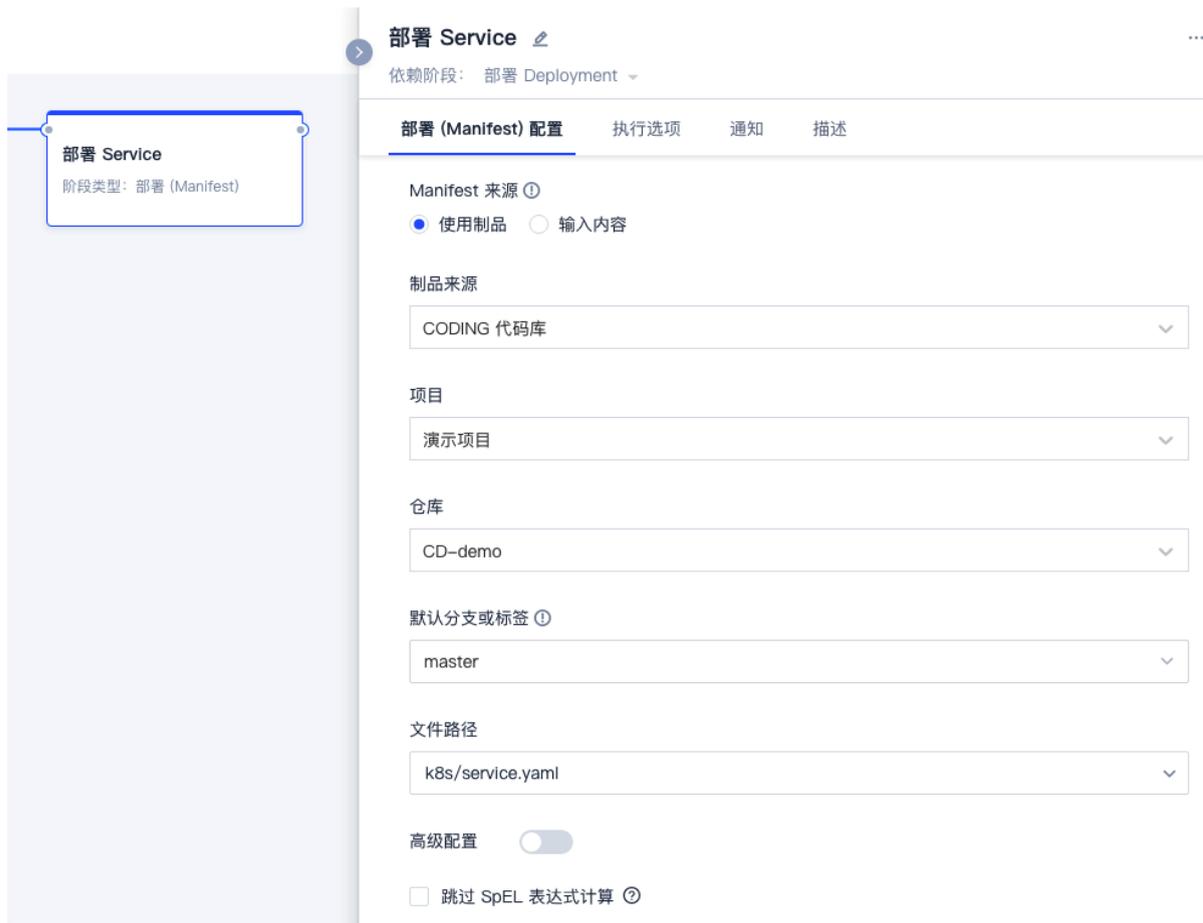
- 流程的执行选项（在此示例中保持默认即可）

- 部署 Deployment 阶段以及部署 Service 阶段所需制品
- 手动或自动触发

首先配置部署（Manifest）阶段。基础设置选择已绑定的云账号，在 Manifest 来源选择：**CODING 代码库**，填写相应的路径。

The screenshot displays the configuration interface for a Deployment stage in the Tencent Cloud DevOps console. On the left, a pipeline diagram shows a '基础配置' (Basic Configuration) stage followed by a '部署 Deployment' stage. The '部署 Deployment' stage is selected, and its configuration is shown on the right. The configuration is divided into several sections: 'Manifest 来源' (Manifest Source), '高级配置' (Advanced Configuration), and '镜像版本配置' (Image Version Configuration). The 'Manifest 来源' section is highlighted with a red box and contains the following settings: '制品来源' (Artifact Source) set to 'CODING 代码库', '项目' (Project) set to 'Flask Demo', '仓库' (Repository) set to 'k8sDemo', '默认分支或标签' (Default Branch or Tag) set to 'master', and '文件路径' (File Path) set to 'k8s/deployment.yaml'. Below this, the '高级配置' section is expanded, showing a toggle for '高级配置' (Advanced Configuration) and a checkbox for '跳过 SpEL 表达式计算' (Skip SpEL expression calculation). The '镜像版本配置' section is collapsed.

