持续集成

最佳实践

产品文档

腾讯云

文档目录

最佳实践
在持续集成中使用 Docker

最近更新时间：2023-05-25 10:01:59

本文为您介绍如何在持续集成中使用 Docker。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

背景介绍

在持续集成当中，除了使用 Docker 作为持续集成的 构建环境 外，您可能经常需要以 Docker 的形式运行额外的服务作为测试依赖，或在持续集成过程中构建 Docker 镜像，并推送到相关的制品库。

运行指定 Docker 镜像并在其中执行命令

在构建过程中，您可能会需要使用到公有的 Docker 镜像仓库。以下是关于如何拉取指定的 Docker 镜像执行命令的 Jenkinsfile 参考。

```
pipeline {
agent any
stages {
stage('Test') {
steps {
script {
docker.image("ubuntu").inside('-e MY_ENV=123') {
sh 'echo ${MY_ENV}'
}
}
}
}
}
}
```

运行指定 Registry 的 Docker 镜像

在构建过程中，您可能会需要使用到私有的 Docker 镜像仓库，例如希望使用 CODING 制品库中已上传的 Docker 镜像仓库。以下是相应的 Jenkinsfile 参考。

```
pipeline {
agent any
stages {
stage('Test') {
steps {
script {
docker.withRegistry('https://registry.example.com') {

// 将会从主机名 registry.example.com 拉取 my-custom-image
docker.image('my-custom-image').inside {
sh 'make test'
}
}
}
}
}
}
}
```

若所配置的 registry 对拉取操作带有鉴权，需要您提供有效的凭证 ID，以下是相应的 Jenkinsfile 参考。

```
pipeline {
agent any
stages {
stage('Test') {
steps {
script {
docker.withRegistry('https://registry.example.com', 'my-credentials-id') {

}
}
}
}
}
}
```

在持续集成过程中构建 Docker 镜像

```
pipeline {
agent any
stages {
// 需要检出代码后，才可以使用代码仓库内的 Dockerfile
stage('Checkout') {
steps {
checkout([
```

```
$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]])
}
}
stage('Build') {
steps {
script {
// 默认将使用根路径的 Dockerfile 进行构建
docker.build('my-docker-image:1.0.0')
}
}
}
}
}
```

如需为构建指定额外的参数，例如使用指定目录的 Dockerfile，以下是相应的 Jenkinsfile 参考。

```
pipeline {
agent any
stages {
// 需要检出代码后，才可以使用代码仓库内的 Dockerfile
stage('Checkout') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]])
}
}
stage('Build') {
steps {
script {
// 将使用 ./dockerfiles/Dockerfile.build 进行构建
docker.build('my-docker-image:1.0.0', '-f Dockerfile.build ./dockerfiles')
}
}
}
}
}
```

将 Docker 镜像推送到指定的 Registry

以下是相应的 Jenkinsfile 参考。

```
pipeline {
agent any
```

```
stages {
// 需要检出代码后，才可以使用代码仓库内的 Dockerfile
stage('Checkout') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
}
}

stage('Build') {
steps {
script {
docker.build('my-docker-image:1.0.0')

docker.withRegistry('https://registry.example.com', 'my-credentials-id') {
docker.image('my-docker-image:1.0.0').push()
}
}
}
}
}
}
```

使用 Docker 运行额外的服务作为测试依赖

在测试过程当中，您可以使用 Docker 来运行如 MySQL 等可被用作测试依赖的服务。下述示例使用了两个容器，一个作为 MySQL 的服务，另一个提供执行环境（使用 docker link 连接两个容器）。

```
pipeline {
agent any
stages {
stage('Test') {
steps {
script {
docker.image('mysql:5').withRun('-e "MYSQL_ROOT_PASSWORD=my-secret-pw"') { c ->
// 注意：这里 callback 的运行环境并不是上面运行的 mysql:5 环境内，而是运行 docker 的宿主机环境

// 运行第二个 mysql 作为执行环境

docker.image('mysql:5').inside("--link ${c.id}:db") {
// 这里执行的命令都是在第二个运行的 mysql docker 容器内
// 等待 mysql 服务等待
sh 'while ! mysqladmin ping -hdb --silent; do sleep 1; done'
}
```

```
// callback 内容运行完毕后，mysql docker 容器将会自动 stop 和 rm 掉
}
}
}
}
}
}
```

同时运行多个容器作为测试的依赖服务

有时候您可能不止需要一个额外的服务作为测试依赖，可以使用嵌套的方式来运行多个服务。

```
pipeline {
agent any
stages {
stage('Test') {
steps {
script {
docker.image('mysql:5').withRun('-e "MYSQL_ROOT_PASSWORD=my-secret-pw"') { c1 ->
// 注意：这里 callback 的运行环境并不是上面运行的 mysql:5 环境内，而是运行 docker 的宿主机环境

docker.image('redis').withRun('') { c2 ->
// 注意：这里 callback 的运行环境并不是上面运行的 redis 环境内，而是运行 docker 的宿主机环境
sh 'docker ps'
}
}
}
}
}
}
}
```

参考文档

如您还想进一步了解 Jenkins 当中使用 Docker 的配置方式，可以参考 Jenkins 官方文档：

- 在流水线中使用 Docker
- 流水线语法 —— 代理

如何使用 when 条件判断语法

最近更新时间：2022-03-25 15:15:58

本文为您介绍如何使用 when 条件判断语法。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

背景介绍

Jenkins 支持使用 when 语法对构建过程进行条件判断，用上一阶段的构建结果决定是否继续执行下一阶段。此语法适用于多种场景，例如：

场景	介绍
合并请求	检查代码规范，若代码不符合规范则中断构建过程，并返回错误告知。
代码合并	代码合并成功后自动进入制品发布阶段。
git tag	推送代码标签后自动进入制品部署阶段。

您可以在构建计划设置中的**流程配置**使用文本编辑器填入以下命令：

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('合并请求时：检查代码规范') {
when {
changeRequest()
}
steps {
script {
sh 'npm run lint'
}
```

```
}
}
stage('无论何时：单元测试') {
steps {
sh 'npm run test'
}
}
stage('无论何时：编译') {
steps {
sh 'npm run build'
}
}
stage('代码合并后或 git tag：构建 Docker 镜像') {
when {
anyOf {
branch 'main';
tag '*'
}
}
steps {
script {
if (env.TAG_NAME =~ /.*/ ) {
DOCKER_IMAGE_VERSION = "${env.TAG_NAME}"
} else {
DOCKER_IMAGE_VERSION = "${env.BRANCH_NAME.replace('/', '-')}-${env.GIT_COMMIT_SHORT}"
}
// 注意：创建项目时链接标识不要使用下划线，而是连字符，例如 My Project 的标识应为 my-project
// 请修改 build/my-api 为您的制品库名称和镜像名称
CODING_DOCKER_IMAGE_NAME = "${env.PROJECT_NAME.toLowerCase()}/build/my-api"
// 本项目内的制品库已内置环境变量 CODING_ARTIFACTS_CREDENTIALS_ID，无需手动设置
docker.withRegistry("https://${env.CCI_CURRENT_TEAM}-docker.pkg.coding.net", "${env.CODING_ARTIFACTS_CREDENTIALS_ID}") {
docker.build("${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION}").push()
}
}
}
}
stage('代码合并后或 git tag：部署') {
when {
anyOf {
branch 'main';
tag '*'
}
}
steps {
sh 'coscmd upload -r ./dist /'
}
}
```

```
}
}
```

在持续集成中使用 SSH

最近更新时间：2023-08-08 15:13:15

本文为您介绍如何在持续集成中使用 SSH。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

功能介绍

在持续集成中执行构建时，您可能需要通过 SSH 协议登录到一个远端服务器以执行必要的脚本或者指令。您可以在**持续集成**构建计划设置中的**流程配置**使用文本编辑器填入相关命令。

如何使用 SSH 相关指令

CODING 持续集成中支持您通过 SSH 命令操作远端服务器。

- sshCommand：在远端机器执行指定命令。
- sshPut：将当前工作空间的文件或目录放置到远端机器。
- sshGet：从远端机器获取文件或目录到当前工作空间。
- sshScript：读取本地 shell 脚本，在远端机器执行，而不是执行远端机器上的脚本，否则将会报错：does not exists。
- sshRemove：将远端机器的某个文件或目录移除。

例如，下文将演示如何通过账号和密码连接远端机器并执行 SSH 相关命令，Jenkinsfile 配置示例如下：

```
def remote = [:]
remote.name = "node"
remote.host = "node.abc.com"
remote.allowAnyHosts = true

node {
withCredentials([usernamePassword(credentialsId: 'sshUserAcct',
passwordVariable: 'password', usernameVariable: 'userName')]) {
remote.user = userName
remote.password = password

stage("SSH Steps Rocks!") {
writeFile file: 'test.sh', text: 'ls'
sshCommand remote: remote,
command: 'for i in {1..5}; do echo -n \"Loop \$i \"; date ; sleep 1; done'
sshScript remote: remote, script: 'test.sh'
```

```
sshPut remote: remote, from: 'test.sh', into: '.'
sshGet remote: remote, from: 'test.sh', into: 'test_new.sh', override: true
sshRemove remote: remote, path: 'test.sh'
}
}
}
```

如何使用 SSH 连接到远端服务

除了上述示例通过账号和密码连接远端服务外，您还可以通过 SSH 私钥来连接到远端服务，Jenkinsfile 配置示例如下：

```
def remote = [:]
remote.name = "node"
remote.host = "node.abc.com"
remote.allowAnyHosts = true

node {
withCredentials([sshUserPrivateKey(credentialsId: 'sshUser', keyFileVariable: 'identity')]) {
// ssh 登录用户名
remote.user = 'root'
// 私钥文件地址
remote.identityFile = identity
stage("SSH Steps Rocks!") {
writeFile file: 'abc.sh', text: 'ls'
sshCommand remote: remote,
command: 'for i in {1..5}; do echo -n \"Loop \$i \"; date ; sleep 1; done'
sshPut remote: remote, from: 'abc.sh', into: '.'
sshGet remote: remote, from: 'abc.sh', into: 'bac.sh', override: true
sshScript remote: remote, script: 'abc.sh'
sshRemove remote: remote, path: 'abc.sh'
}
}
}
```

拓展阅读

- 想要了解更多 Jenkinsfile 中关于 SSH 命令的内容，请参见 Jenkins 官方帮助文档。
- 想要了解更多 Jenkins 的 SSH 插件相关内容，您可以查看该插件的 官方主页。

自建静态网站

最近更新时间：2022-03-25 15:16:07

本文为您介绍如何使用持续集成自建静态网站。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的  ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

本文将主要介绍如何通过持续集成任务将静态网站发布上线。

背景

搭建简单的静态网站无需购买昂贵的服务器，通过对象存储即可快速让网站上线。静态网站分为两种类型：

- 无内容的单页应用：VUE/React SPA。
- 有内容的页面：HTML 或**程序生成 HTML**。

VUE/React SPA 难以被搜索引擎收录，不适合作为公司官网、个人博客。如果有 SEO 需求，推荐使用 MkDocs、Hexo、VUE Nuxt、React Next 框架进行网站开发。

前置准备

- CODING 项目
- 腾讯云 COS 存储桶

- 在腾讯云控制台中购买并开启 COS 存储桶服务，单击创建存储桶并将访问权限设置为**公有读私有写**。

- 在**基础配置**中开启静态网站功能。

- 在**访问设置**中获取具备存储桶权限的密钥信息。

创建构建任务

单击项目菜单栏左侧的持续集成，选择自定义构建过程即可，在持续集成设置中填入如下 Jenkinsfile 命令。


```
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('部署到腾讯云存储') {
steps {
sh "coscmd config -a ${env.COS_SECRET_ID} -s ${env.COS_SECRET_KEY} -b ${env.COS_BUCKET_NAME} -r ${env.COS_BUCKET_REGION}"
```

```
sh 'rm -rf .git'
sh 'coscmd upload -r ./ /'
}
}
}
}
```

修改环境变量

1. 在上传至腾讯云 COS 过程中涉及到相关的访问密钥，因此需要以 环境变量 的方式将其注入至构建计划配置中。

2. 在持续集成设置中的变量与缓存中添加以下参数：

变量名	含义	参考值
COS_SECRET_ID	腾讯云访问密钥 ID	stringLength36stringLength36string36
COS_SECRET_KEY	腾讯云访问密钥 KEY	stringLength32stringLength323232
COS_BUCKET_NAME	腾讯云对象存储桶	devops-host-1257110097
COS_BUCKET_REGION	腾讯云对象存储区域	ap-nanjing

3. 其中访问密钥 ID 与 KEY 填写上文中在 腾讯云控制台 > 访问设置中获取的参数。

运行持续集成

保存持续集成配置后，单击**立即构建**，您可以在构建过程中查看各运行步骤详情。

参考命令

以下是各个框架生成 HTML 文件的 Jenkinsfile 命令参考。

MKDocs

```
pipeline {
agent any
stages {
stage('检出') {...}
stage('构建') {
steps {
echo 'Markdown 转成 HTML'
sh 'pip install --upgrade mkdocs six'
sh 'mkdocs build --clean'
}
}
stage('部署到云存储') {
steps {
sh "coscmd config ..."
sh 'coscmd upload -r site/ /'
}
}
}
}
```

VUE Nuxt

```
pipeline {
agent any
stages {
stage('检出') {...}
```

```
stage('构建') {
steps {
echo 'VUE Nuxt 生成 HTML'
sh 'npm install'
sh 'npm run generate'
}
}
stage('部署到云存储') {
steps {
sh "coscmd config ..."
sh 'coscmd upload -r dist/ /'
}
}
}
}
```

VUE

```
pipeline {
agent any
stages {
stage('检出') {...}
stage('构建') {
steps {
echo 'VUE 生成 HTML'
sh 'npm install'
sh 'npm run build'
}
}
stage('部署到云存储') {
steps {
sh "coscmd config ..."
sh 'coscmd upload -r dist/ /'
}
}
}
}
```

React

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

```
echo 'React 生成 HTML'
sh 'npm install'
sh 'npm run build'
}
}
stage('部署到云存储') {
steps {
sh "coscmd config ..."
sh 'coscmd upload -r build/ /'
}
}
}
}
```

快速构建应用
GWT 应用

最近更新时间：2022-03-25 15:16:12

本文的示例会创建一个简单的 GWT 应用，并利用持续集成工具实现自动构建、测试和发布。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

前置准备

- Java
- Maven
- GWT SDK
- CODING 项目
- Generic 制品仓库

JDK

1. GWT 要求 JDK 1.6或以上版本，在终端中执行命令 java -version 检查是否已安装，输出如下图类似内容，即表示已安装。

```
└─$ java -version
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_242-b08)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.242-b08, mixed mode)
```

2. 如果没有安装，可通过 Homebrew 工具安装，命令如下：

```
brew tap adoptopenjdk/openjdk
brew cask install adoptopenjdk8
```

Maven

1. 通过命令 mvn -v 检查 Maven 版本，输出类似下图即表示已安装。

```
└─$ mvn -v
Apache Maven 3.6.3 (cecedc                              8a6ba2883f)
Maven home: /usr/local/Cellar/maven/3.6.3_1/libexec
Java version: 13.0.2, vendor: N/A, runtime: /usr/local/Cellar/openjdk/13.0.2+8_2/libexec/openjdk.jdk/Contents/Home
Default locale: zh_CN_#Hans, platform encoding: UTF-8
OS name: "mac os x", version: "10.15.4", arch: "x86_64", family: "mac"
```

2. 如果没有安装，可通过 Homebrew 工具安装，命令如下：

```
brew instll maven
```

GWT SDK

1. 可以通过 Homebrew 工具或者下载 GWT SDK 压缩包安装。写作本文时，GWT 最新稳定版本是2.8.2。您可以通过 **Homebrew** 或**手动安装**两种方式进行安装。

   Homebrew 安装

   ```
   brew install gwt
   ```

   默认安装路径为：

   ```
   $ ll /usr/local/Cellar/gwt/2.8.2
   total 88
   -rw-r--r--   1 cooper  admin    12K 10 18  2017 COPYING
   -rw-r--r--   1 cooper  admin    15K 10 18  2017 COPYING.html
   -rw-r--r--   1 cooper  admin   572B  4  8 23:03 INSTALL_RECEIPT.json
   -rw-r--r--   1 cooper  admin   3.3K 10 18  2017 about.html
   -rw-r--r--   1 cooper  admin   1.1K 10 18  2017 about.txt
   drwxr-xr-x   5 cooper  admin   160B  4  8 23:03 bin
   drwxr-xr-x  25 cooper  admin   800B  4  8 23:03 libexec
   ```

   手动安装

   i. 为方便大家下载，已将官网 GWT SDK 2.8.2版本压缩包上传到制品库，可通过如下方式下载：

   ```
   curl -L "https://coding-public-generic.pkg.coding.net/demo-gwt/generic/gwt.zip?version=2.8.2" -o gwt-2.8.2.zip
   ```

   ii. 下载后的压缩包解压到任意目录（例如：/home/user/gwt-2.8.2），并配置到 PATH 环境变量，如下所示：

   ```
   PATH=$PATH:/home/user/gwt-2.8.2/
   export PATH
   ```

   iii. 以上任一方式安装后，可执行如下命令测试是否安装成功：

   ```
   webAppCreator
   ```

2. 输出结果如下：

   ```
   $ webAppCreator
   Missing required argument 'moduleName'
   Google Web Toolkit 2.8.2
   WebAppCreator [-[no]overwriteFiles] [-[no]ignoreExistingFiles] [-templates template1,template2,...] [-out dir] [-junit pathToJUnitJar] [-[no]maven] [-[no]ant] moduleName

   where
     -[no]overwriteFiles       Overwrite any existing files. (defaults to OFF)
     -[no]ignoreExistingFiles  Ignore any existing files; do not overwrite. (defaults to OFF)
     -templates                Specifies the template(s) to use (comma separeted). Defaults to 'sample,ant,eclipse,readme'
     -out                      The directory to write output files into (defaults to current)
     -junit                    Specifies the path to your junit.jar (optional)
     -[no]maven                DEPRECATED: Create a maven2 project structure and pom file (default disabled). Equivalent to specifying 'maven' in the list of templates. (defaults to OFF)
     -[no]ant                  DEPRECATED: Create an ant configuration file. Equivalent to specifying 'ant' in the list of templates. (defaults to OFF)
   and
     moduleName                The name of the module to create (e.g. com.example.myapp.MyApp)
   ```

使用 Intellij IDEA 创建 GWT 应用

IDEA 默认支持创建 GWT 应用，本文使用命令行工具创建项目，不依赖具体 IDE，通过 IDE 创建 GWT 应用的方式可参见 Intellij IDEA 相关介绍。

步骤一：创建 GWT 应用

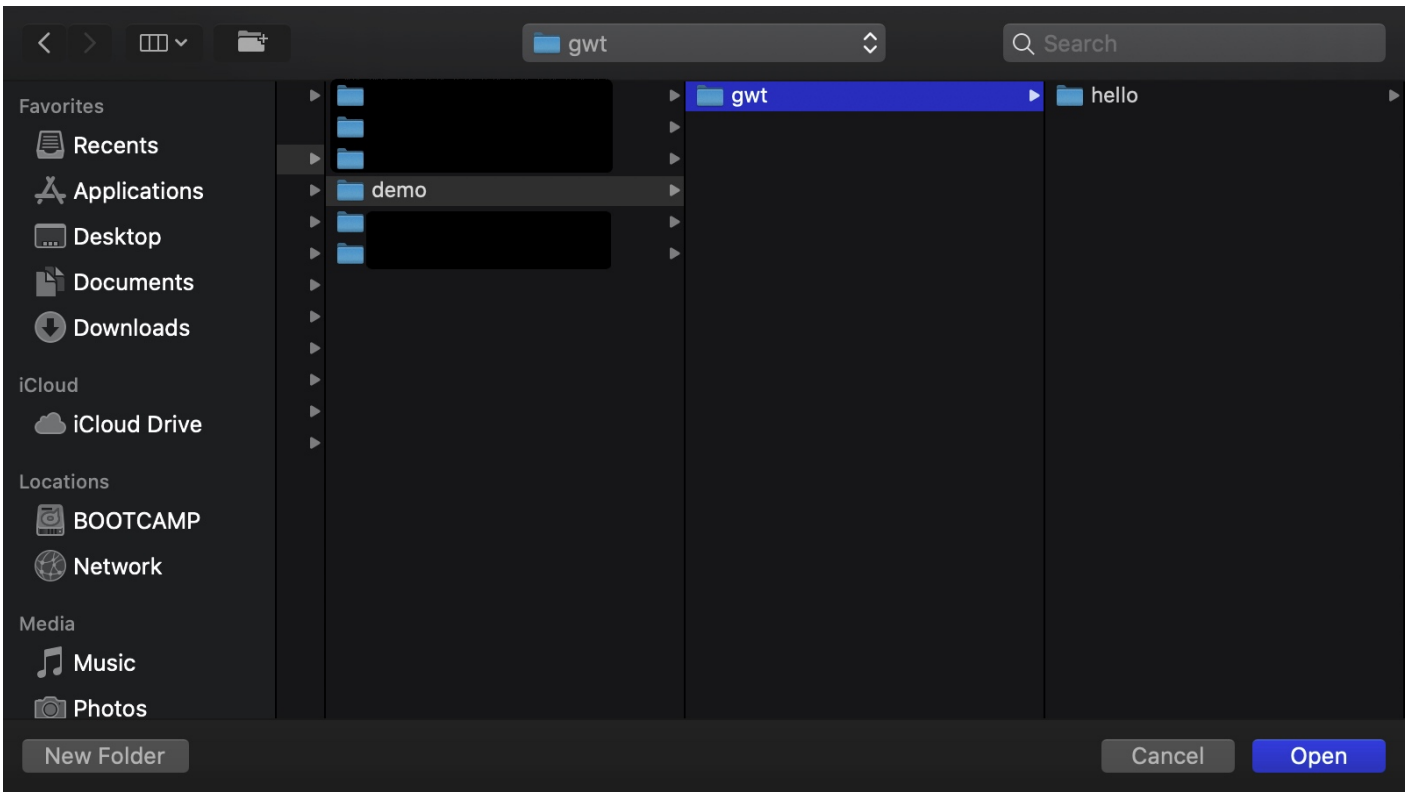
1. 使用命令行工具创建 GWT Maven 项目：

```
webAppCreator -out hello -templates maven,sample,readme com.demo.gwt.HelloWorld
```

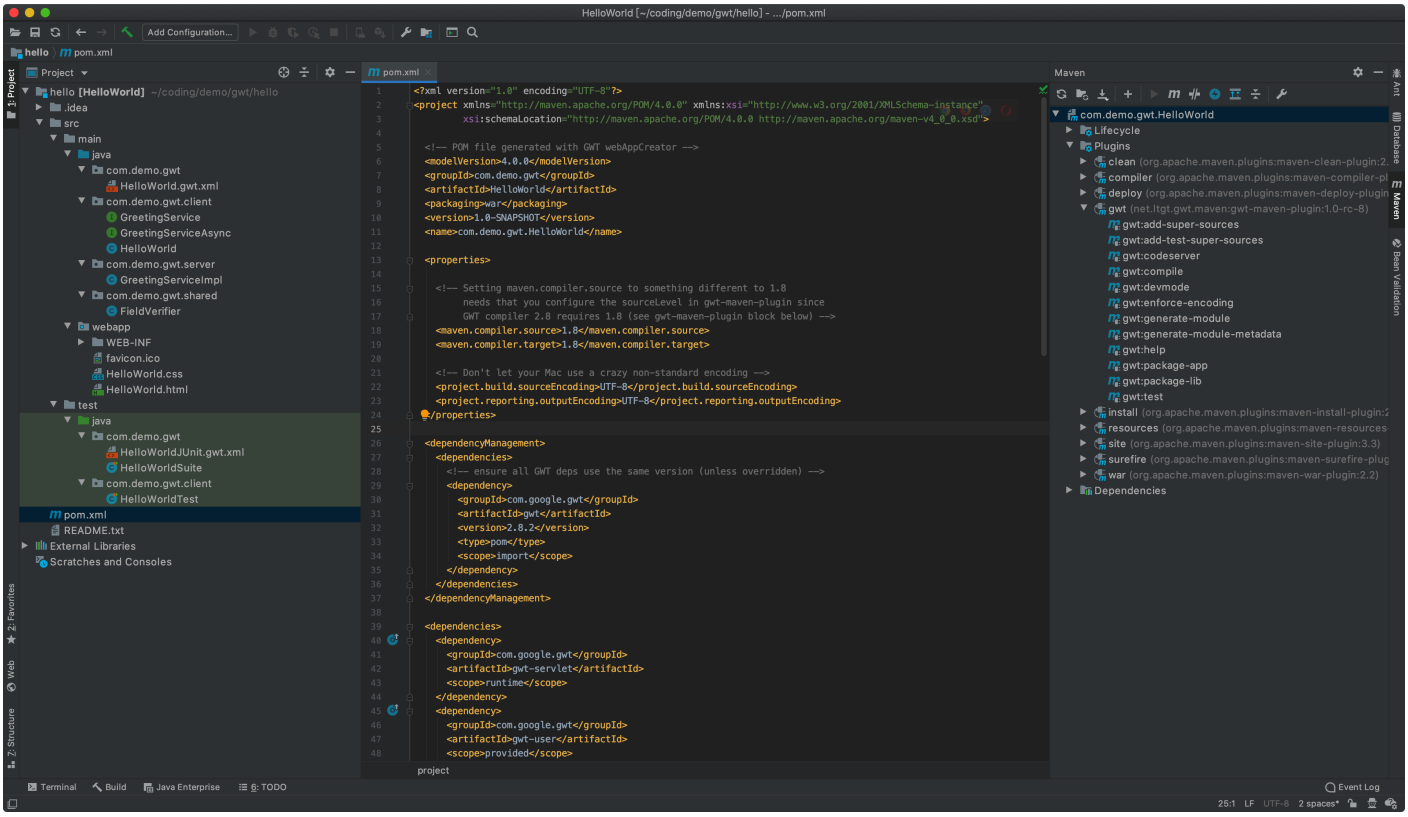
```
└─$ webAppCreator -out hello -templates maven,sample,readme com.demo.gwt.HelloWorld
Generating from templates: [maven, readme, _sample-test, sample]
Created directory hello
Created directory hello/src/test/java
Created directory hello/src/test/java/com/demo/gwt
Created directory hello/src/test/java/com/demo/gwt/client
Created directory hello/src/main/java
Created directory hello/src/main/java/com/demo/gwt
Created directory hello/src/main/java/com/demo/gwt/client
Created directory hello/src/main/java/com/demo/gwt/server
Created directory hello/src/main/java/com/demo/gwt/shared
Created directory hello/src/main/webapp
Created directory hello/src/main/webapp/WEB-INF
Created file hello/pom.xml
Created file hello/README.txt
Created file hello/src/test/java/com/demo/gwt/HelloWorldJUnit.gwt.xml
Created file hello/src/test/java/com/demo/gwt/HelloWorldSuite.java
Created file hello/src/test/java/com/demo/gwt/client/HelloWorldTest.java
Created file hello/src/main/java/com/demo/gwt/HelloWorld.gwt.xml
Created file hello/src/main/java/com/demo/gwt/client/GreetingService.java
Created file hello/src/main/java/com/demo/gwt/client/GreetingServiceAsync.java
Created file hello/src/main/java/com/demo/gwt/client/HelloWorld.java
Created file hello/src/main/java/com/demo/gwt/server/GreetingServiceImpl.java
Created file hello/src/main/java/com/demo/gwt/shared/FieldVerifier.java
Created file hello/src/main/webapp/WEB-INF/web.xml
Created file hello/src/main/webapp/HelloWorld.css
Created file hello/src/main/webapp/HelloWorld.html
Created file hello/src/main/webapp/favicon.ico
```

2. 在 IDEA 中打开上面创建的项目，依次单击：

```
File --> Open --> hello（项目名）
```



3. 打开后可以看到项目目录结构：



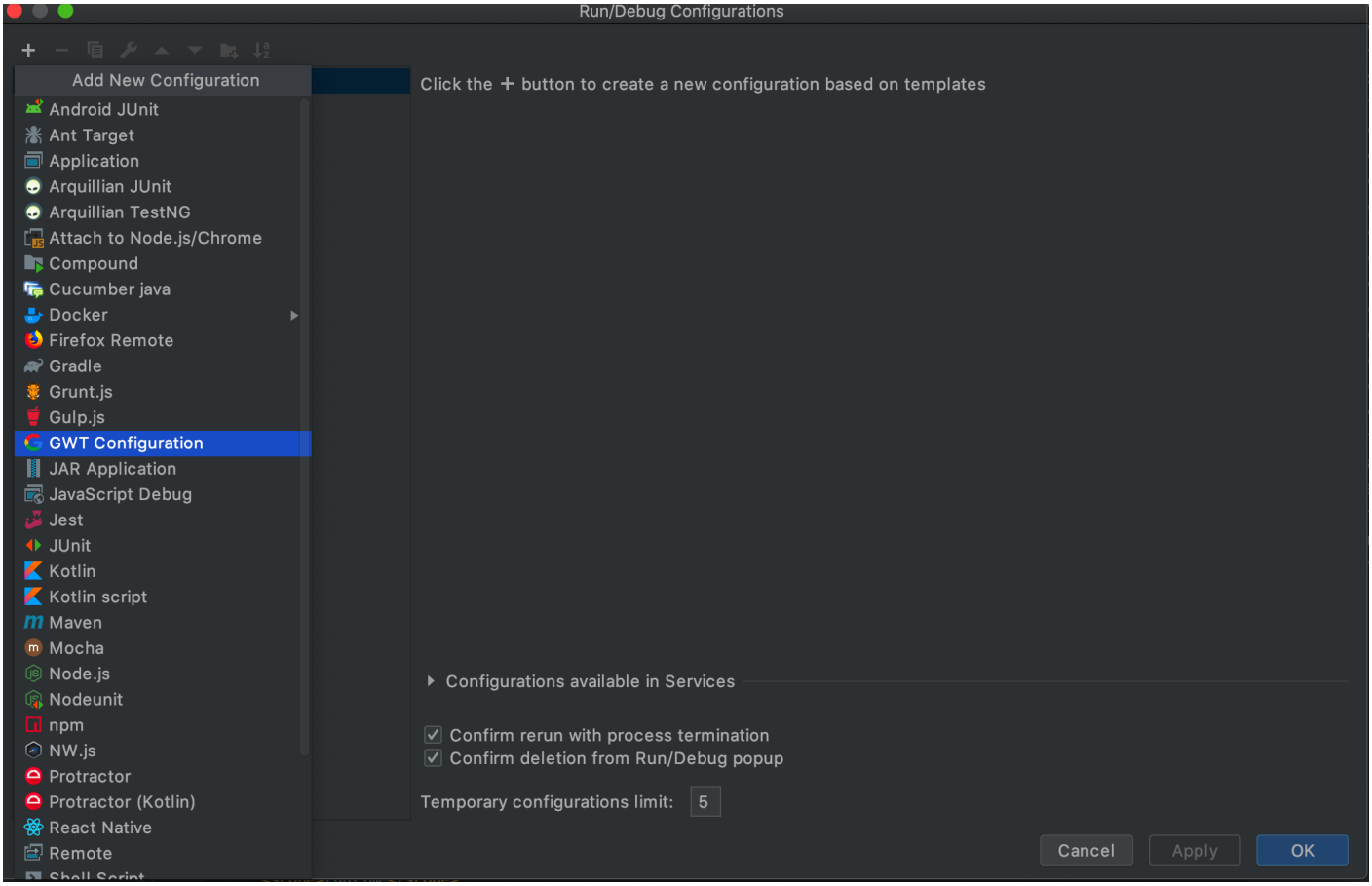
步骤二：运行 GWT 应用

GWT 项目文件主要由四部分组成（见下表），可以根据自己的需要修改，为避免增加复杂度，这里不做修改。

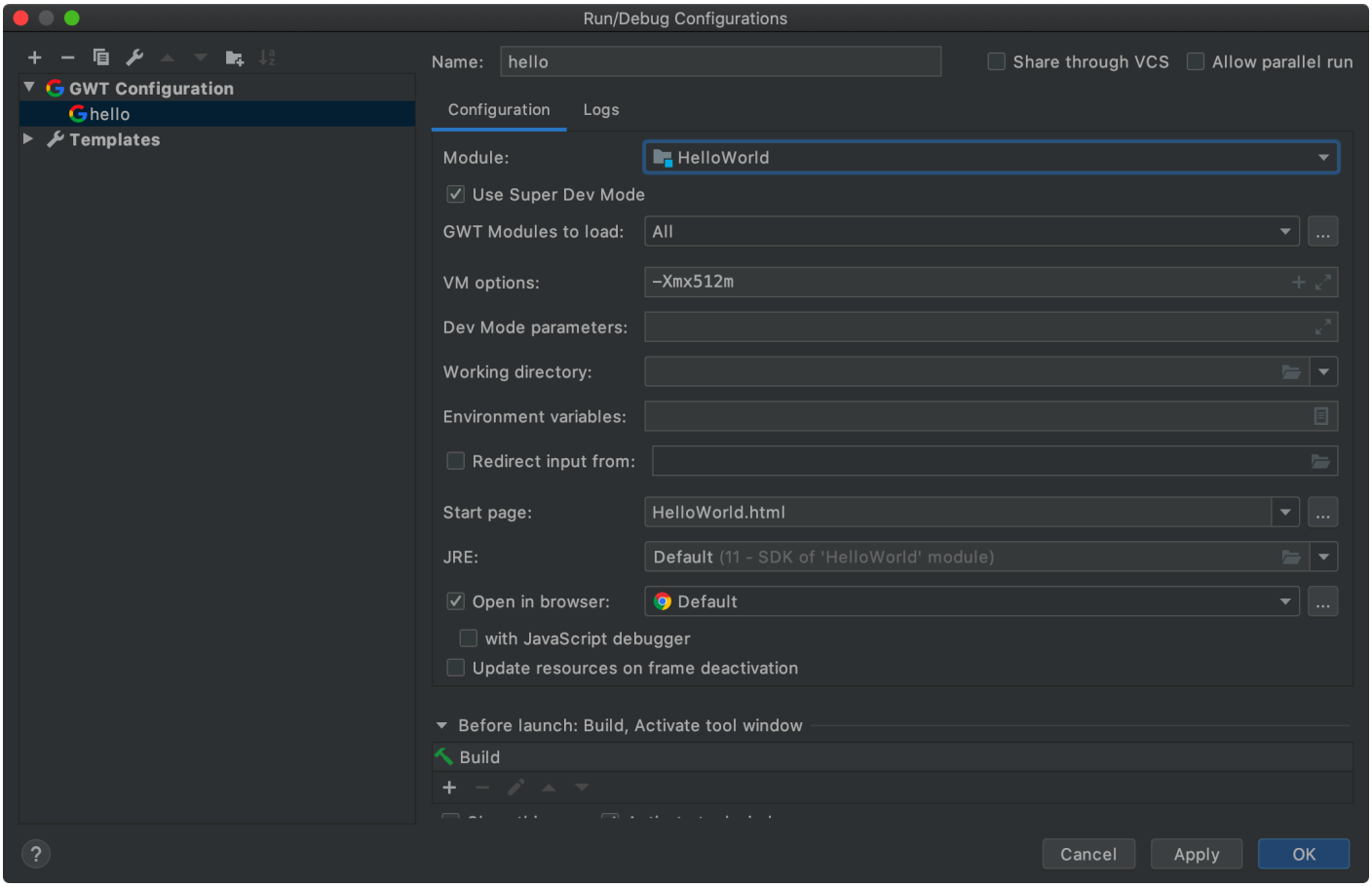
内容	说明	位置

内容	说明	位置
模块描述符	用于配置 GWT 应用，XML 格式	src/main/java/com/demo/gwt/HelloWorld.gwt.xml
公共资源	GWT 模块引用的文件，如 HTML 页面、CSS 样式或图像	src/main/webapp
客户端代码	实现应用程序业务逻辑的 Java 代码，GWT 编译器将其转换为 JavaScript，最终在浏览器中运行	src/main/java/com/demo/gwt/client
服务端代码	可选的，如果应用不需要服务端处理，不用提供	src/main/java/com/demo/gwt/server

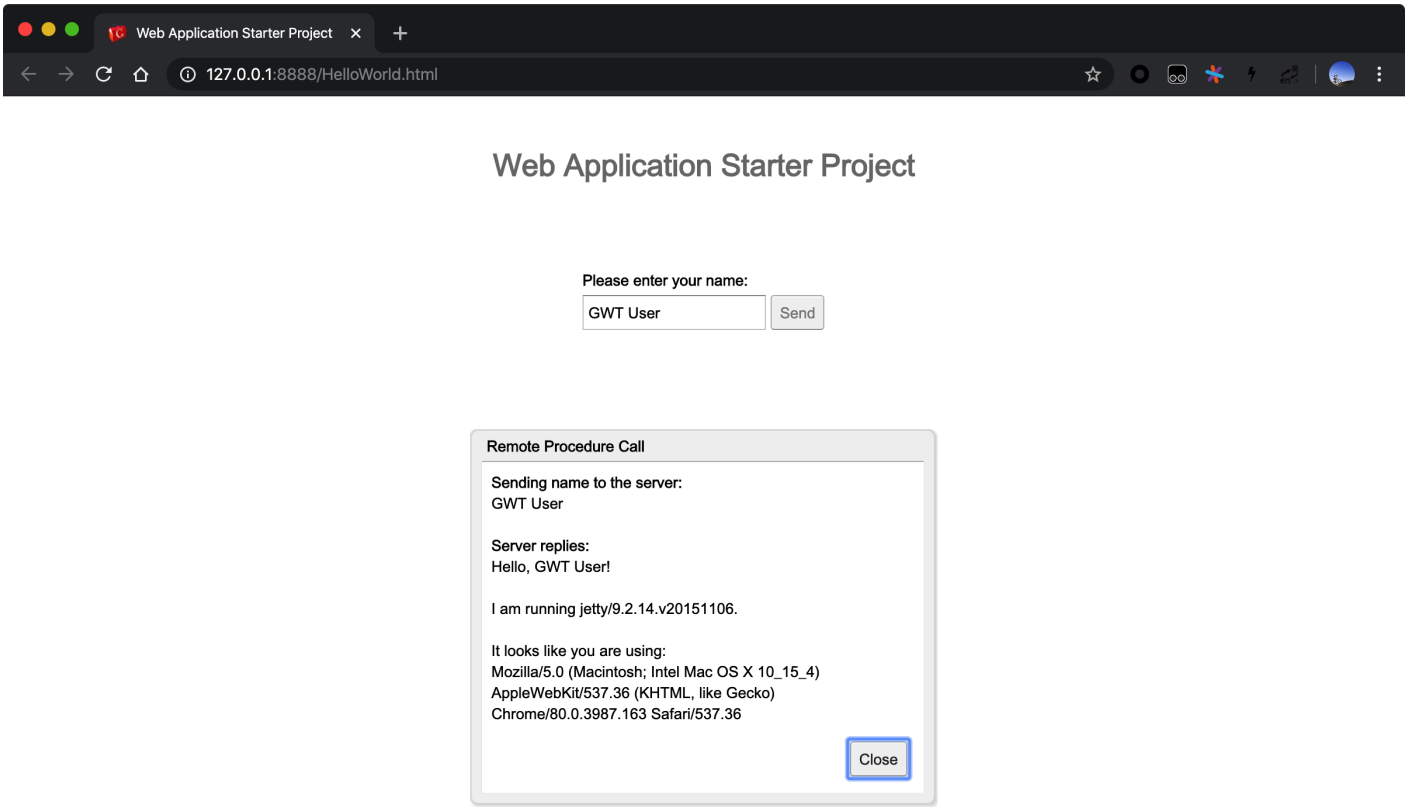
1. 在 IDEA 菜单栏中选择编辑配置，打开运行配置弹窗，添加 GWT 配置。



2. 修改配置名，选择 HelloWorld 模块，然后单击 OK 保存配置并退出弹窗。



3. 选择上面创建的 GWT 运行配置，单击运行，IDEA 会自动打开浏览器，运行应用。



步骤三：利用持续集成进行自动构建、测试、发布

在已创建的项目中新建代码仓库，您可以通过导入示例代码或手动上传代码，将 GWT 应用代码上传至项目中的代码仓库中。

1. 上传示例代码

   在新建代码仓库时选择**导入外部仓库**，粘贴 示例仓库 中的代码地址。

Git 仓库 URL *

https://e.c[redacted]gwt.git

仓库名称 *

demo-gwt 8/100

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 通过模版创建 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

2. 手动上传代码

请参见 快速入门，使用 Git 命令将上文中创建的 GWT 应用代码上传至 CODING 代码仓库中。

接下来前往**持续集成**新建构建计划，单击右侧**新建构建计划**。可根据自己需要选择合适模板，此处选择**自定义构建过程**。

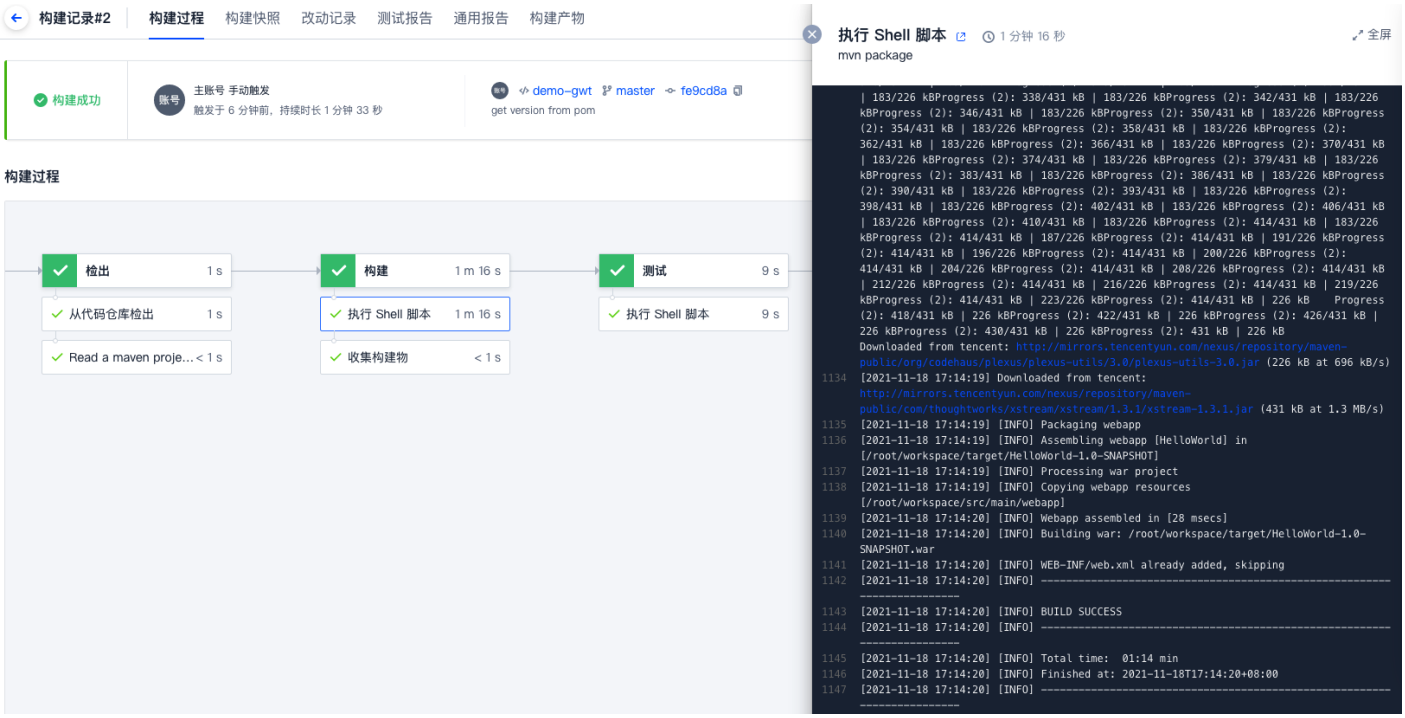
在基础信息中的代码源选择已上传的应用。

参考以下配置，在流程配置中修改静态 Jenkinsfile 配置。

```
def ARTIFACT_ID = ""
def VERSION = ""
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
]]
])

script {
def pom = readMavenPom()
ARTIFACT_ID = pom.getArtifactId()
VERSION = pom.getVersion()
}
}
}

stage('构建') {
steps {
sh 'mvn package'

archiveArtifacts(artifacts: "target/${ARTIFACT_ID}-${VERSION}.war", fingerprint: true)
}
}

stage('测试') {
steps {
sh 'mvn gwt:test'
}
}

stage('发布到 generic 制品库') {
steps {
codingArtifactsGeneric(
files: "${ARTIFACT_ID}-${VERSION}.war",
repoName: "${env.GENERIC_REPO_NAME}",
version: "${VERSION}",
workspace: "/root/workspace/target"
)
}
}
}
}
```

需要按照实际制品仓库的情况修改 GENERIC_REPO_NAME 与 VERSION 参数。您还可以在**触发规则**中配置自动触发持续集成任务的条件。配置完成后，单击立即构建。



您可以在构建过程中查看各个环节的运行情况。构建完成后，打开制品库页面还可以看到已发布的制品文件。

总结

在本次教程中，我们创建了简单的 GWT 应用，熟悉了 GWT 项目结构及其命令行工具的使用，而且借助 CODING 持续集成实现了应用的自动构建、测试和发布。

自动发布 Electron 应用

最近更新时间：2022-04-13 14:43:39

本文为您介绍如何使用持续集成自动发布 Electron 应用。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

环境准备

本文涉及到 MacOS 程序开发，因此需要将苹果电脑接入至自定义节点中，详情请参见 自定义节点。

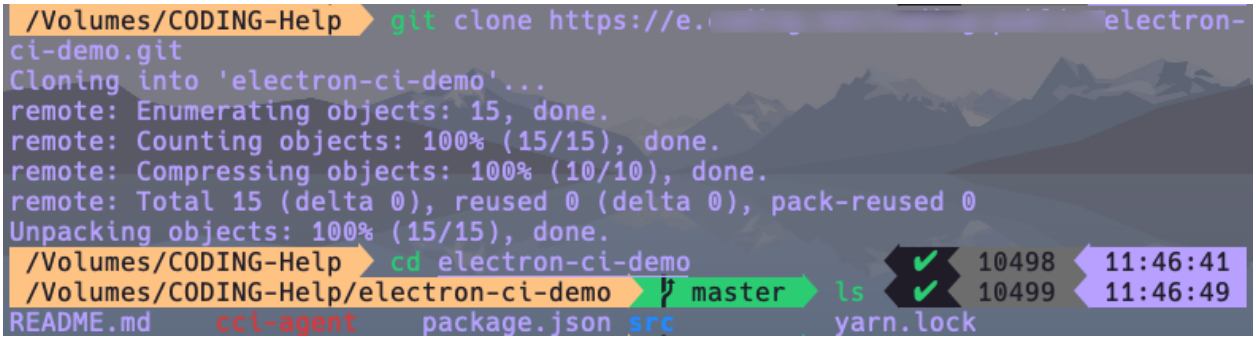
- node 环境
- yarn 环境
- CODING 项目
- 将 示例仓库 导入或关联外部仓库至 CODING 仓库
- CODING 制品库

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

初始化仓库代码

示例仓库为 Electron 官方提供的一系列的脚手架及 CLI 工具，拉取至本地。



手动编译与制品上传

一般软件工程的开发流程为：**开发 > 部署 > 测试**。由于原生应用给到测试团队的不能像 Web 一样仅仅是一个链接，因此我们需要把打包好的安装包或其他交付物以制品的形式交付给测试部门。当开发人员将制品上传至制品仓库后，由测试人员下载最新版本即可。

切换至示例项目的根目录，使用 yarn install 命令自动安装相关依赖。