配置部署 Service 阶段时步骤同上，但在文件路径处需选择 `k8s/service.yaml` 文件。



镜像版本配置默认选择自动获取镜像来源。若设置自定义版本规则，将仅传送特定的 image 版本信息号至集群中。



完成部署阶段配置后，您可以在基础配置中选择触发器类型，选择 Docker 仓库触发器。当开发人员更新代码仓库并使用 CI 将镜像打包推送至制品库后，Docker 镜像的更新将自动触发部署流程，并将应用发布至 Kubernetes (TKE) 集群，完成后可以在基础设施页面查看并确认应用是否发布成功。

cd-demo

基础配置

制品

- ◆ k8s/deployment.yaml
master
- ◆ k8s/service.yaml
master
- ☁ hello-world
--

部署 Deployment

阶段类型: 部署 (Manifest)

基础配置

执行选项 自动触发器 启动参数 通知 描述

自动触发器

▼ CODING docker 仓库触发器 🗑

触发器启用开关

触发器类型

CODING docker 仓库触发器 ▼

- CODING docker 仓库触发器
- TCR 个人版仓库触发器
- TCR 企业版仓库触发器
- TCR Helm 仓库触发器
- + Git 仓库触发器

▼ webhook 触发器

定时触发器

- + CODING Generic 仓库触发器

▼ 通知

暂无通知

步骤6: 发布成功

发布成功后，可以查看发布的制品及启动参数及阶段执行详情等信息。

gogo ✔ 成功
部署 Deployment

基础信息

手动触发

👤 账号 主账号

📅 2021-07-27 19:53:02

🕒 25 秒

制品

- 🔗 StrayBirds-docker.pkg.coding.net/flask-demo/cd-demo/hello-world latest
- 🔗 k8s/deployment.yaml master
- 🔗 k8s/service.yaml master

阶段

✔ 成功

部署 Deployment

耗时: 19 秒

部署 Deployment

状态	成功	开始时间	2021-07-27 19:53:03
耗时	19 秒		

阶段详情

状态	脚本名称	启动时间	耗时
✔ 成功	部署 Deployment	2021-07-27 19:53:03	19 秒

DeployStatus Task Status Artifact Status

▼ Deployment k8sdemo-deployment [查看 Yaml 内容](#) [跳转查看资源详情](#)

ScalingReplicaSet

3 分钟 以前

Scaled up replica set k8sdemo-deployment-994479977 to 1

当需要查看某个资源在集群中的运行状态时，单击集群下的工作负载即可查看详情（例如工作负载的 Pod 实例，日志等信息）。

cd-demo
发布单 集群 部署流程

工作负载 服务
云账号: 全部
命名空间: 全部
类型: 全部
状态: 全部

名称	命名空间	云账号
deployment k8sdemo-deployment	cd-demo	Go

V001 StrayBirds-docker.pkg.coding.net/flask-demo/cd-demo/hello-world:latest Load E

k8sdemo-deployment-994479977

操作

基本信息

创建时间: 2021-07-27 19:53:05
 云账号: Go
 命名空间: cd-demo
 资源类型: replicaSet
 控制器: deploymentk8sdemo-deployment

镜像

StrayBirds-docker.pkg.coding.net/flask-demo/cd-demo/hello-world:latest

事件

1 x SuccessfulCreate
 - 以前
 Created pod: k8sdemo-deployment-994479977-nmkdx

LABELS

- app:k8sdemo
- app.kubernetes.io/managed-by:spinnaker
- app.kubernetes.io/name:cd-demoteam174750
- pod-template-hash:994479977

在腾讯云的容器服务中查看工作负载。

容器服务

- 概览
- 集群
- 弹性容器
- 边缘集群
- 服务网格
- 应用中心
- 应用
- 镜像仓库
- 应用市场
- 运维中心
- 集群运维
- 云原生监控
- 容器镜像服务
- DevOps
- 快速入门

集群(广州) / cls-ow54ato4(体验版集群)

Deployment

新建 监控

命名空间: cd-demo

Label格式要求: name=value, 多个关键字用竖线

名称	Labels	Selector	运行/期望Pod数量	Request/Limits	操作
<input type="checkbox"/> k8sdemo-deployment	app:k8sdemo, a...	app:k8sdemo	1/1	CPU: 无限制 / 无限制 内存: 无限制 / 无限制	更新Pod数量 更新Pod配置 更多

第 1 页 20 条 / 页

支持

文档