```
/Volumes/CODING-Help/electron-ci-demo  master  yarn install
yarn install v1.22.4
[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...
[4/4] 🔨  Building fresh packages...
[4/4] ⣾ electron
```

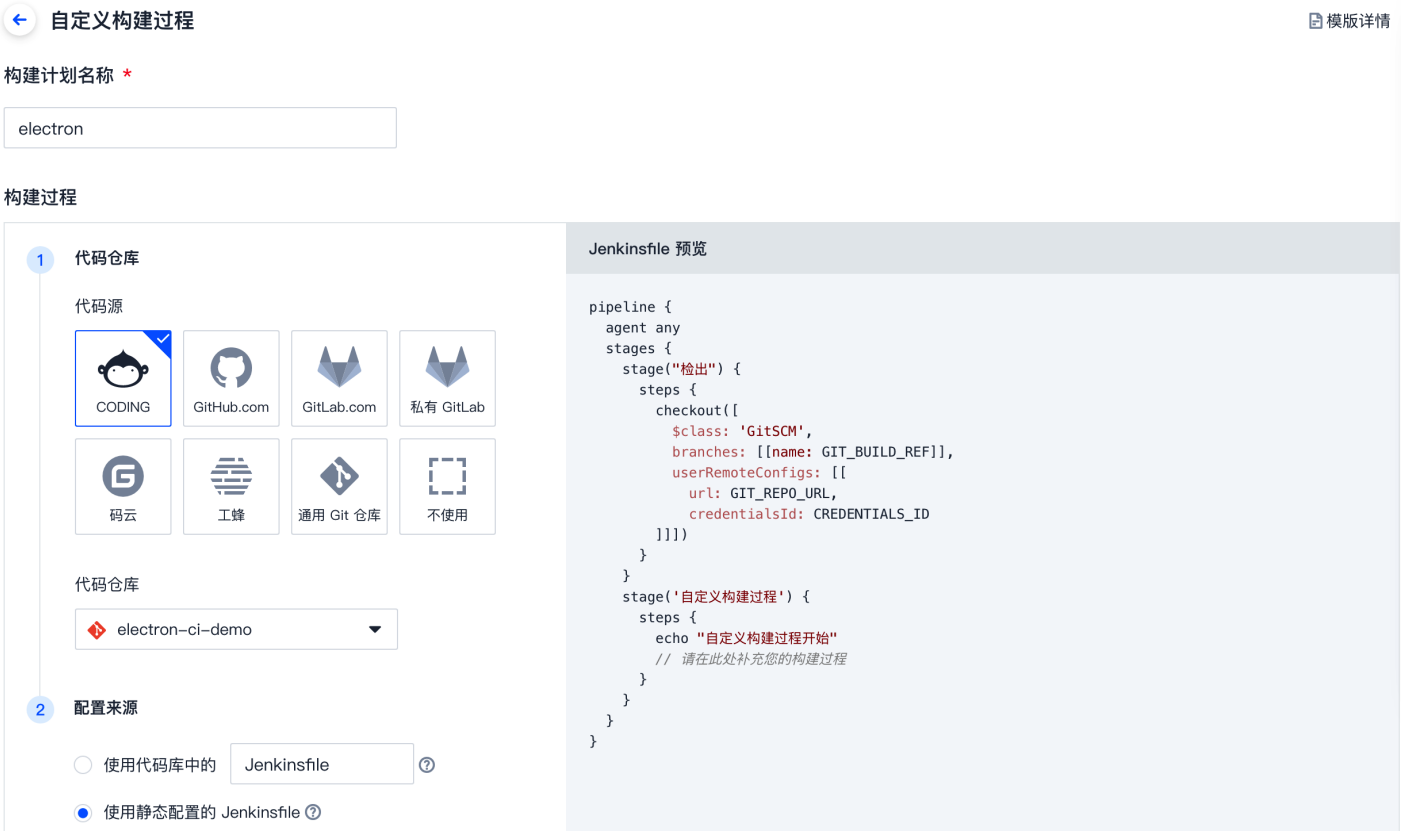
示例项目中的 package.json 命令已定义了构建命令，因此仅需运行 yarn dist 命令即可编译出 Windows 和 Mac 平台的安装包。文件夹中的 .dmg(macOS) 及 .exe(windows) 文件是最终需要部署的安装包。将构建出来的安装包拖拽上传至团队 Generic 类型制品库 中以方便测试及产品验收。

至此已成功在本地中构建了交付物，并且已上传至制品仓库中。但如果对交付物内容进行二次开发修改，进行手动构建过程并将构建物再次上传至制品仓库未免显得过于繁琐，不能够满足敏捷研发的需要。如果能够在将 commit 推送至代码仓库时就能够自动构建，并且还能够自动将交付物部署至制品仓库，那么可以解放开发的部分劳动，并实现构建内容的回退及追溯功能。

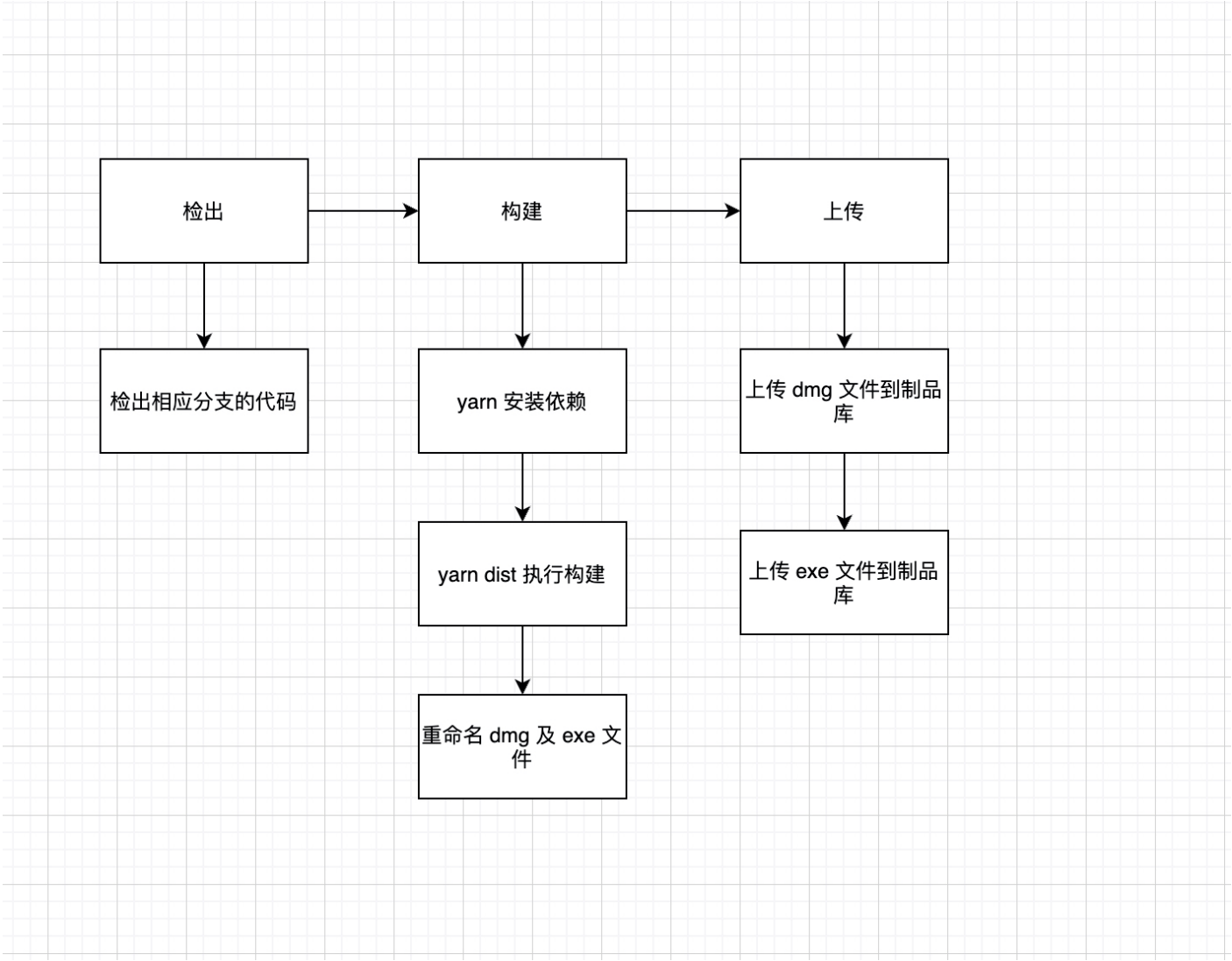
使用持续集成

使用 CODING 持续集成，可以帮助我们很好的自动化这些重复的工作。我们需要做的只是编写好一次编译、测试脚本，或者 Jenkinsfile，那么之后我们只需要提交代码至远端，即可自动触发构建，并及时编译、测试。如果在期间发现代码有问题，持续集成还能及时通过多种形式通知到您，让您及时发现错误、改正错误，而不是等待项目正式上线后才发现问题。

1. 创建新的构建计划，选择**自定义构建过程**，代码仓库选择已导入的 示例仓库。



2. 勾选**前往配置详情**后，单击确定进入**流程配置 > 文本编辑器**中。此时我们可以将手动操作过程转译为 Jenkinsfile，将制品的上传过程自动化。



3. 填写 Jenkinsfile 后，在**触发规则**中还可以设定自动触发条件。

Jenkinsfile

您可以参考此配置并填写至配置详情中。

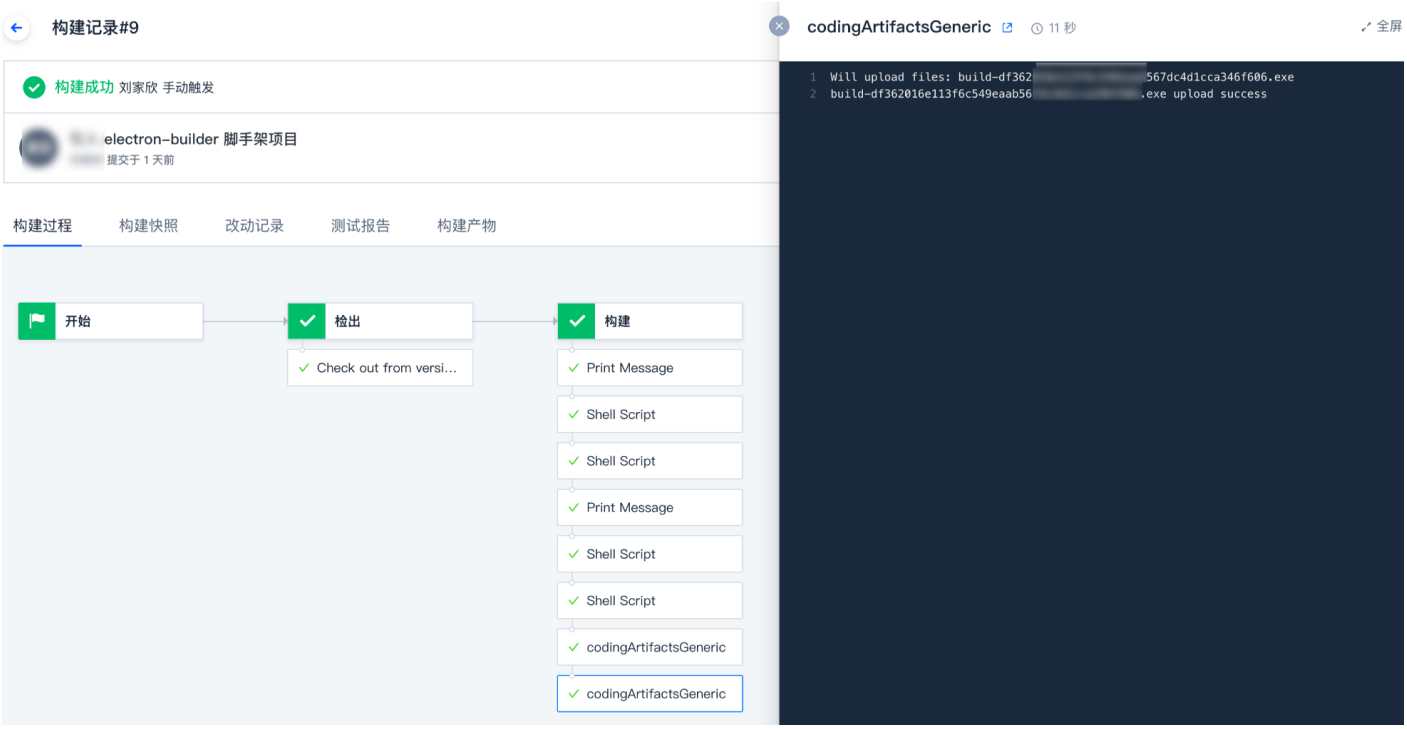
```
pipeline {
agent any
stages {
stage('检出') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[
url: env.GIT_REPO_URL,
credentialsId: env.CREDENTIALS_ID
]]])
}
}
stage('构建') {
```

```
steps {
echo '构建中...'
sh 'yarn'
sh 'yarn dist'
echo '构建完成.'
sh 'mv ./dist/*.dmg build.dmg'
sh 'mv ./dist/*.exe build.exe'
}
}
stage('上传') {
steps {
codingArtifactsGeneric(files: '*.dmg', repoName: 'electron-build', credentialsId: '${env.CODING_ARTIFACTS_CREDENTIALS_ID}', withBuildProps: true, version: '${env.GIT_BUILD_REF}')
codingArtifactsGeneric(files: '*.exe', repoName: 'electron-build', credentialsId: '${env.CODING_ARTIFACTS_CREDENTIALS_ID}', withBuildProps: true, version: '${env.GIT_BUILD_REF}')
}
}
}
}
```

配置构建节点

1. 因为 CODING 提供的构建云服务器仅支持 Linux 系统，而此项目需要使用到 MacOS 环境，因此需要将 MacOS 作为自定义节点接入至节点池中。单击侧边栏持续集成中的构建节点选项，接入新节点。

2. 在持续集成配置中选定自定义节点，开始构建。



3. 我们可以看到，构建成功后制品库出现了两个以 build + commit hash 为名字的制品，这就是我们构建出来的安装包。

总结

通过持续集成，我们能够在修改代码的时候就能够触发持续集成任务，自动发布交付物。

每日一句小应用

最近更新时间：2022-03-25 15:16:20

本文为您介绍如何使用持续集成构建每日一句小应用。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

前置准备

- 示例仓库
- 本地安装 Docker 环境
- Docker 类型制品仓库

进入项目

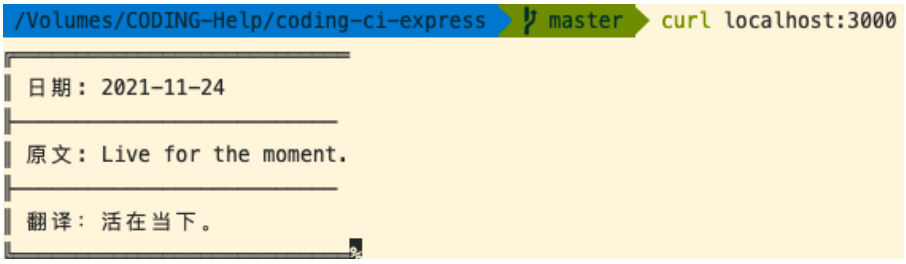
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

本地运行

1. 将仓库拉取至本地后，运行以下命令：

```
# 安装
npm install
# 运行
npm run dev
# 部署
curl localhost:3000
```

2. 本地终端中出现以下效果：



3. 在本地运行无误后，下文将演示如何打包为 Docker 类型制品，上传至制品仓库后供团队其他成员使用。

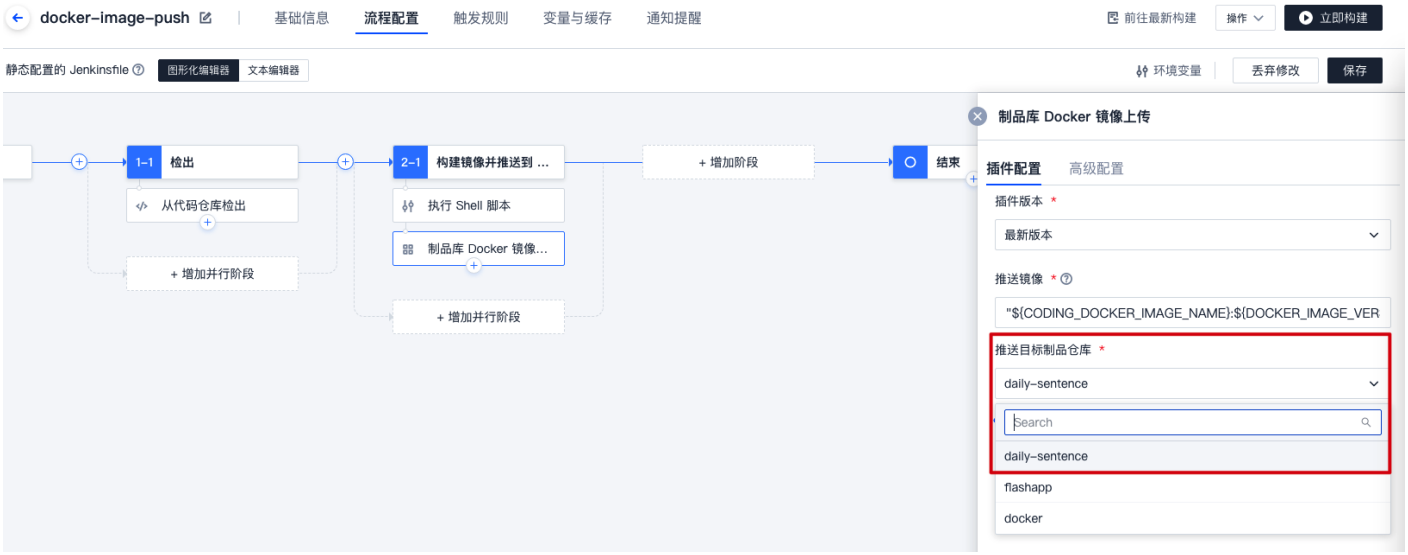
导入示例仓库

在创建代码仓库时选择**导入外部仓库**，粘贴示例仓库地址。

创建构建计划

1. 因示例仓库中已内置 Dockerfile 文件，因此可以直接使用**CODING Docker 镜像推送**模板。

2. 选择已创建的 Docker 仓库，您也可以在**图形化编辑器**中调整目标制品仓库。



Jenkinsfile 参考

若希望通过手动编写配置过程，您可以参考下列构建流程文件：

```
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('打包镜像') {
steps {
sh "docker build -t ${env.DOCKER_IMAGE_NAME}:${env.CI_BUILD_NUMBER} ."
}
}
stage('推送到制品库') {
steps {
script {
docker.withRegistry("https://${env.CODING_DOCKER_REG_HOST}", "${env.CODING_ARTIFACTS_CREDENTIALS_ID}") {
docker.image("${env.DOCKER_IMAGE_NAME}:${env.CI_BUILD_NUMBER}").push()
}
}

}
}
}
environment {
CODING_DOCKER_REG_HOST = "${env.CCI_CURRENT_TEAM}-docker.pkg.${env.CCI_CURRENT_DOMAIN}"
DOCKER_IMAGE_NAME = "${env.PROJECT_NAME.toLowerCase()}/${env.DOCKER_REPO_NAME}/hello-world"
```

```
}
}
```

查看制品

在制品仓库中您可以查看已上传的应用制品，团队中的其他成员拉取此制品后即可直接使用了。

更多操作

我们可以将该应用和终端结合使用，这样就可以在每次启动终端的时候就可以看到“每日一句”了。

```
# 先启用服务
npm run dev
# 写入终端
echo "curl localhost:3000" >> ~/.zshrc
# or
echo "curl localhost:3000" >> ~/.bashrc
```

自动化发布 AI 应用

最近更新时间：2022-03-25 15:16:26

本文为您介绍如何使用持续集成自动发布 AI 应用。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

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

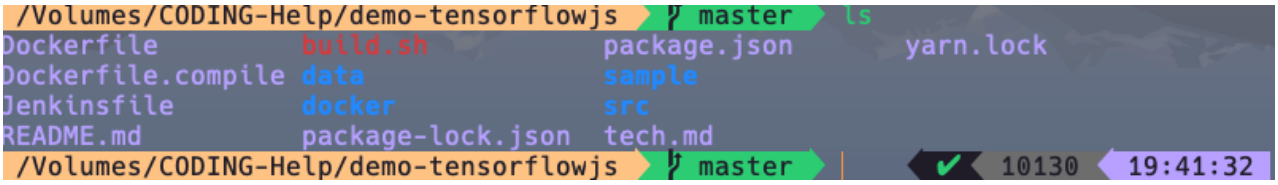
进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

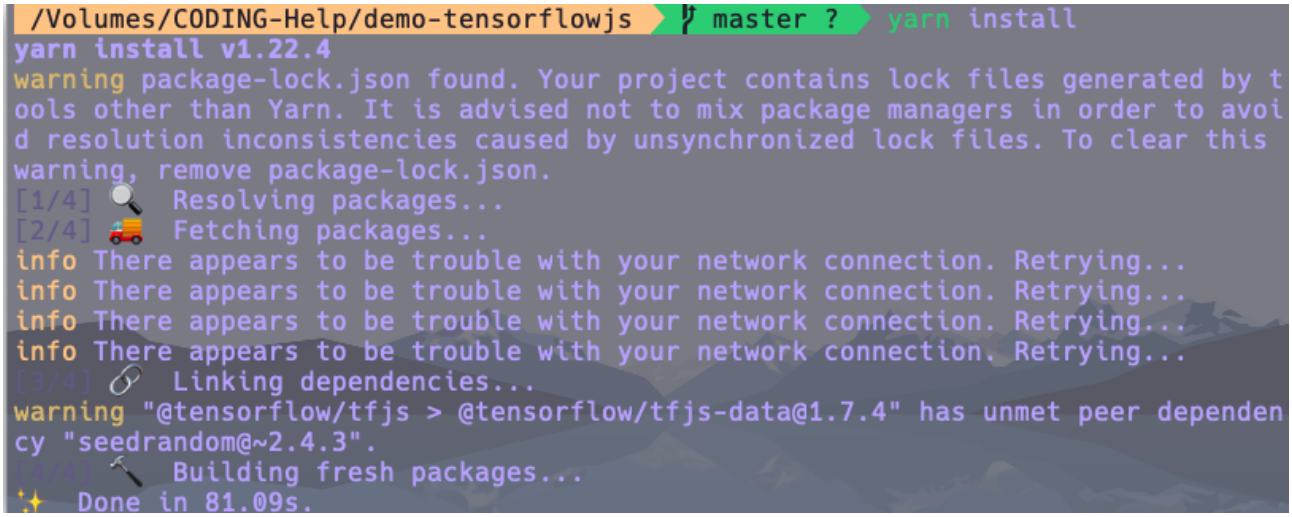
初始化

1. 在创建代码仓库时选择**导入外部仓库**，粘贴示例仓库地址。

2. 导入成功后，将代码拉取至本地中。



3. 接下来可以运行 yarn install 命令并在本地安装依赖。



4. 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

1. 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。

2. 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149Mb。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://s
t/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

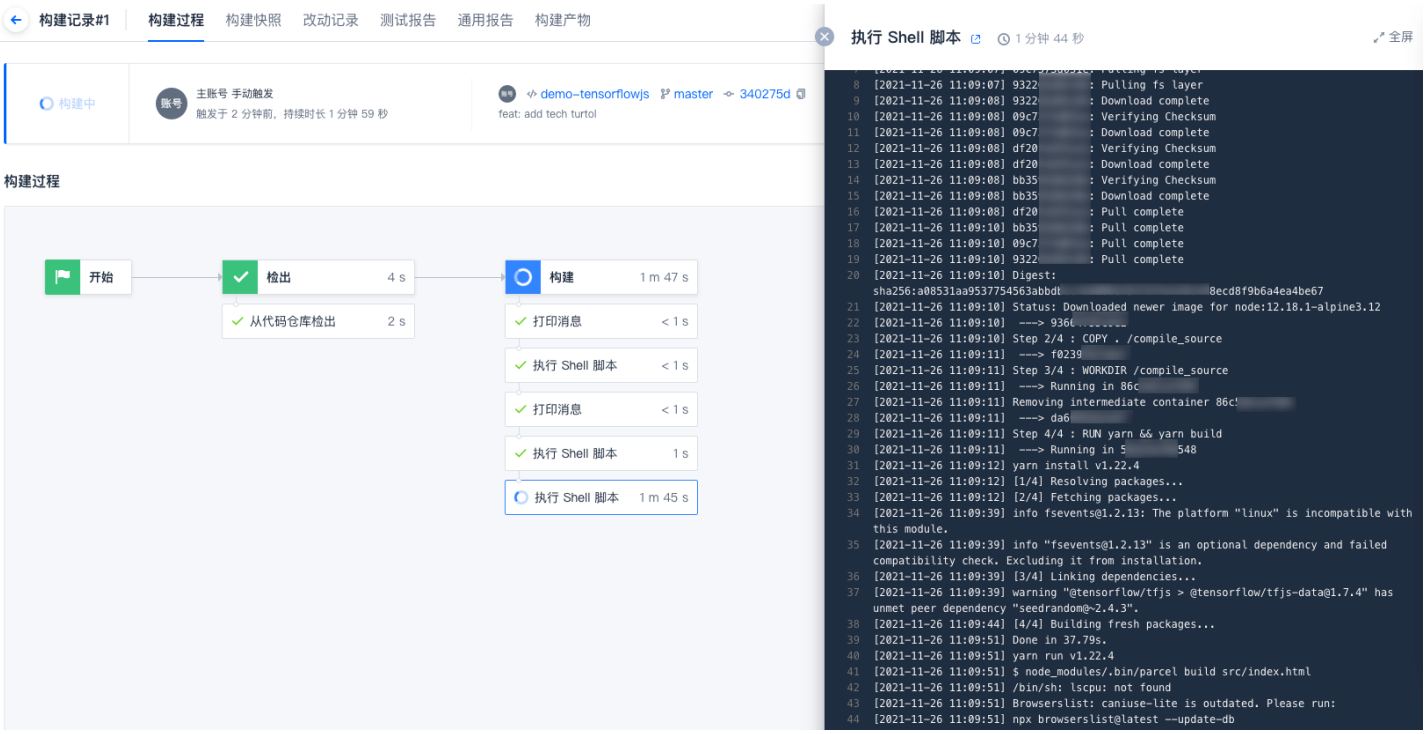
持续集成过程中，我们总会将一些配置（如：账号密码和版本号等）信息以环境变量的形式注入到构建过程中。

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

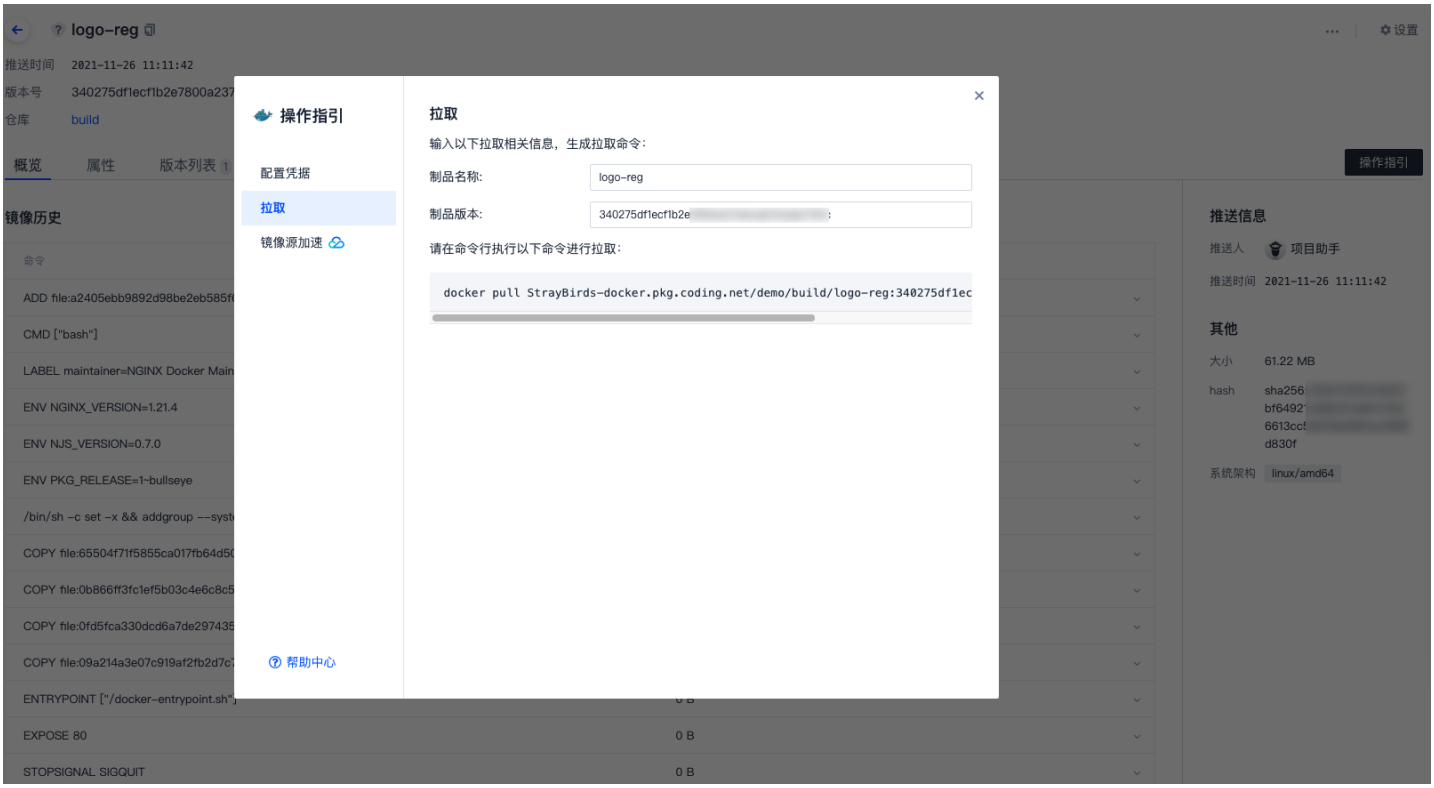
执行构建

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。



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。



运行应用

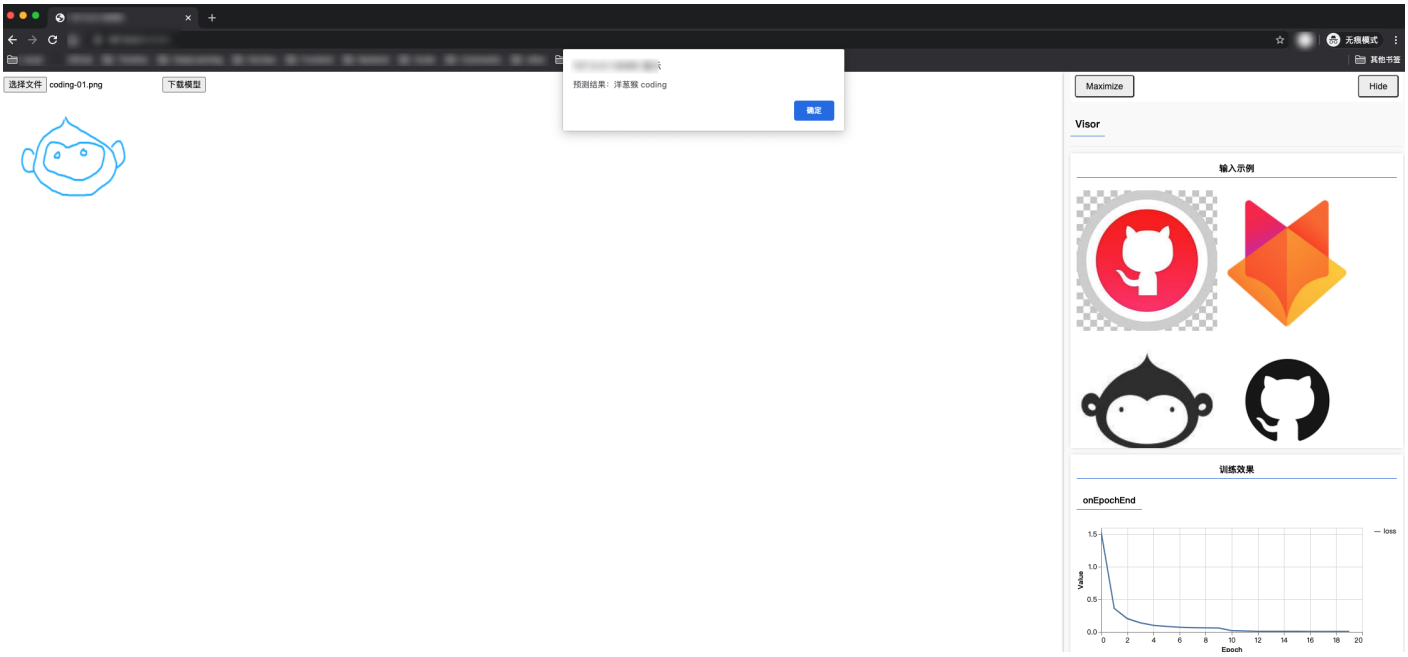
1. 使用以下命令，运行已拉取的制品，即可开始通过机器学习以辨别 CODING、GitHub、GitLab Logo。

```
# 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:340275df1ecf1b2e7800a237ebceb10ceee7161c
```

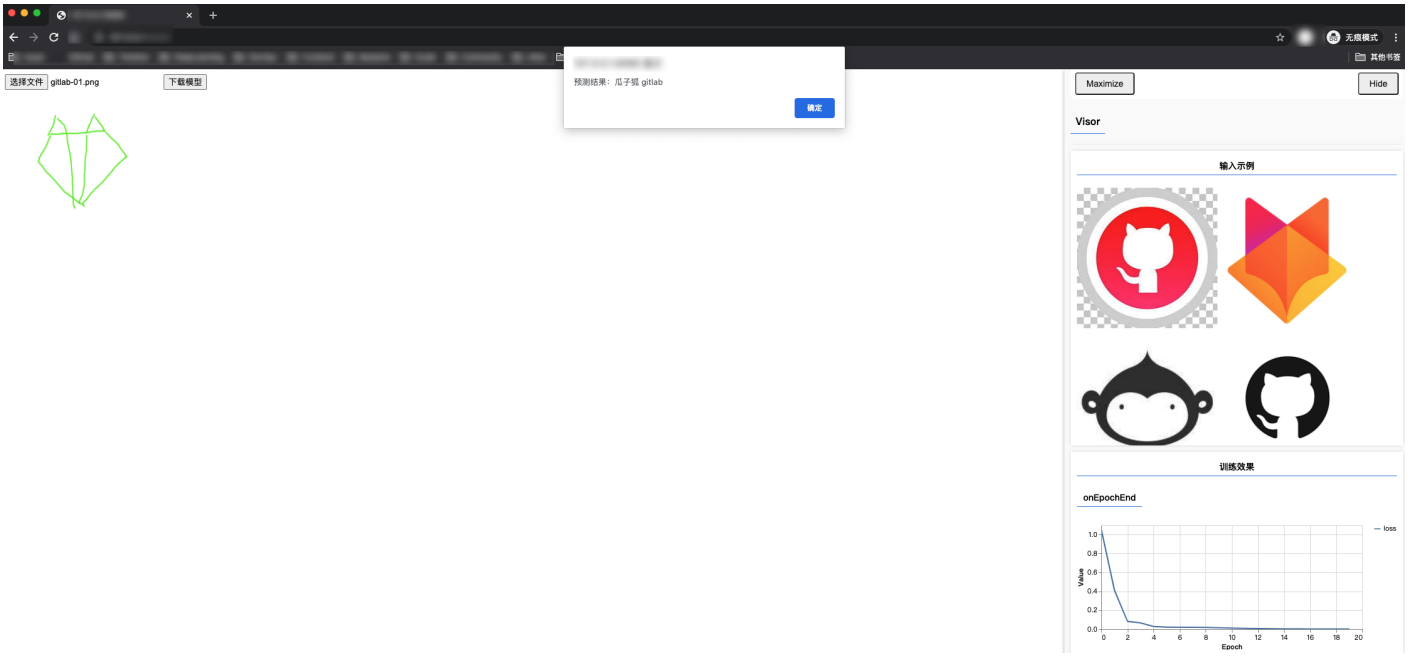
```
/Volumes/CODING-Help/demo-tensorflowjs  master  docker run -p 8080:80 StrayBirds-docker.pkg.coding.net/demo/build/logo-reg:340275df1ecf1b2e7800a237ebceb10ceee7161c
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.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/conf.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
```

2. 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



预测 GitLab logo



3. 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。

自动构建微信小程序

最近更新时间：2022-03-25 15:16:31

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。
整个流程大致如下：

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）。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

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。

进入项目

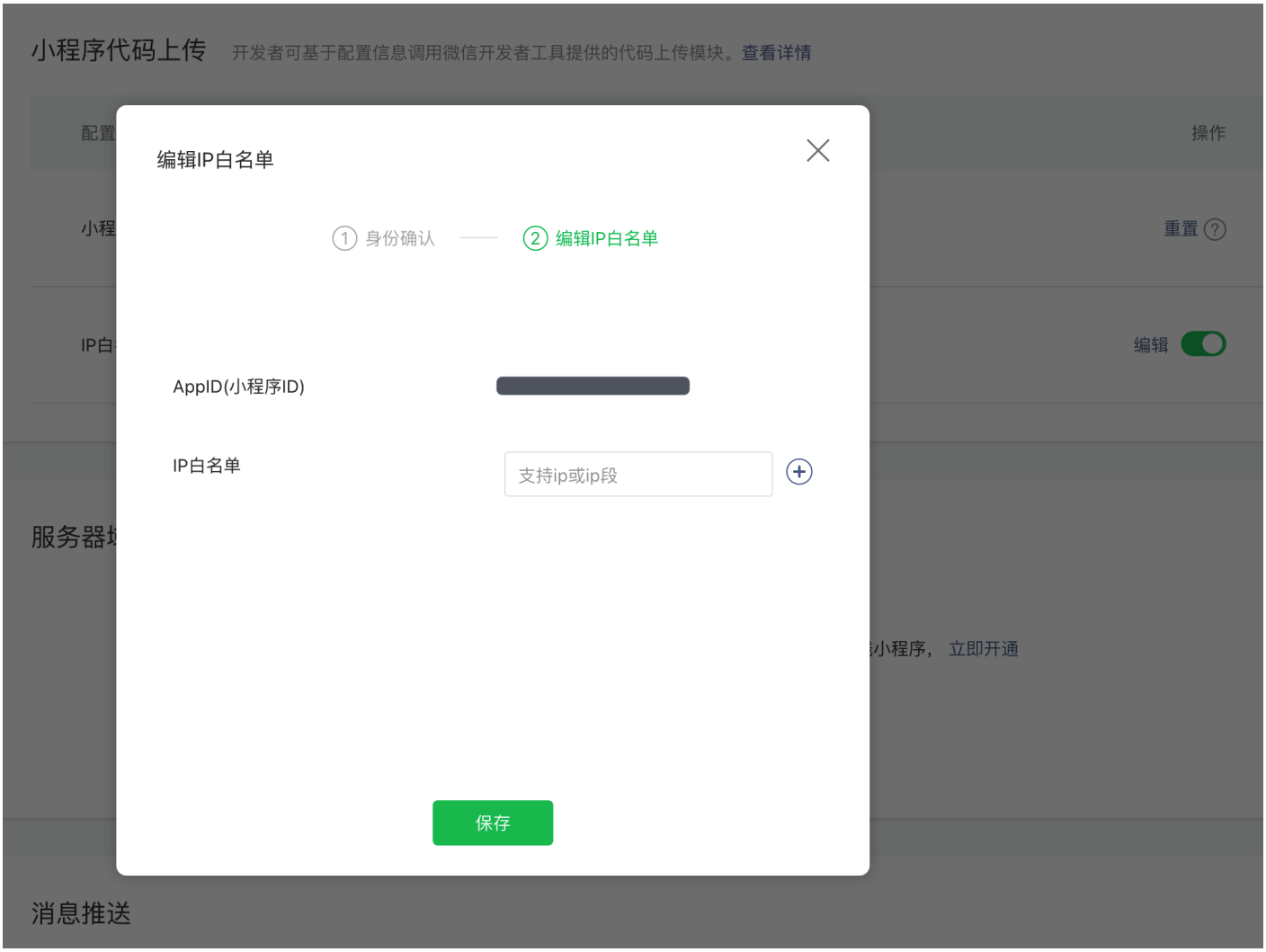
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

设置小程序白名单

1. 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。


```
# 中国上海
111.231.92.100/32,81.68.101.44/32

# 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
```

2. 前往微信小程序的管理后台，点击左侧菜单栏中的**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。



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。

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（如：账号密码和版本号等）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。

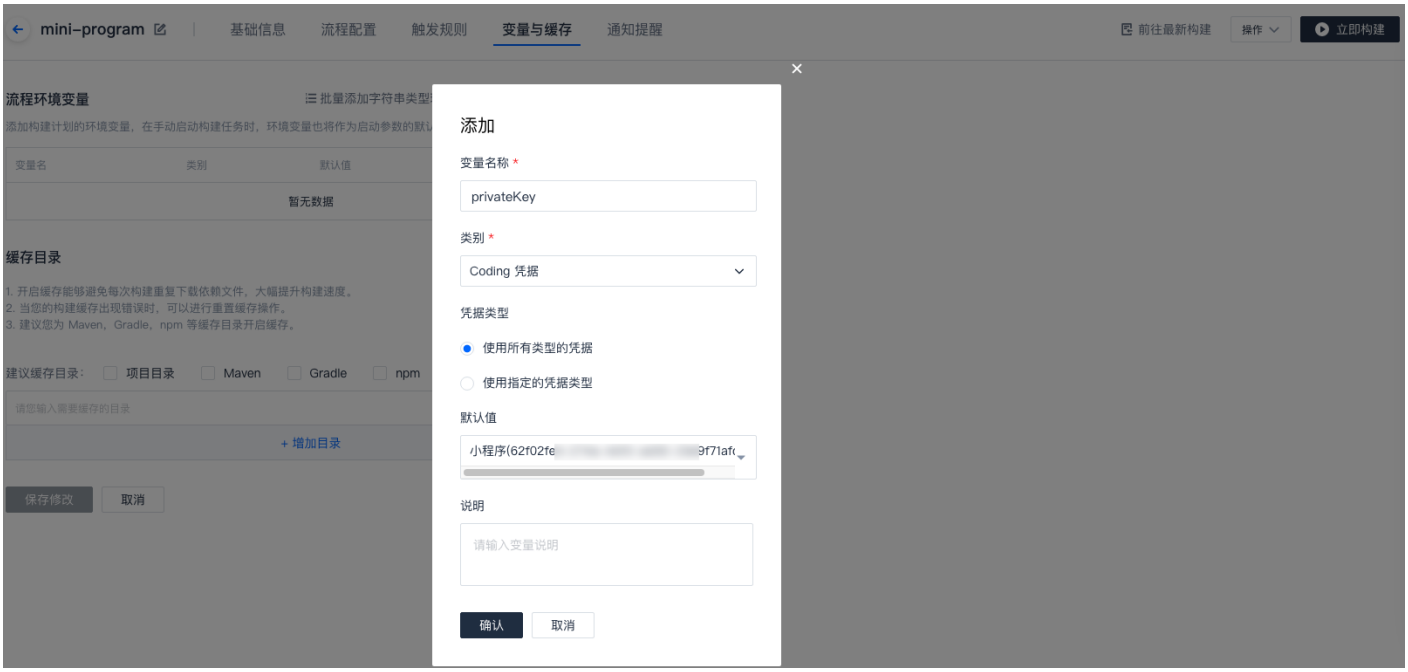
- 微信小程序代码上传私钥
- 企业微信机器人 webhook 地址

微信小程序代码上传私钥

1. 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。

2. 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**。

3. 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或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));
```

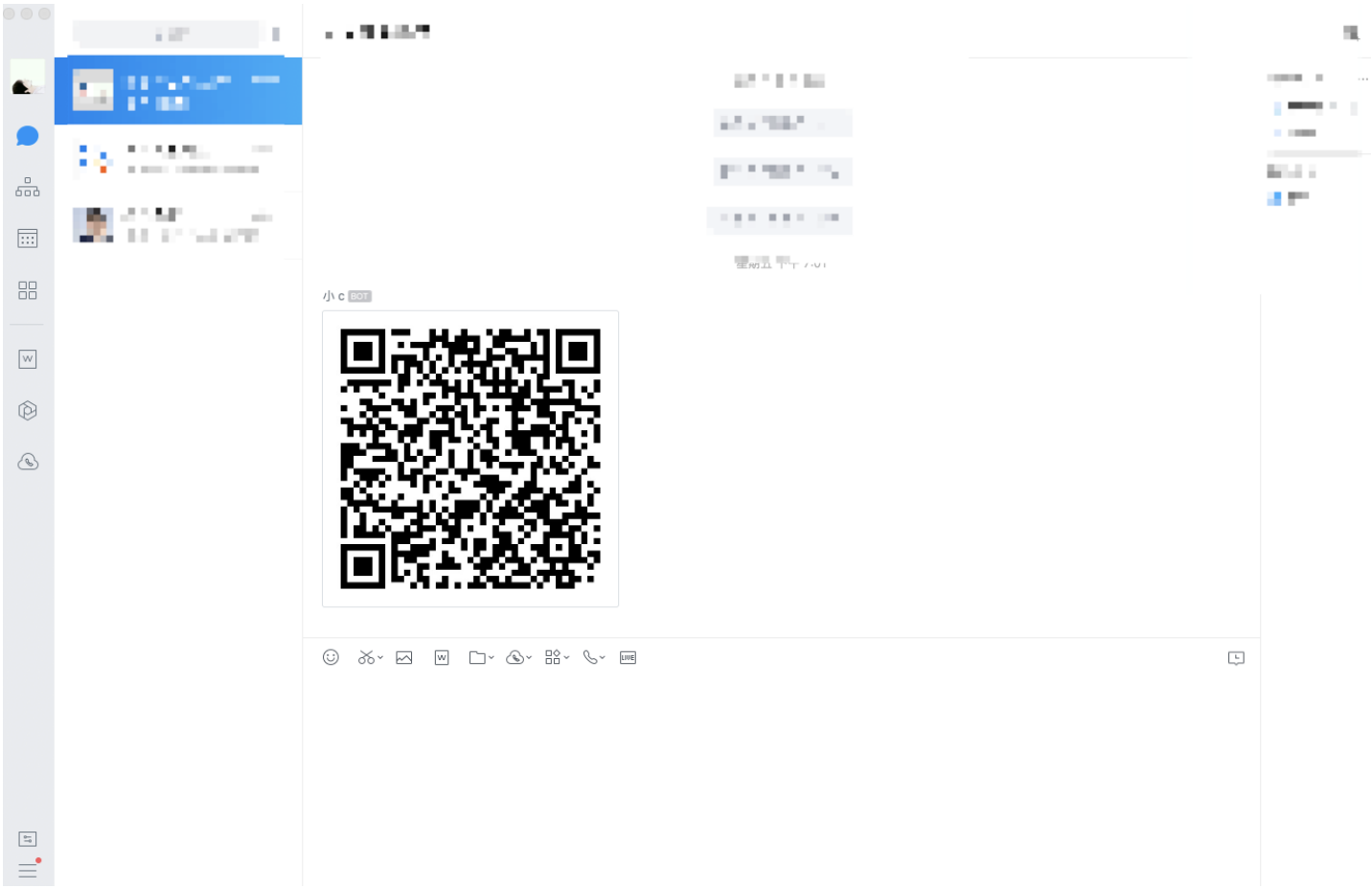
```
const package = require('./package.json')
const appDirectory = fs.realpathSync(process.cwd());
const ProjectConfig = require('./dist/project.config.json');
const previewPath = path.resolve(appDirectory, './preview.jpg');


(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 的可查看这里 企业微信文档。

```
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

版本号和版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。

使用 Flask 构建 Web 应用

最近更新时间：2022-04-13 14:42:12

本文为您介绍如何使用 Flask 构建 Web 应用。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

前置准备

- Git
- Python
- CODING 项目
- Docker 制品仓库
- 示例仓库

  示例项目中的目录结构与各文件的功能解释：

```
python-flask-demo
├── .gitignore
├── Dockerfile
├── Jenkinsfile
├── README.rst
├── app.py
└── requirements.txt
0 directories, 5 files
```

参数	介绍
gitignore	可以在内声明一些不需要加入到 Git 管理的文件或文件夹，这样就不用每次提交代码的时候再特地处理这些文件了。
Dockerfile	Dockerfile 就是我们用来构建 Docker 镜像的源码，Docker 可以读取里面的指令来自动构建镜像。
Jenkinsfile	Jenkinsfile 是一个文本文件，它定义了 Jenkins 流水线，将作为配置文件检入至源代码控制仓库中。
README.rst	由开发者写给其他人阅读的一个文本文件，描述了这是个什么样的项目，有什么用，如何安装和使用等。
app.py	包含 main 函数的主要程序代码文件。
requirements.txt	requirements.txt 可以保证项目依赖包版本的确定性, 不会因为依赖更新而导致异常产生，通常可以由 pip 包管理工具生成。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

初始化项目

1. 将示例项目导入至 CODING 代码仓库中，在新建代码仓库时选择**导入外部仓库**，填入开源仓库地址。

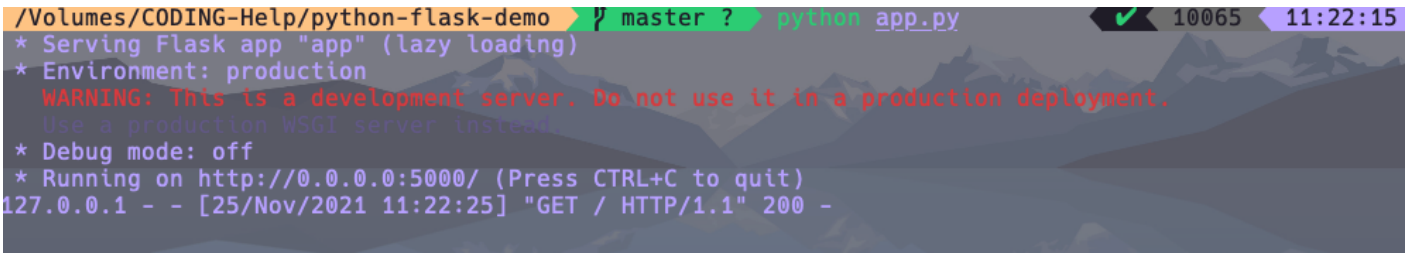
2. 将项目拉取至本地后，输入以下命令安装依赖：

```
pip install -r requirements.txt
```

3. 在本地启用预览：

```
python app.py
```

4. 当终端返回以下数据后，意味着项目已构建成功。



5. 在浏览器中输入 http://127.0.0.1:5000/ 进行预览。

Hello World!

编译与构建

编译（Compiling）就是将源代码文件通过编译器转换为目标文件的过程，是构建（Building）的一个部分，构建还包括连接与测试环节。

为什么需要编译/构建?

我们开发时所写的源代码，技术人员可以理解，机器并不理解其意。而若想在计算机上运行一个应用程序，则必须将该程序由源代码转换成计算机所能理解的二进制机器码。为此，我们需要将程序编译成可执行文件，通过构建来执行更加工程化、结构化、严谨的一系列步骤，并将产出的 制品 部署上生产环境。

在将我们开发的程序给他人使用时，他人的系统中可能并没有安装 Python 环境或没有安装第三方库，这个时候如果要求对方去装环境，会特别不方便，特别是当对方不是技术人员的时候。这个时候如果我们将开发好的程序编译打包成一个可执行文件，那么一切就会变得简单起来。

如何编译/构建?

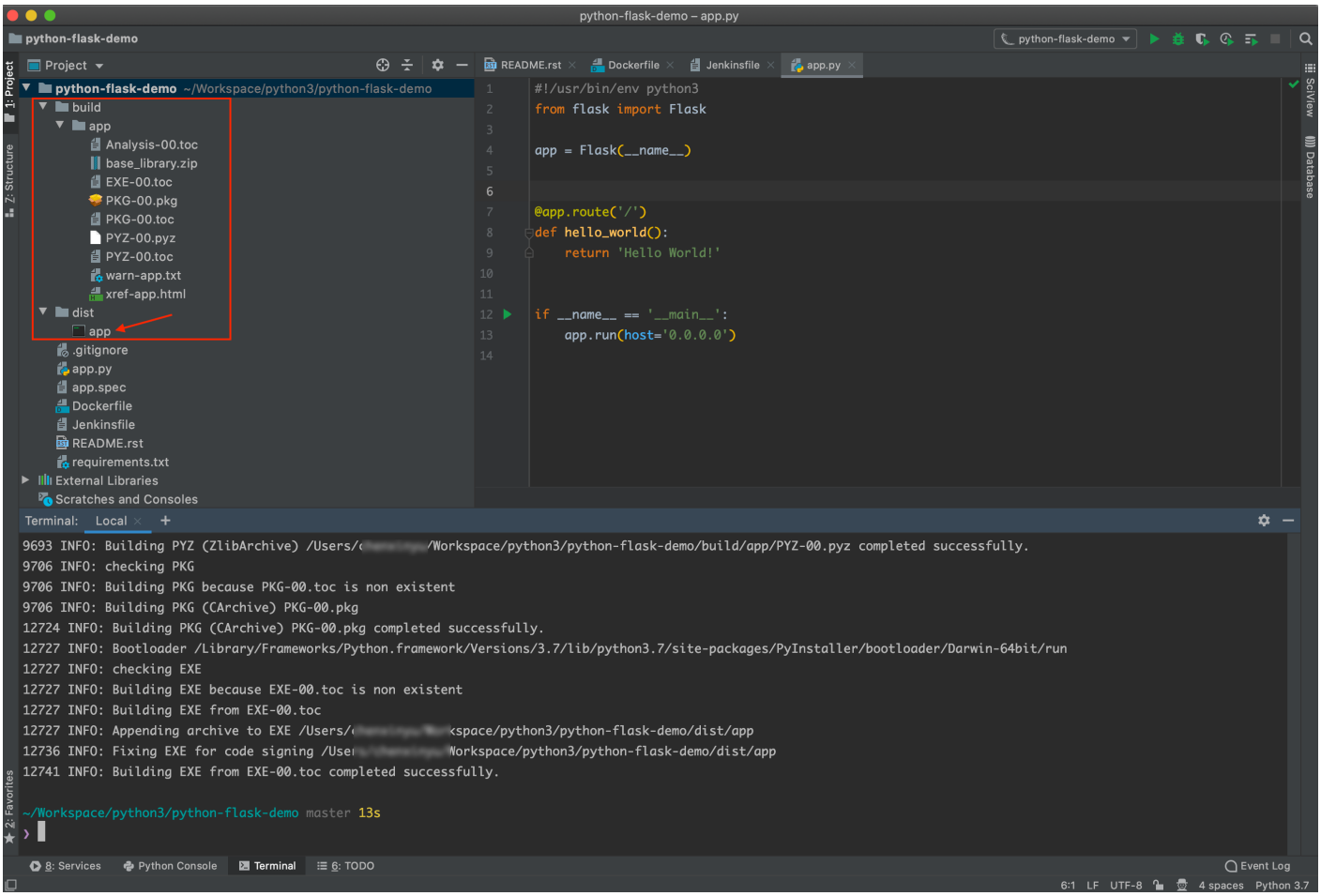
在进行编程操作的时候，我们常常会遇到很多与编程无关的项目管理工作，例如下载依赖、编译源码、单元测试、项目部署等操作。一般的，小型项目我们可以手动实现这些操作，然而大型项目这些工作则相对复杂。构建工具是帮助我们实现一系列项目管理、测试和部署操作的工具。同时，它能大大提升我们的效率。

本文所使用的项目为 Python 项目，我们将通过 PyInstaller 工具进行编译打包。PyInstaller 自身支持跨平台，并且自身基础操作十分简单、易上手。您可以在工具官网进行下载，macOS 用户可以直接使用 brew install PyInstaller 命令直接安装。

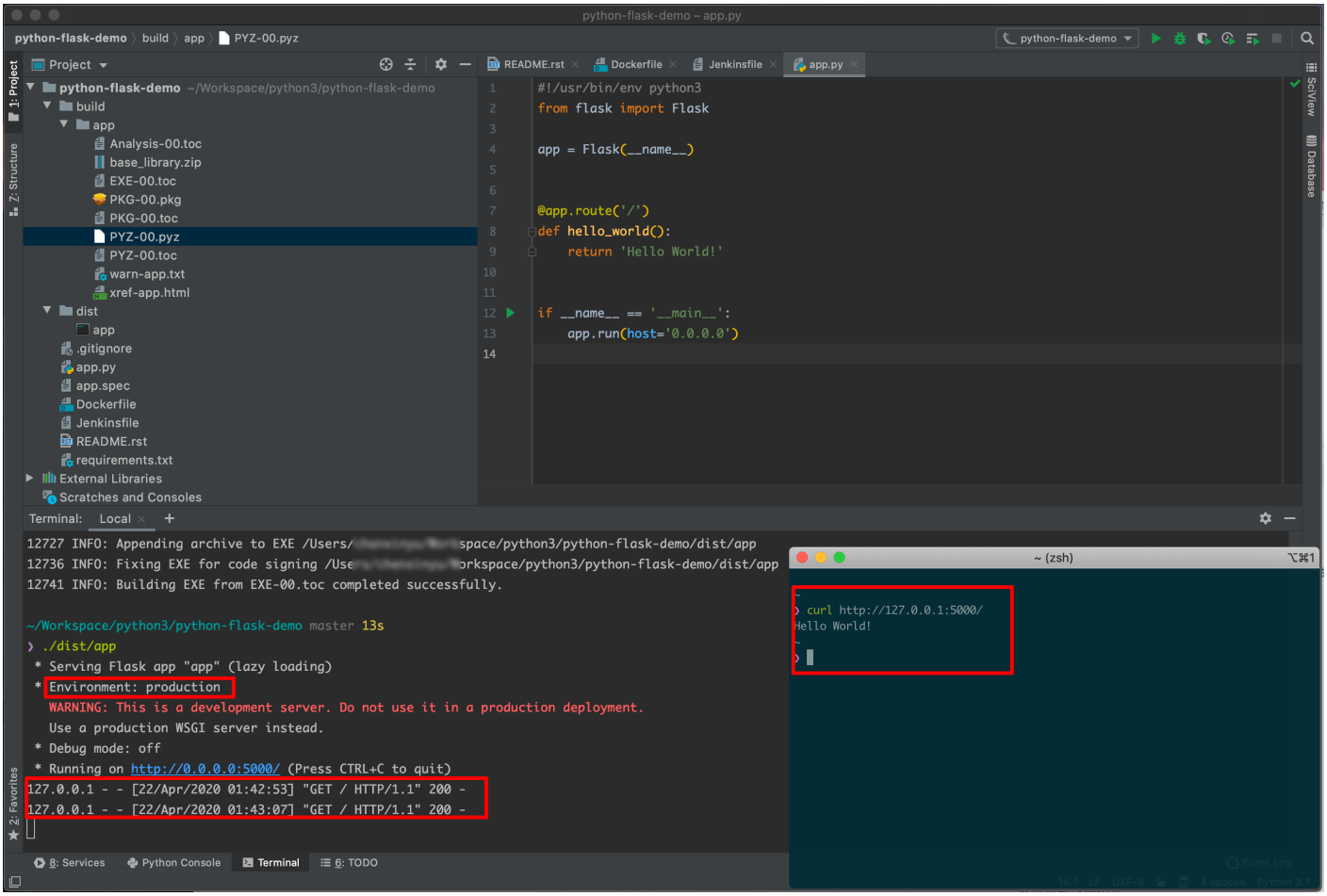
1. **安装完成后，进入项目的址，因只有一个 app.py 源代码文件，所以这里使用 -F Flag 命令编译生成单个可执行文件。**

```
pyinstaller -F app.py
```

2. **运行命令，在产生了许多编译日志之后，在项目下生成了两个新目录：build 和 dist。**



3. **这里我们可以看到 dist 目录下生成了一个可执行程序 app。这就是我们这次编译所得到的主要制品了。打开此制品后，即可在5000端口运行相同的服务。**



为什么需要持续集成?

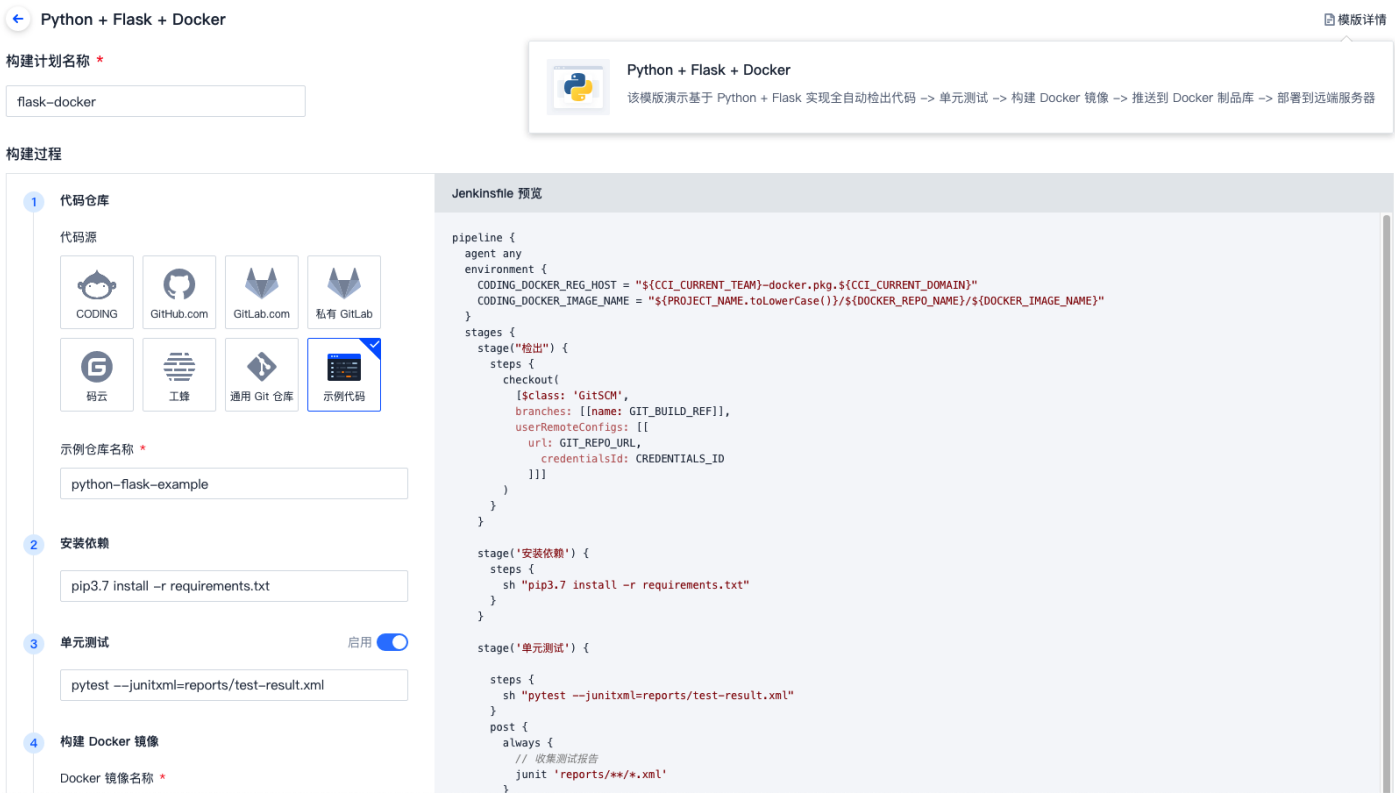
在上述操作中，存在着一些完全可以抽离出来自动化的步骤。例如我们每次写完代码，在提交前要进行编译、单元测试，而这些步骤的操作可能大致相同，那么我们是不是可以把这些步骤交给某个自动化的工具来完成呢？甚至您可能已经写好了编译、测试的脚本，只需要运行一下即可，那么我们甚至可以把这个运行脚本的步骤也抽离出去。

通过持续集成，能够实现在代码自动提交至远端后，自动进入编译、测试、发布流程。如果在此期间发现代码有问题，CODING 持续集成还能够通过多种进行形式通知，及时暴露错误并加以改正，而不是等合进 master 分支后才发现问题。

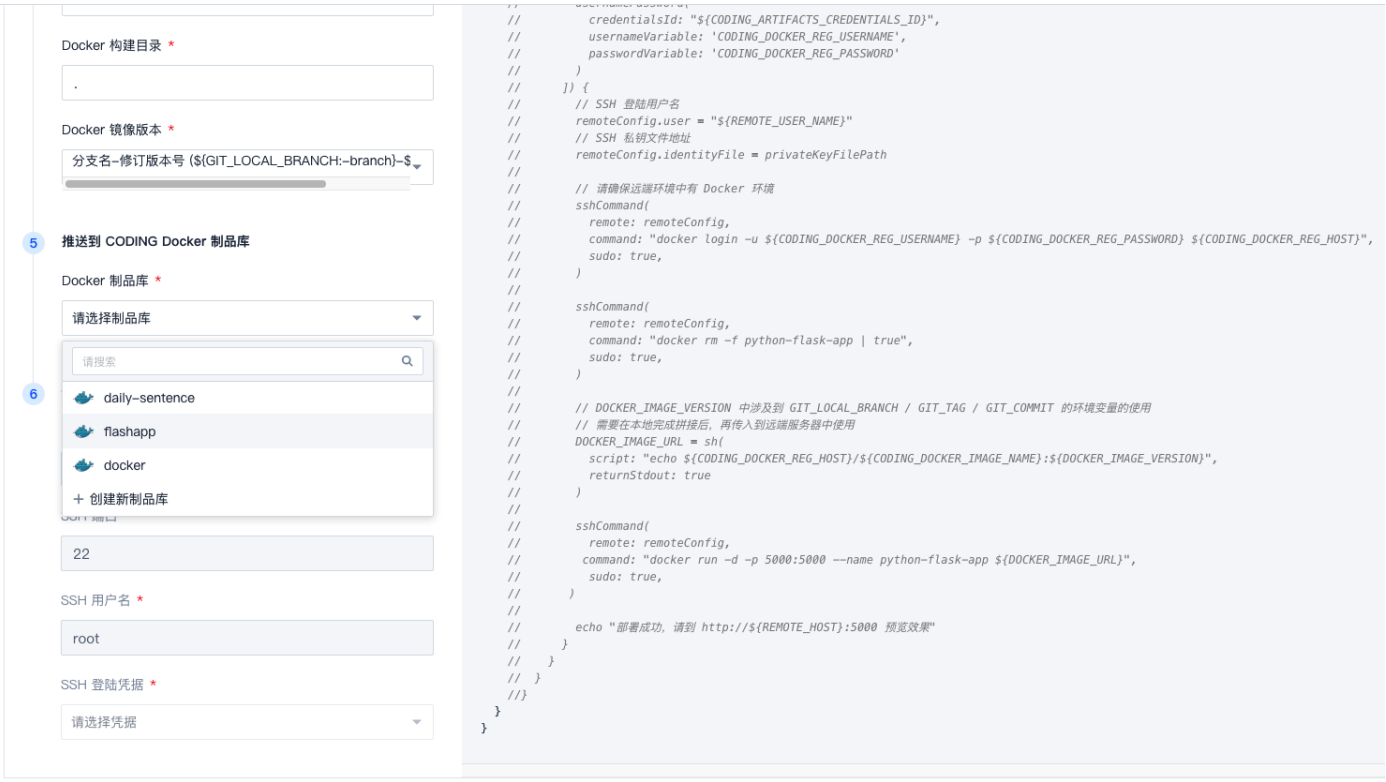
接下来我将演示如何使用持续集成将上述项目进行编译与测试。

创建构建计划

1. 进入项目后，单击进入左侧的**持续集成**功能，选择 **Python+Flask+Docker** 模板。



2. 选择需要导出目标制品库。

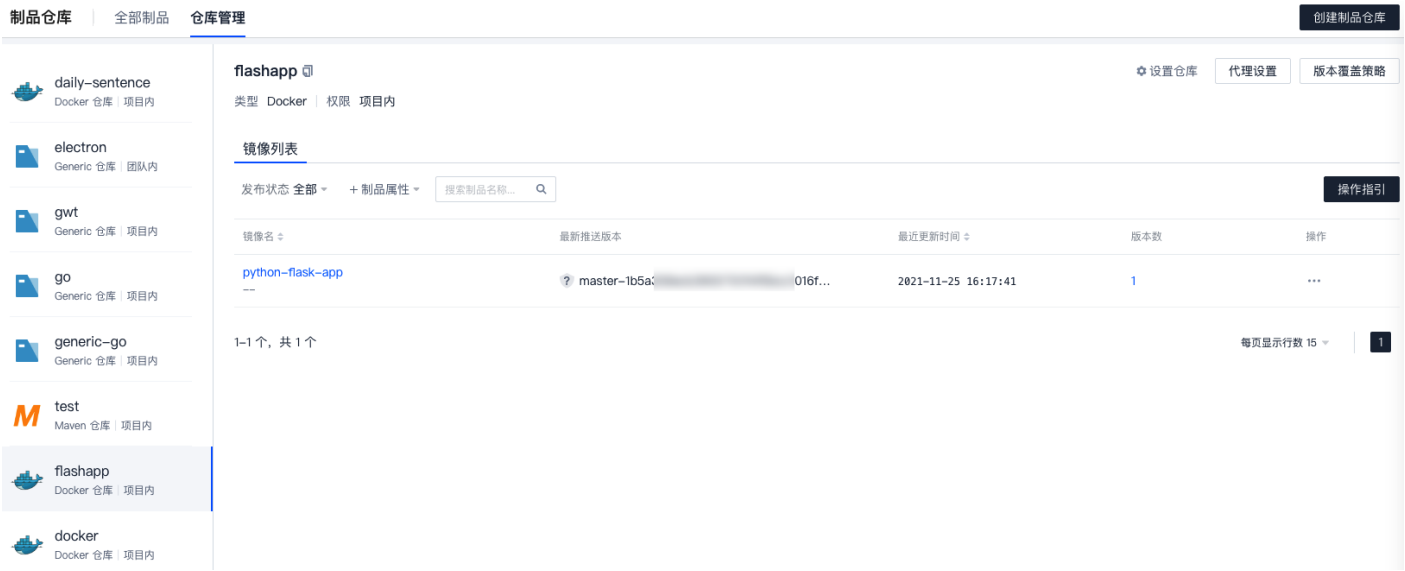


3. 您还可以根据自身情况，修改持续集成的**触发规则**、**变量与缓存**以及**通知提醒**等。在**触发规则**中还可以配置自动监听代码分支，自动触发持续集成。

开始构建

1. 我们返回至**持续集成** > **构建**页面，此时可以看到我们刚新建的**构建计划**。触发后可以在**构建过程**中查看整体的构建过程与状态。

2. 在制品仓库中可以看到已上传的镜像。



结语

通过持续集成和制品仓库的搭配使用，能够轻松地将应用打包发布，用以加速开发流程。

将 Ruby 项目发布至腾讯云 TKE

最近更新时间：2022-04-11 15:29:24

本文将使用 CODING DevOps 与腾讯云容器服务（下称 TKE）实践 Ruby 和 Sinatra 应用开发。本文涵盖单元测试、构建 Docker 镜像、推送到制品库、部署到 K8s 集群等环节，讲述如何实现项目的自动化构建、测试、发布、部署。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

前置准备

- 示例仓库
- CODING 项目
- Docker 制品仓库
- 腾讯云 TKE 集群

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

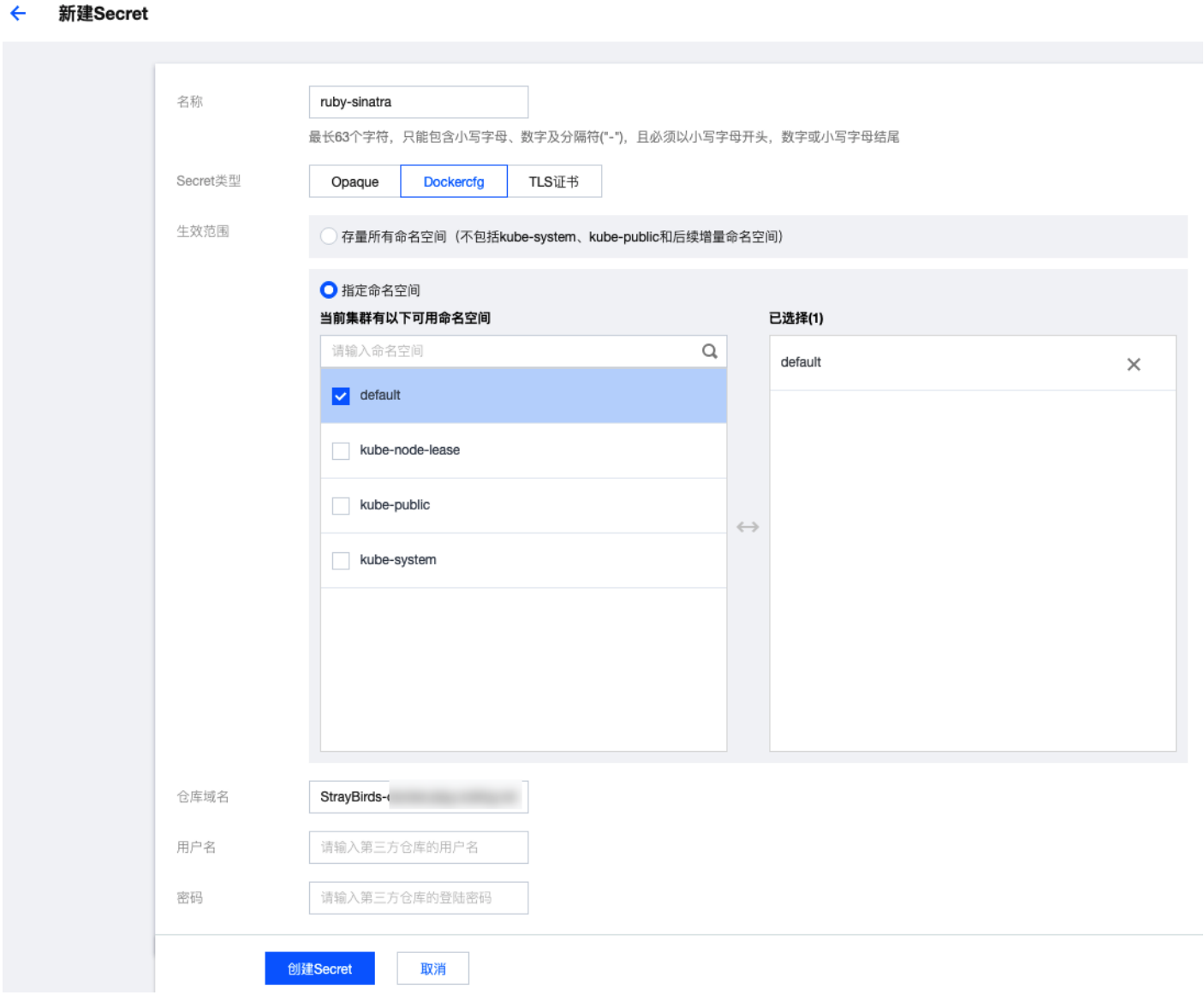
将制品库凭证录入至 TKE

1. 因 K8s 集群从制品库拖取镜像需要访问凭证，需要将 CODING 项目中的 Docker 制品仓库的配置凭据导入至 TKE。输入密码后复制系统自动生成的访问令牌：

2. **设置凭证中包含三重信息：**

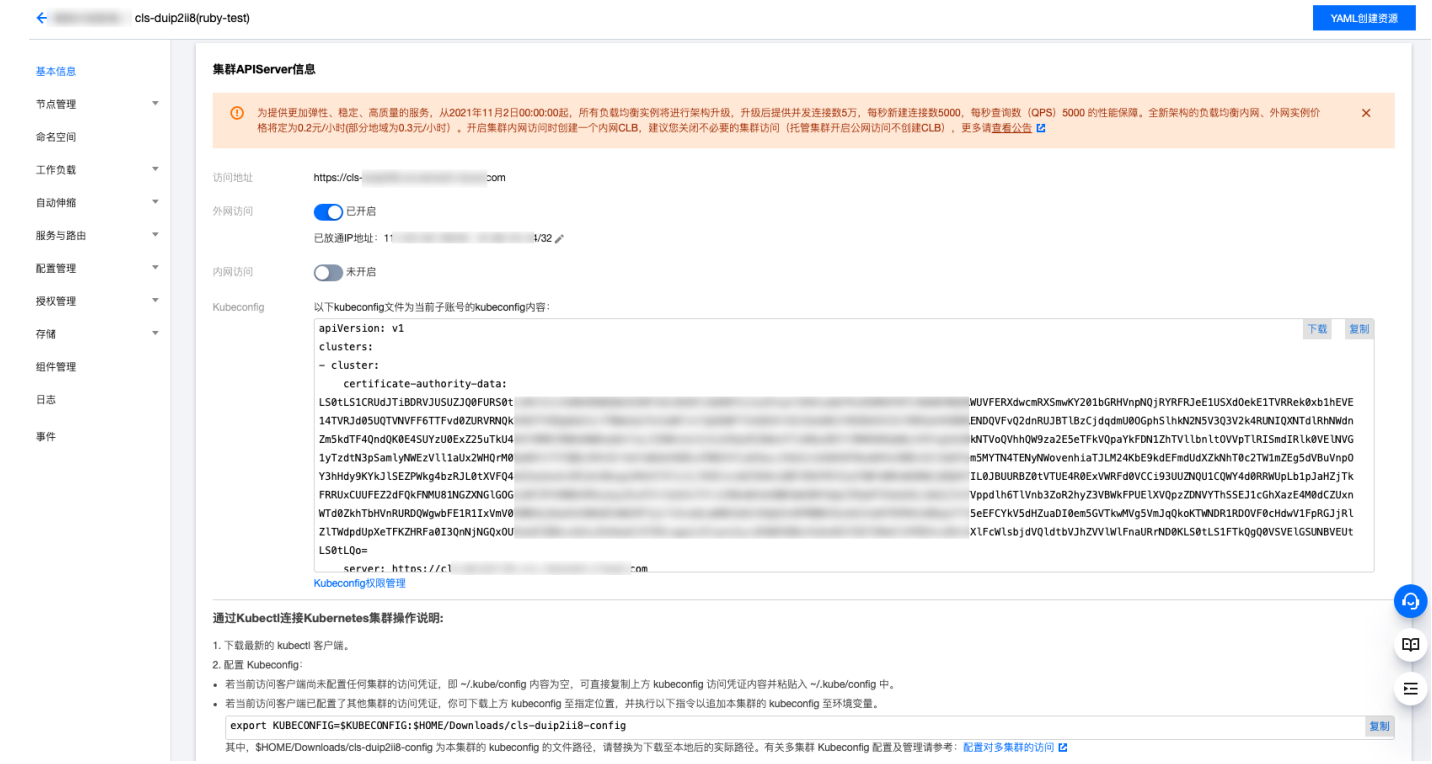
- **库域名：**StrayBirds-docker.pkg.coding.net
- **用户名：**ruby****
- **密码：**1261*****

3. 在 TKE 集群中单击**集群** > **配置管理** > **Secret**，将制品仓库凭据进行录入。

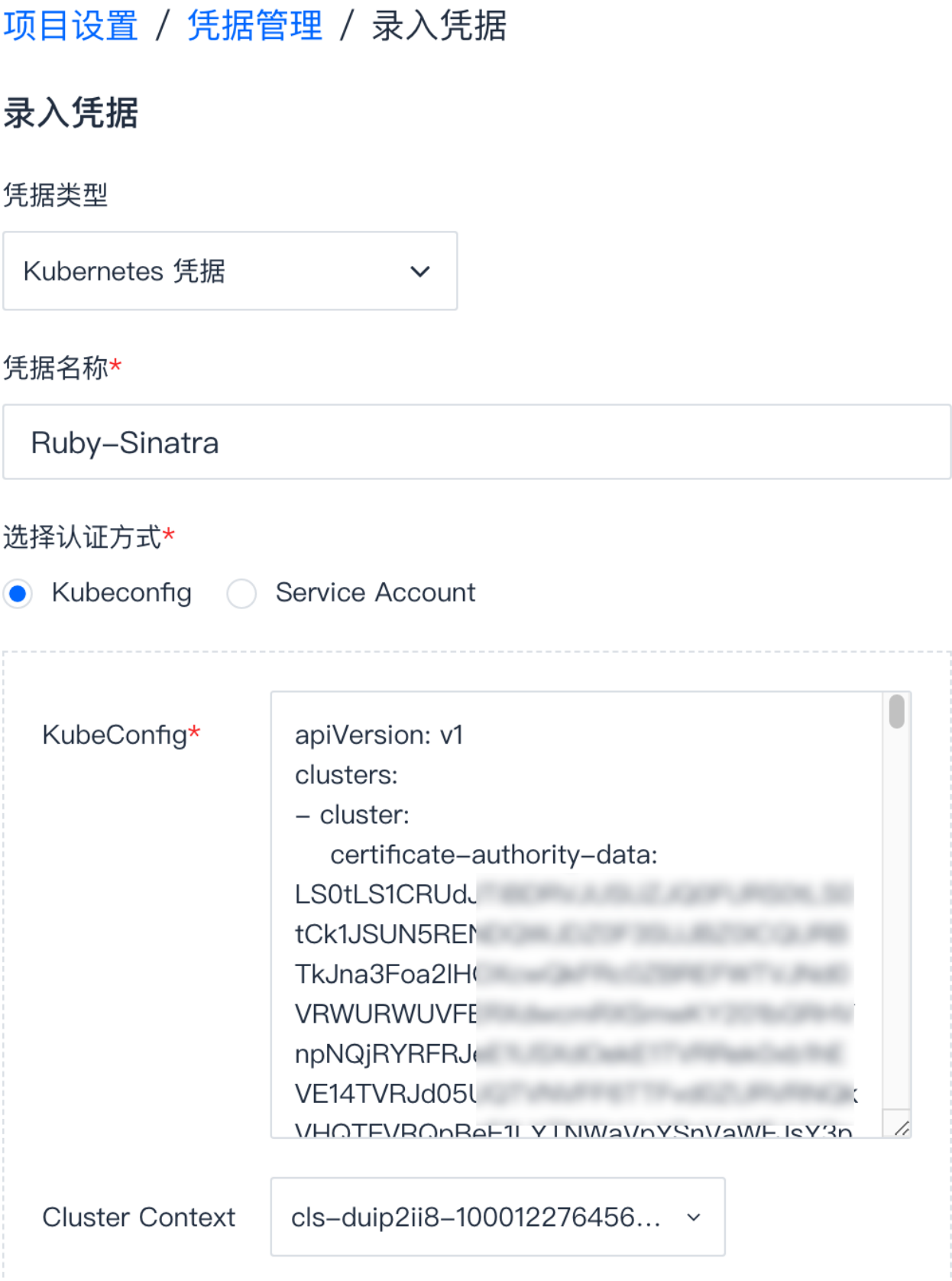


录入集群凭据

1. 前往 TKE 控制台中的基本信息，复制集群 APIServer 信息中的 Kubeconfig。需勾选**外网访问**，录入构建机的出口 IP：111.231.92.100/32,81.68.101.44/32。


cls-duip2ii8(ruby-test)
YAML创建资源
基本信息
节点管理
命名空间
工作负载
自动伸缩
服务与路由
配置管理
授权管理
存储
组件管理
日志
事件
集群APIServer信息
为提供更加弹性、稳定、高质量的服务，从2021年11月2日00:00:00起，所有负载均衡实例将进行架构升级，升级后提供并发连接数5万，每秒新建连接数5000，每秒查询数（QPS）5000 的性能保障。全新架构的负载均衡内网、外网实例价格将定为0.2元/小时(部分地域为0.3元/小时）。开启集群内网访问时创建一个内网CLB，建议您关闭不必要的集群访问（托管集群开启公网访问不创建CLB），更多请查看公告
访问地址
https://cls-com
外网访问
已开启
已放通IP地址：1
4/32
内网访问
未开启
Kubeconfig
以下kubeconfig文件为当前子账号的kubeconfig内容：
下载
复制
apiVersion: v1
clusters:
- cluster:
certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0t
WUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJeE1USXdOekE1TVRRek0xb1hEVE
14TVRJd05UQTVNVFF6TTFvd0ZURVRNQk
ENDQVFvQ2dnRUJBTlBzCjdqdmU0OGphSlhkN2N5V3Q3V2k4RUNIQXNTdlRhNWdn
Zm5kdTF4QndQK0E4SUYzU0ExZ25uTkU4
kNTVoQVhhQW9za2E5eTFkVQpaYkFDN1ZhTVllbnltOVVpTlRISmdIRlk0VElVNVG
1yTzdtN3pSamlyNWEzVll1aUx2WHQrM0
m5MYTN4TENyNWovenhiaTJLM24KbE9kdEFmdUdXZkNhT0c2TW1mZEg5dVBuVnpO
Y3hHdy9KYkJlSEZPWkg4bzRJL0tXVFQ4
IL0JBUURBZ0tVTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTk
FRRUxCUUFEZ2dFQkFNMU81NGZXNGlGOG
Vppdlh6TlVnb3ZoR2hyZ3VBWkFPUElXVQpzZDNVYThSSEJ1cGhXazE4M0dCZUxn
WTd0ZkhTbHVnRURDQWgwbFE1R1IxVmV0
5eEFCYkV5dHZuaDI0em5GVTkwMVg5VmJqQkoKTWNDR1RDOVF0cHdwV1FpRGJjRl
ZlTWdpdUpXeTFKZHRFa0I3QnNjNGQxOU
XlFcWlsbjdVQldtbVJhZVVlWlFnaURrND0KLS0tLS1FTkQgQ0VSVElGSUNBVEUt
LS0tLQo=
server: https://cl com
Kubeconfig权限管理
通过Kubectl连接Kubernetes集群操作说明:
1. 下载最新的 kubectl 客户端。
2. 配置 Kubeconfig：
若当前访问客户端尚未配置任何集群的访问凭证，即 ~/.kube/config 内容为空，可直接复制上方 kubeconfig 访问凭证内容并粘贴入 ~/.kube/config 中。
若当前访问客户端已配置了其他集群的访问凭证，你可下载上方 kubeconfig 至指定位置，并执行以下指令以追加本集群的 kubeconfig 至环境变量。
export KUBECONFIG=$KUBECONFIG:$HOME/Downloads/cls-duip2ii8-config
复制
其中，$HOME/Downloads/cls-duip2ii8-config 为本集群的 kubeconfig 的文件路径，请替换为下载至本地后的实际路径。有关多集群 Kubeconfig 配置及管理请参考：配置对多集群的访问

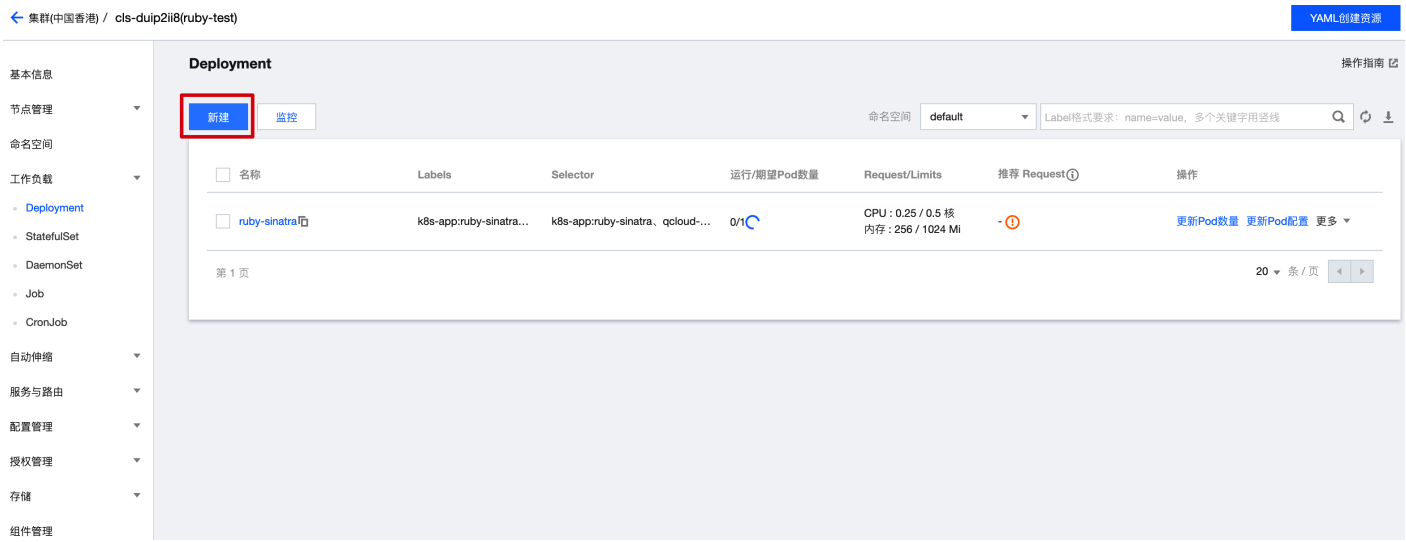
2. 复制后将其录入至**项目设置** > **开发者选项** > **凭据管理**中。凭据类型选择 **Kubernetes 凭据**并勾选所有持续集成计划授权。



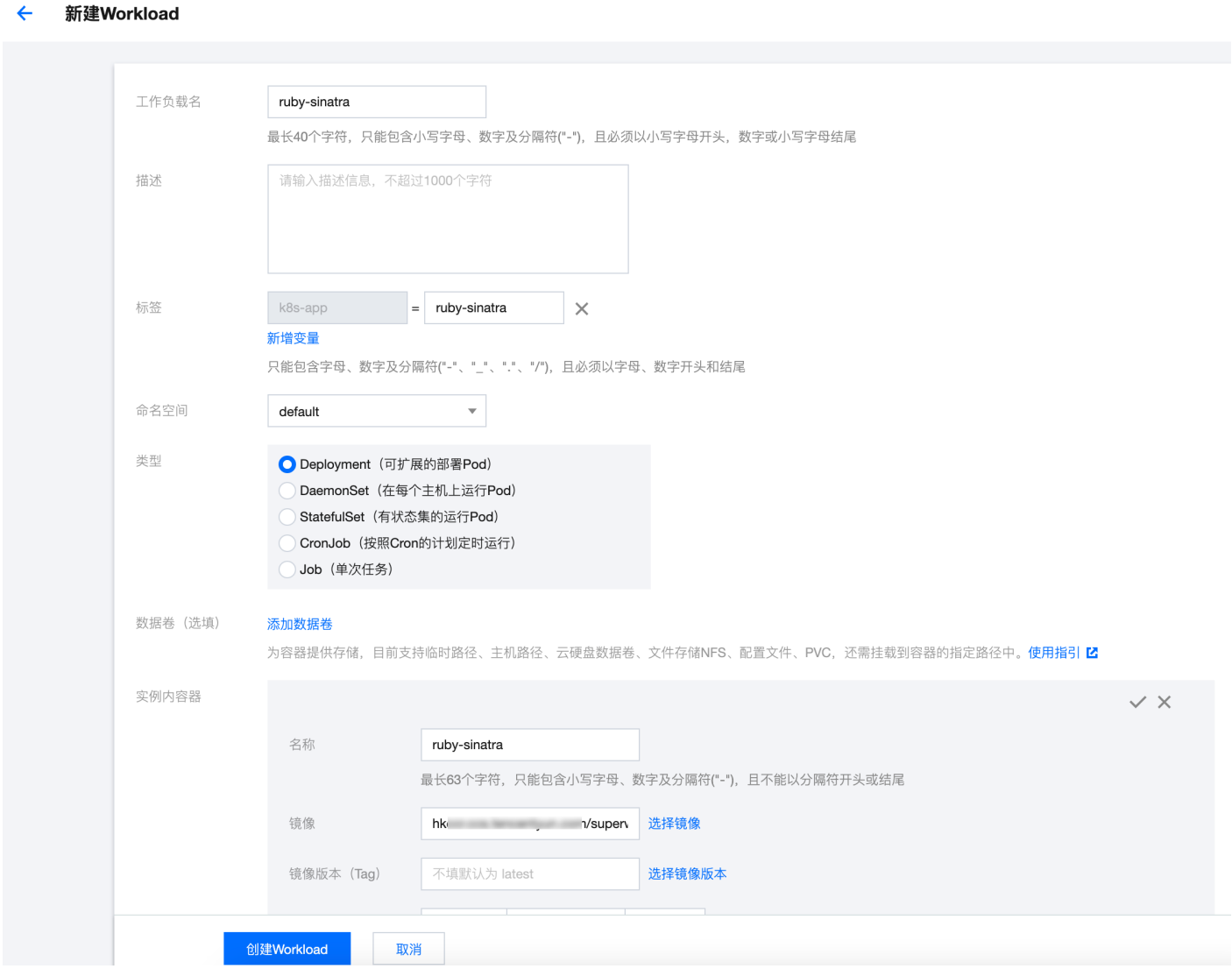
在集群中创建 Deployment

为方便将应用部署 K8s 集群中，需要先创建 Deployment 工作负载，之后就可以通过持续集成任务反复将新构建的应用镜像部署到 K8s 集群上。

1. 单击**工作负载 > Deployment** 中的**新建**。



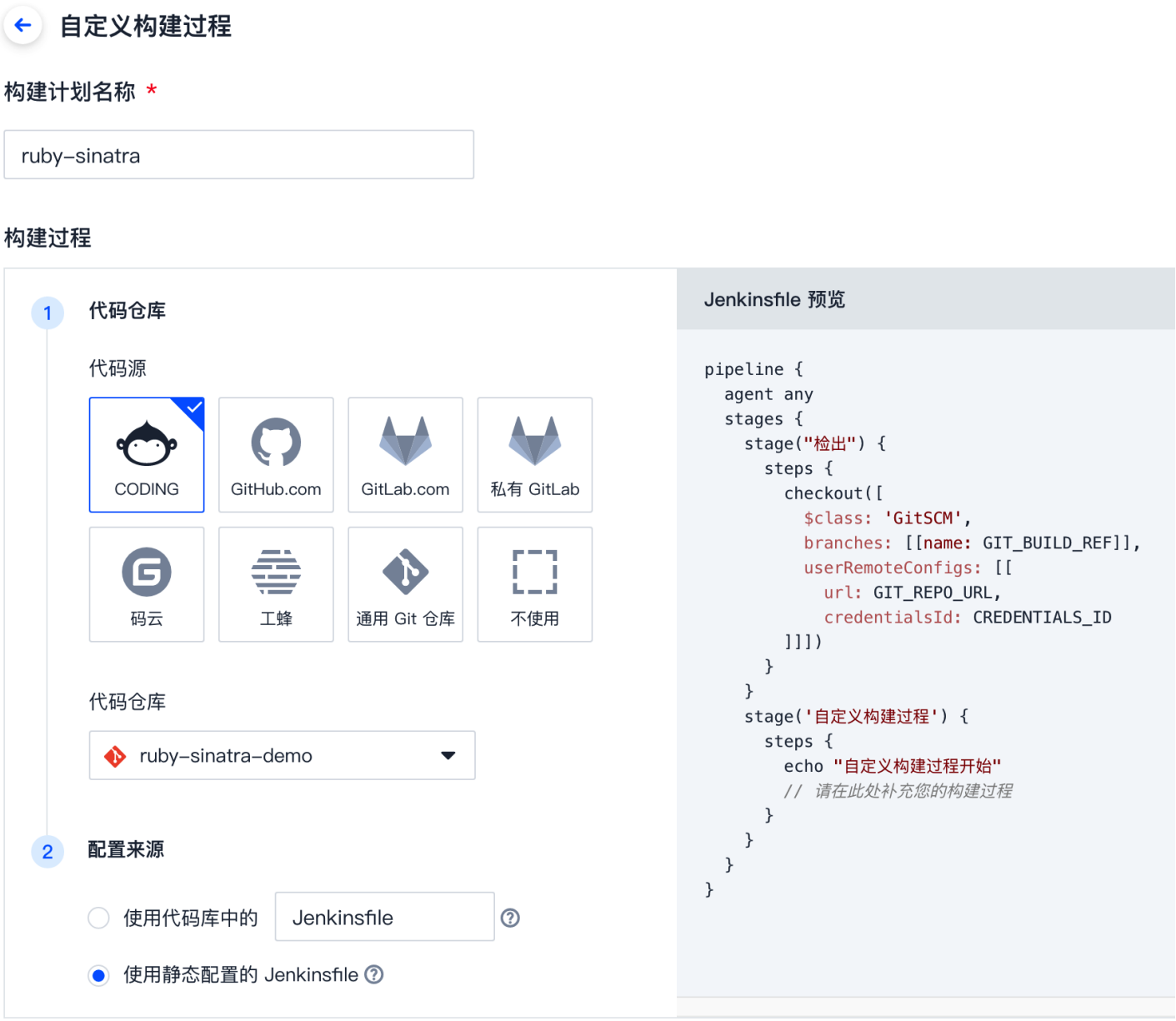
2. 由于现在还没有构建物镜像，此处选用 nginx r 镜像进行代替，以确保 Deployment 正常运行。



3. 在访问设置中填写所需开放的容器端口，确认无误后完成 Workload 创建。

配置构建计划

单击项目左侧菜单栏中的**持续集成**功能，单击右上角的**创建构建计划**，选择**自定义构建过程**模板。



Ruby 和 Sinatra 应用持续集成过程分成5个阶段：

1. 开始阶段
2. 运行单元测试
3. 构建 Docker 镜像
4. 推送到制品库
5. 部署到 K8s 集群

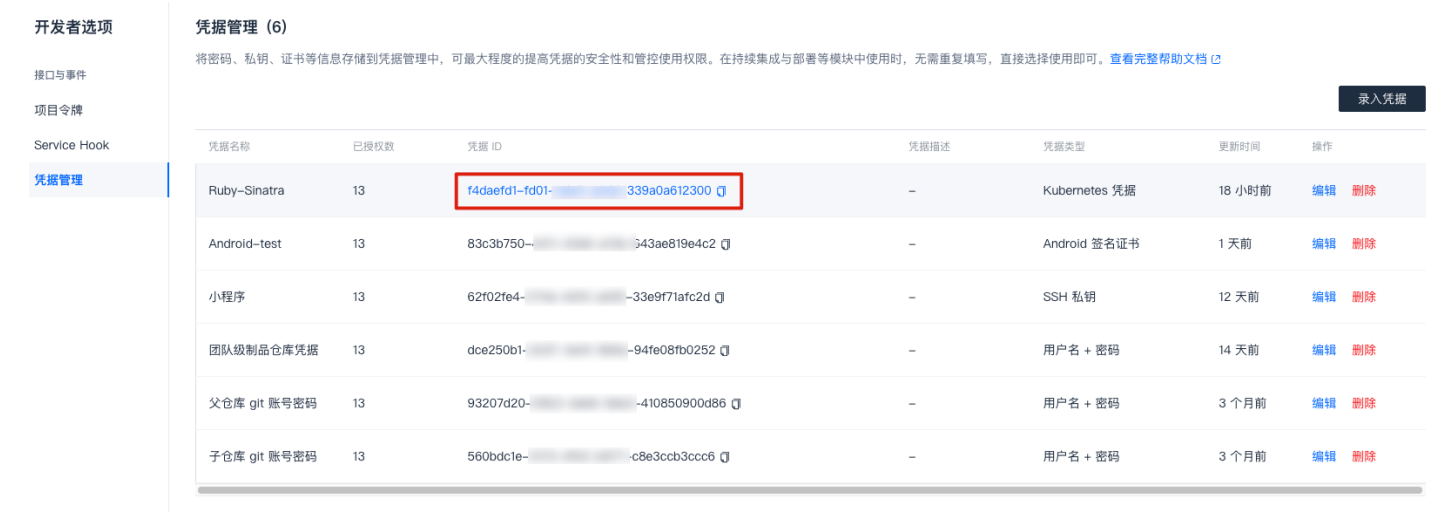
步骤1：开始阶段

1. 此阶段是默认生成的，几乎所有持续集成的构建计划都会包含此阶段。在这个阶段中可以配置一些全局使用的参数，例如构建基础环境、环境变量等。本实践需要在此处添加3个 环境变量，以便被后续阶段所引用。

参数	介绍
CODING_DOCKER_REG_HOST	制品库主机，用于登录制品库。
DOCKER_IMAGE_NAME	Docker 镜像名称，用于构建和推送镜像。
TKE_CLUSTER_CREDENTIAL_ID	TKE 集群凭据 ID。

2. 您可以参照下图进行命令填写或替换：

3. TKE 的集群凭据 ID 填写在上文中录入的 Kubernetes 凭据。



步骤2：配置单元测试阶段

检出阶段不作修改。在图形化编辑器中单击**蓝色+号**新建阶段，命名为单元测试。在步骤中添加**执行 Pipeline 脚本**，填写下文中的示例测试代码：

```
docker.image('ruby:2.6').withRun('-v $(pwd):/app -w /app') { c ->
sh 'bundle install'
sh 'rake test'
}
```

步骤3：构建 Docker 镜像阶段

新建阶段并命名为构建 Docker 镜像。在步骤中添加**执行 Pipeline 脚本**，填写下文中的示例代码：

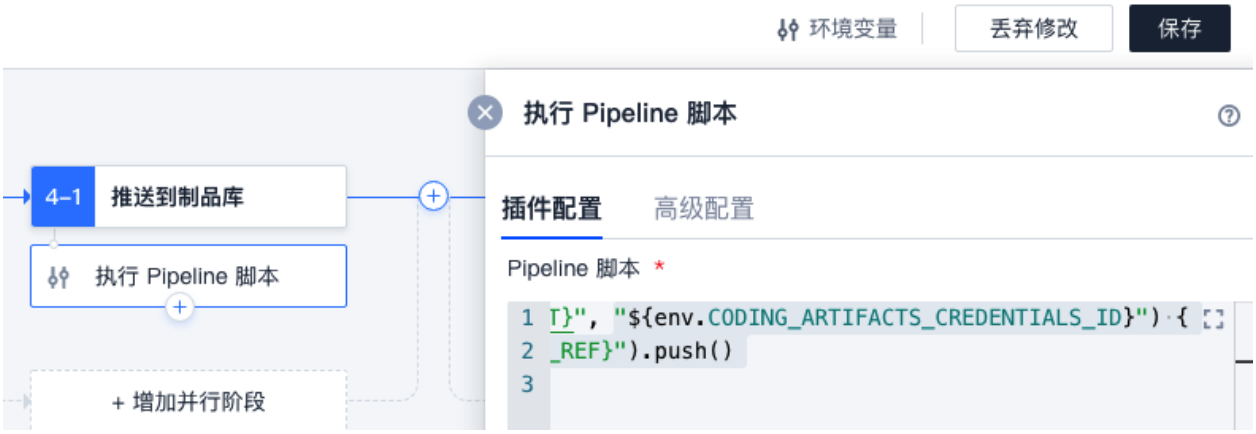
```
docker.build("${env.DOCKER_IMAGE_NAME}:${env.GIT_BUILD_REF}")
```

此处涉及的环境变量已在**开始阶段**中进行配置，而标签所内置的环境变量 GIT_BUILD_REF 的值对应检出代码步骤中的 Git 修订版本号。

步骤4：推送到制品库阶段

新建阶段并命名为构建 Docker 镜像。在步骤中添加**执行 Pipeline 脚本**，填写下文中的示例代码：

```
docker.withRegistry("https://${env.CODING_DOCKER_REG_HOST}", "${env.CODING_ARTIFACTS_CREDENTIALS_ID}") {
docker.image("${env.DOCKER_IMAGE_NAME}:${env.GIT_BUILD_REF}").push()
}
```



步骤5：署到 K8s 集群

1. 段并命名为部署到 K8s 集群，在步骤中添加**执行 Pipeline 脚本**，填写下文中的示例代码：

```
withKubeConfig([credentialsId: "${env.TKE_CLUSTER_CREDENTIAL_ID}"]) {
sh "kubectl patch deployment ruby-sinatra --patch '{\"spec\": {\"template\": {\"spec\": {\"containers\": [{\"name\": \"ruby-sinatra\", \"image\": \"${env.DOCKER_IMAGE_NAME}:${env.GIT_BUILD_REF}\"}], \"imagePullSecrets\": [{\"name\": \"ruby-sinatra-reg\"}]}}}}'"
}
```

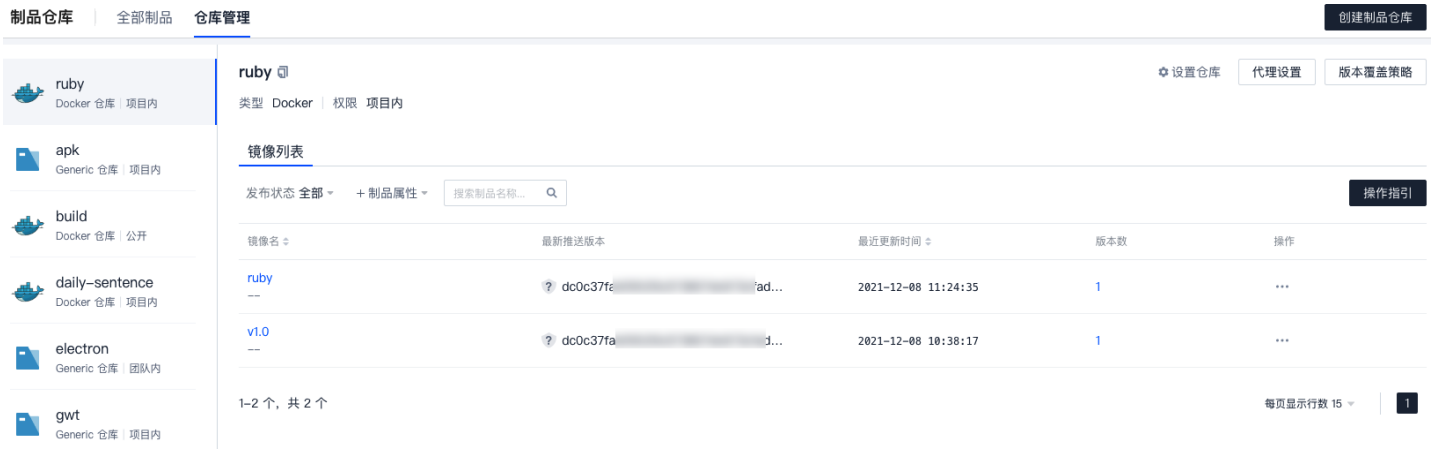
2. 完成配置后，单击**立即构建**手动触发构建过程。待整个流程运行完成后，可以查看每一步的运行日志。

配置自动化触发规则

在持续集成设置中还可以配置多种自动化触发规则，如果默认配置无法满足需求，还可自行配置所需的规则，详情请参见 触发规则。

查看构建产物

持续集成运行完成后，可以看到构建产物已自动上传至制品仓库中。



查看部署结果

构建计划运行成功后，可以前往 TKE 控制台查看 Deployment 的部署状态。

以上是一个简单的自动化部署示例项目。若实际项目需要体系化建设、回退机制、负载流量控制、发布时间窗口、更新策略等可以直接使用 持续部署 功能。

自动构建 Android 应用

最近更新时间：2022-03-25 15:16:44

本文将演示如何实现 Java-Android 项目进行自动化编译构建、签名、上传到 Generic 制品库进行发布这一完整过程。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

前置准备

- Git
- Java
- Gradle
- Android SDK
- CODING 项目
- 示例项目

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

手动构建

> 说明：
> 此部分内容将通过手动在本地编译、签名等过程，描述 .apk 文件的开发路径。若您已熟悉此流程，请参见 自动化构建。

将示例项目中的代码仓库 导入 至 CODING 项目中的代码仓库中。进行手动构建时推荐使用 Android Studio 编辑器进行开发。

编译构建

1. 示例仓库根目录下的 build.gradle 文件含有项目所包含的依赖，以及依赖拉取的来源。使用 Android Studio 打开项目，在根目录下调出终端，输入 ./gradlew test 命令进行测试。
2. 执行 ./gradlew assembleRelease 命令即可开始编译输出未签名的 apk 文件，更多的用法请参见 Android 开发者官方网站。

APK 签名

对 APK 文件进行签名将拥有以下优势：

应用名称	优势
应用升级	当安装应用的更新时，系统会比较新版本和现有版本中的证书。如果证书匹配，则系统允许更新。如果使用不同的证书为新版本签名，您必须为应用分配另一个软件包名称。在此情况下，用户会将新版本作为全新应用进行安装。
应用模块化	Android 允许通过同一证书签名的多个 APK 在同一个进程中运行（如果应用请求这样做），以便系统将其视为单个应用。这样一来，您便可以按模块部署您的应用，并且用户可以独立更新每个模块。
通过权限共享代码和数	Android 提供了基于签名的权限执行机制，以便一个应用可以将功能提供给使用指定证书签名的另一个应用。通过使用同一个证书为多个 APK 签名并使用基于签名的权限检查功能，您的应用可以采用安全的方式共享代码和数据。

据	

1. 在开始对文件签名前，需确保本地有 Java 环境。在终端中运行 keytool 生成 Android 签名证书。例如：

```
keytool -genkeypair -alias android.test -keyalg RSA -validity 36500 -keystore Java-android.p12 -storetype pkcs12
```

2. 其中 android.test 是证书的别名，将在后续过程中进行复用。运行命令后按照提示输入证书的密码等信息：

```
└─$ keytool -genkeypair -alias android.test -keyalg RSA -validity 36500 -keystore java-android.p12 -storetype pkcs12
输入密钥库口令：
再次输入新口令：
您的名字与姓氏是什么？
  [Unknown]:  Java
您的组织单位名称是什么？
  [Unknown]:  Test
您的组织名称是什么？
  [Unknown]:  Test
您所在的城市或区域名称是什么？
  [Unknown]:  SZ
您所在的省/市/自治区名称是什么？
  [Unknown]:  GD
该单位的双字母国家/地区代码是什么？
  [Unknown]:  CN
CN=Java, OU=Test, O=Test, L=SZ, ST=GD, C=CN是否正确？
  [否]:  是
```

3. 最终会在当前文件夹下面生成名为 Java-android.p12 的证书文件。使用 Android Studio 进行签名非常快捷，请参见 使用 Android Studio 对应用进行签名。

管理构建物

对发布物的管理与版本控制将会有助于应用的持续迭代。CODING Generic 类型制品库提供自定义的 Tag 作为版本标签，可以对 APK 文件进行管理和版本追溯。

1. 创建 Generic 类型制品库。

2. 将 APK 文件进行拖拽上传。

3. 查看版本列表管理历史版本。

自动化构建

手动进行编译、签名与上传 APK 文件至制品仓库本质上都是重复劳作。能否通过自动化工具实现代码更新后就获取最新的制品版本。下文将介绍如何使用持续集成代替人工执行这些重复操作。

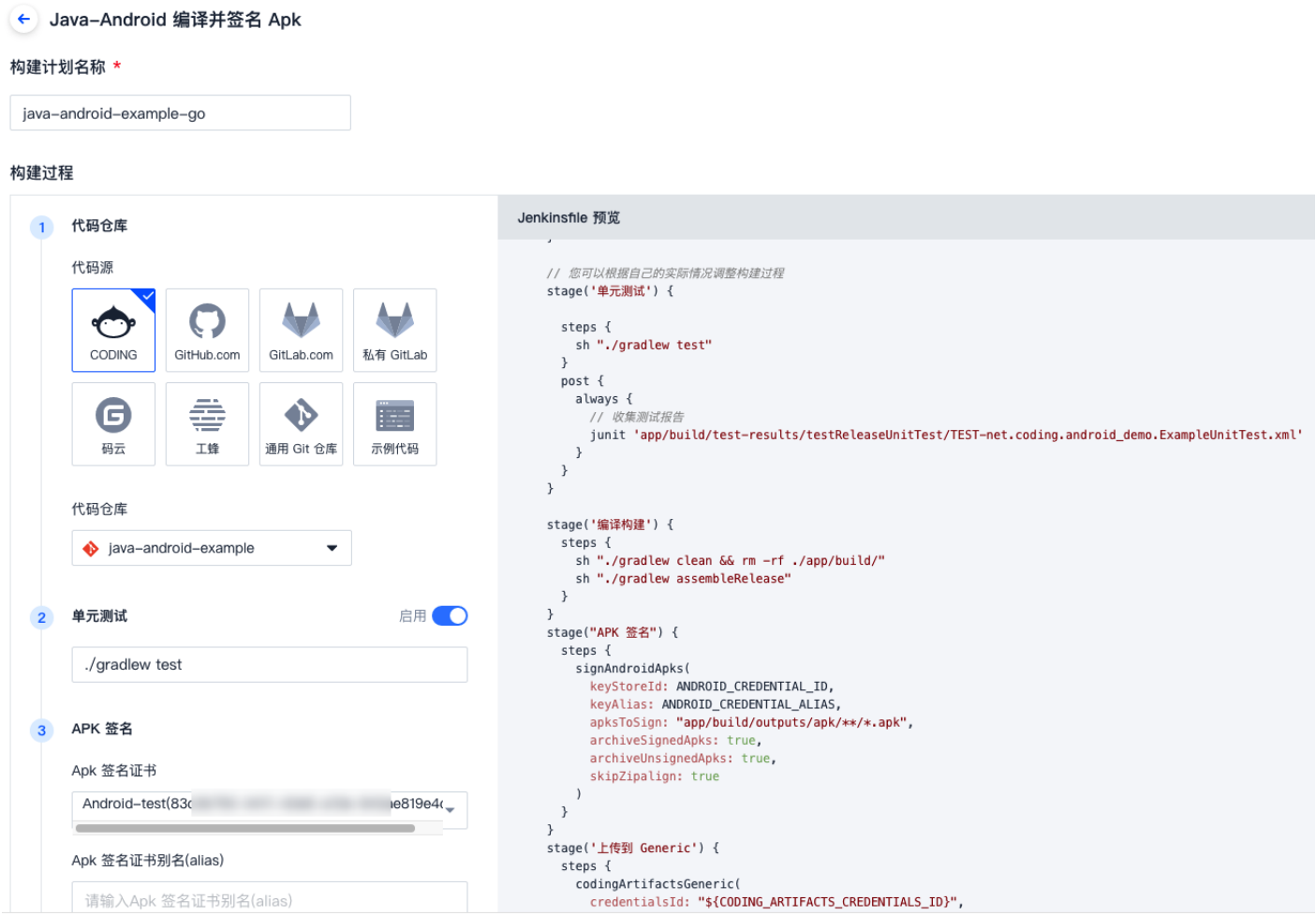
录入证书

在开始配置持续集成前，需要将 APK 签名上传至项目中。若通过在团队内明文共享安全证书，存在效率与安全问题。将证书以凭据的方式录入后，通过调用公开的凭据 ID 即可使用证书。

路径：**项目设置** > **开发者选项** > **凭据管理** > **Android 签名证书**，录入后同时还需勾选需授权的**持续集成计划**。

配置持续集成

1. 单击项目左侧菜单栏的**持续集成**，单击右上角的**创建构建计划**。选择 **Java-Android 编译并签名 APK** 模板。



2. 代码仓库选择已导入至项目中的代码仓库，APK 签名证书选择在上文中录入的凭据。单击**立即构建**后，持续集成将自动进行编译构建、签名、将构建物上传至 Generic 制品库。您可以在构建过程中查看各项步骤的执行情况。

3. 您还可以在持续集成设置中的触发规则中勾选**代码源触发**或其他自动触发条件，以实现在推送代码时自动触发持续集成任务，完成应用构建。

总结

至此，我们已经将整个 Android 开发流程中重要的操作步骤完全自动化流水线化，这极大地简化了人工操作，让开发者能更好的从繁琐的开发流程中脱离出来，专注于代码功能的迭代研发。

将 React 项目发布至腾讯云 COS

最近更新时间：2022-04-13 15:11:48

本文将介绍如何使用 CODING 持续集成实现将 React 框架 Web 应用自动发布至腾讯云 COS（对象存储）服务。

> 说明：
> COS 对象存储常用于管理网站中所需的 html、css、js 文件、图片与视频等静态资源。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

前置准备

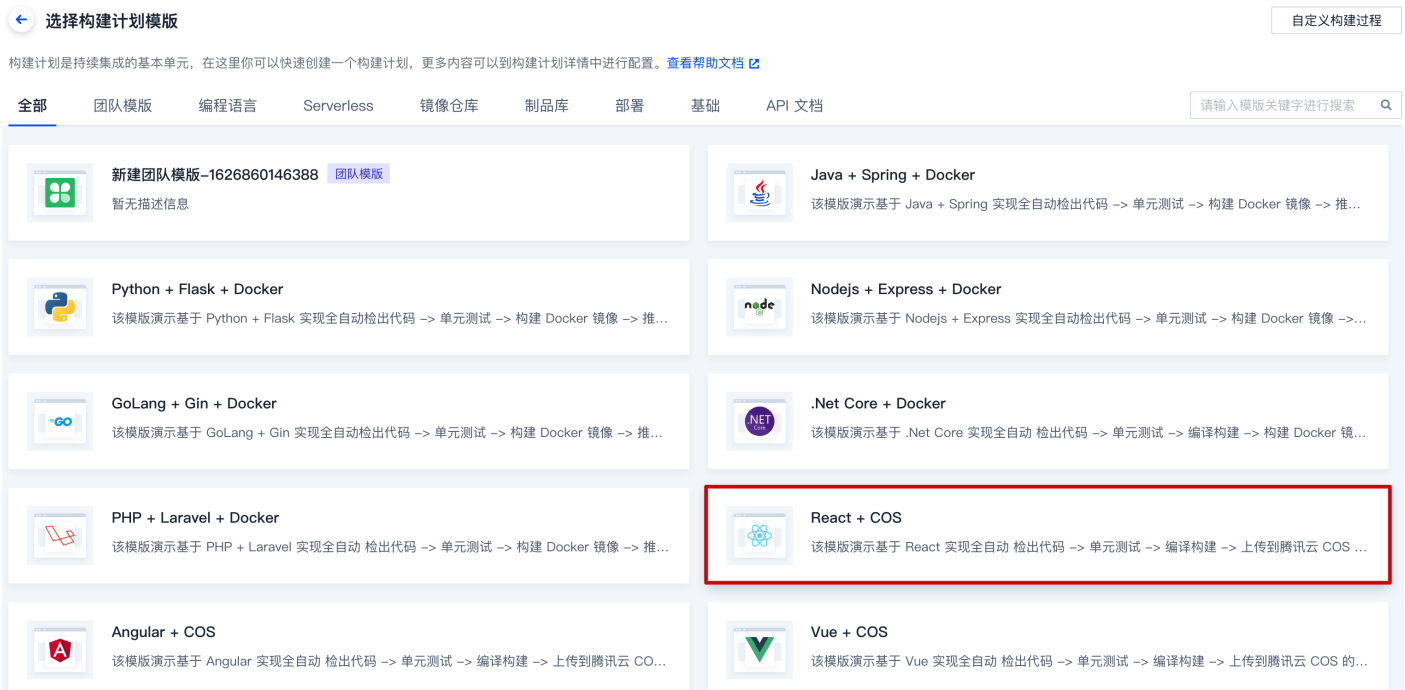
- Git
- node 8.16.0或者版本为10.16.0及以上
- CODING 项目
- 腾讯云 COS 存储桶

进入项目

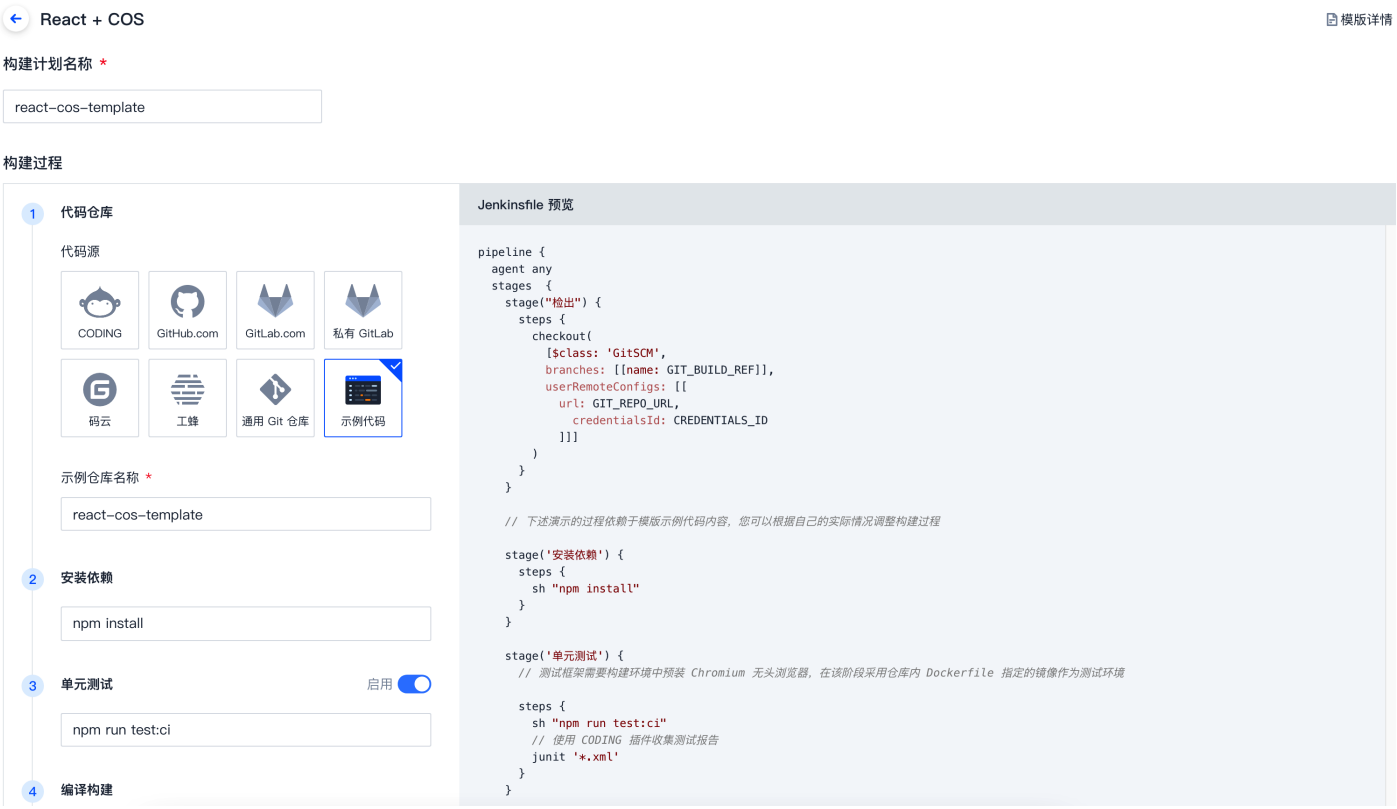
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

配置构建计划

1. 进入项目后，进入左侧菜单栏的**持续集成**功能，单击右上角的**创建构建计划**并选择 React+COS 模板。



2. 自定义构建计划名称、选择示例代码。



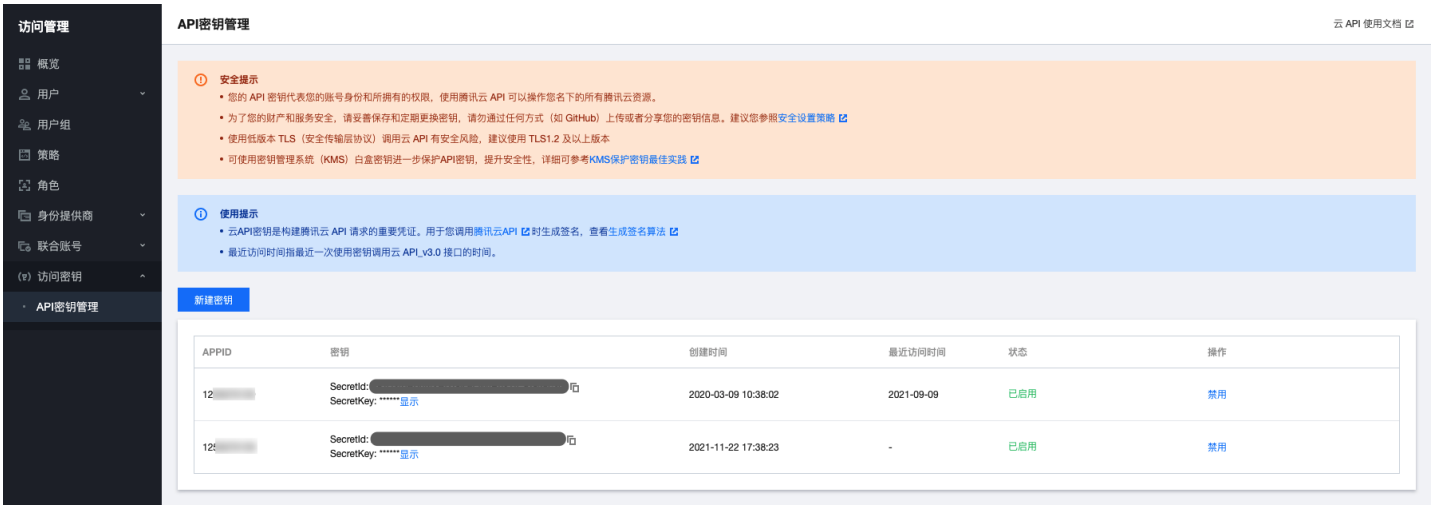
上传到 COS Bucket

此步骤需要将腾讯云 COS 服务填写至构建过程中的四个环境变量中：

- COS_SECRET_ID – 腾讯云访问密钥 **SecretId**
- COS_SECRET_KEY – 腾讯云访问密钥 **SecretKey**
- COS_BUCKET_NAME – 腾讯云 COS 存储桶名称
- COS_BUCKET_REGION – 腾讯云 COS 存储桶地区

获取访问密钥

前往 访问管理 中的**访问密钥 > API 密钥管理**获取相应的密钥 SecretId 与 SecretKey。

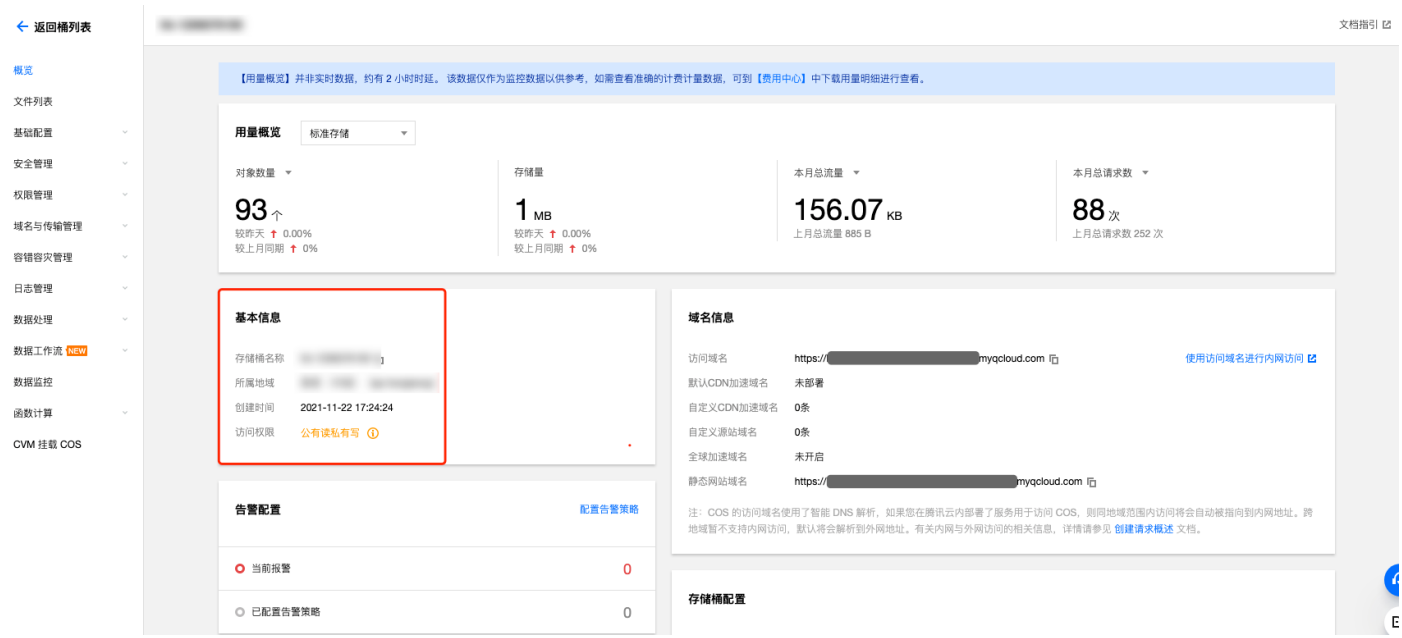


> **说明：**

> 将密钥填写至构建过程中将自动隐藏加密，保障数据安全。

获取 COS 基本信息

1. 前往 存储桶列表，单击目标存储桶的概览获取名称与地区信息。



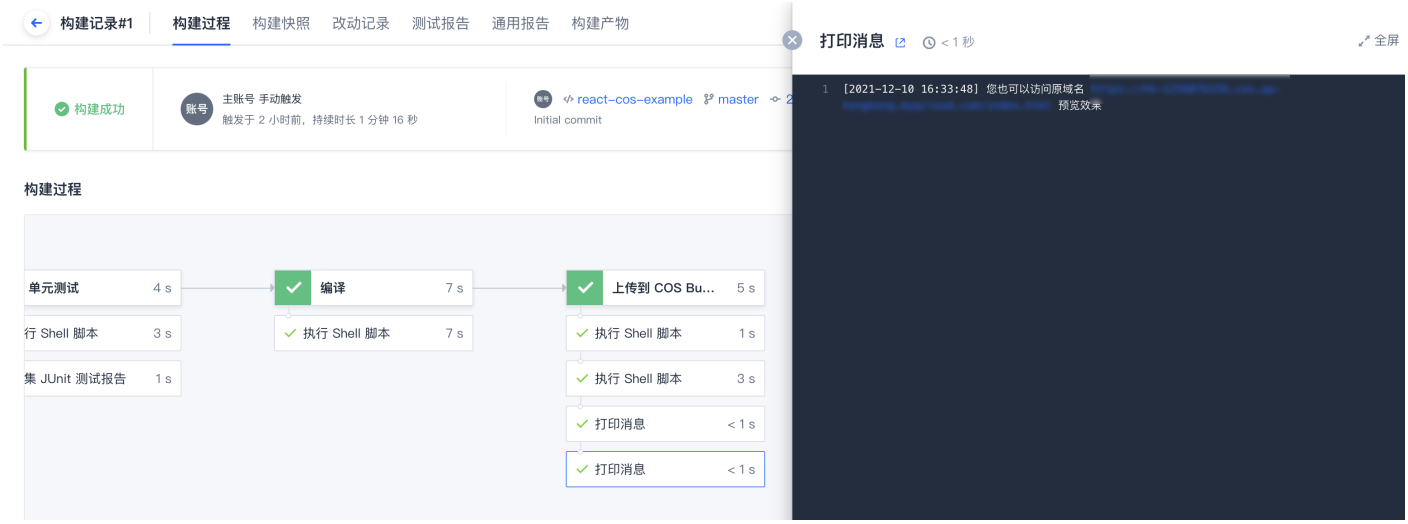
> **注意：**
> 若访问权限不具备公有读权限有可能影响发布结果。

2. 填写完成后，单击**确定**触发构建流程。

查看构建成果

1. 触发构建后，在**构建过程**中可以看到构建的完整日志。

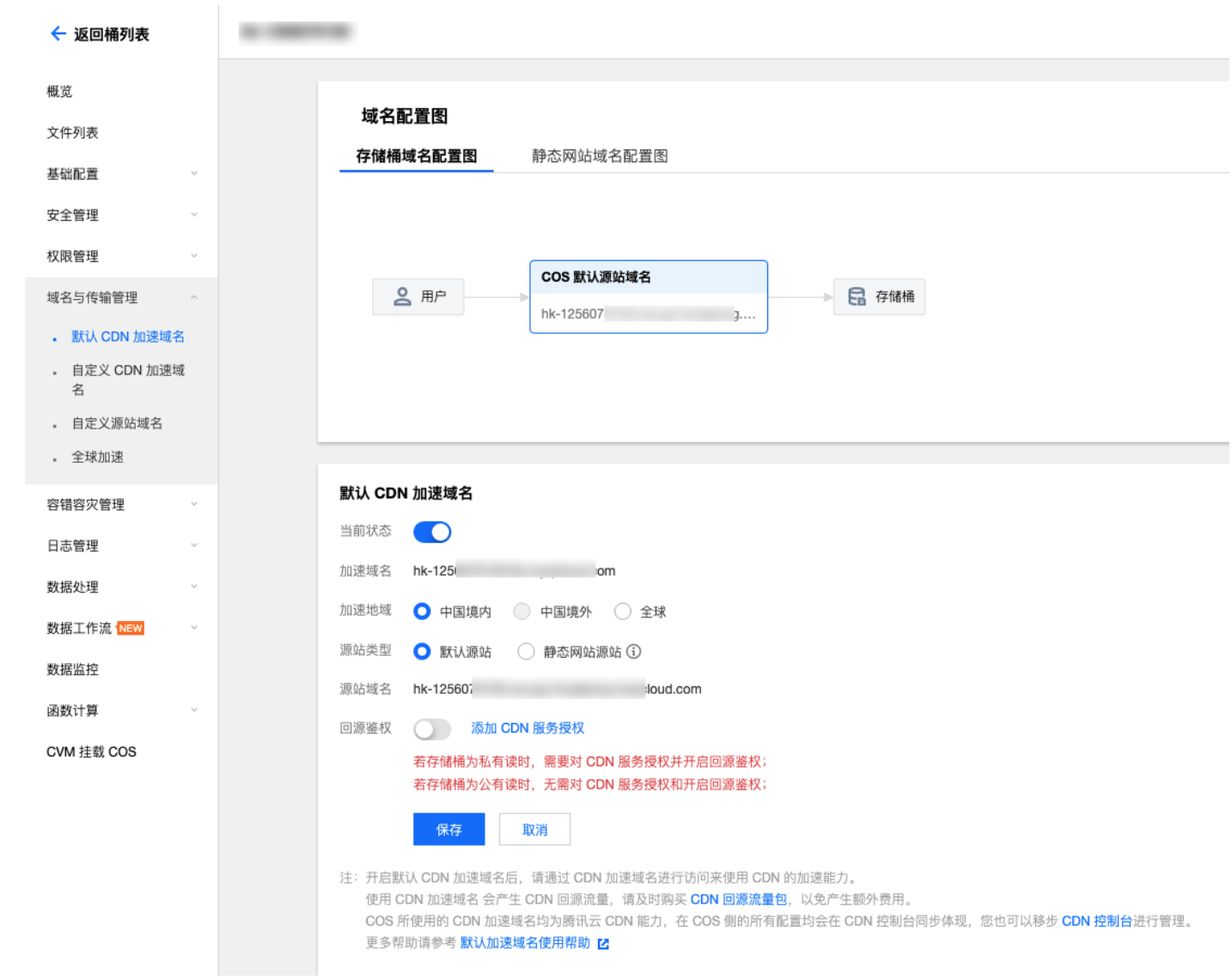
2. 单击**上传到 COS Bucket** 阶段中最后一个**打印消息**处的 COS 地址，即可跳转到构建完毕的 React Web 应用。



3. 一个全新的 React demo 页面已构建完成。

配置 CDN 加速（可选）

1. CDN 服务常用于静态网站加速，能够提升网站的访问速度。单击前往 COS 存储桶 中的**域名与传输管理** > **默认 CDN 加速域名**，开启默认 CDN 加速域名。



2. 保持默认选项即可。由于我们配置了公有读的存储桶，无需回源鉴权。配置完成后等待部署完成，即可获得 CDN 加速效果。

配置触发规则

1. 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。前往**持续集成设置** > **触发规则**进行配置。

2. 若勾选**代码源触发**，将自动监听符合规则的触发事件，例如 master 分支的代码更新，自动触发持续集成任务完成应用部署。

总结

至此，通过配置持续集成任务，已实现自动化发布 Web 应用至腾讯云 COS 存储桶流程，提升了开发前端工程时的工作效率。

将 VUE 项目发布至腾讯云 COS

最近更新时间：2022-03-25 15:16:54

本文将介绍如何使用 CODING 持续集成实现将 VUE 框架 Web 应用自动发布至腾讯云 COS（对象存储）服务。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

前置准备

- Git
- node 8.9 或更高版本
- CODING 项目
- 腾讯云 COS 存储桶

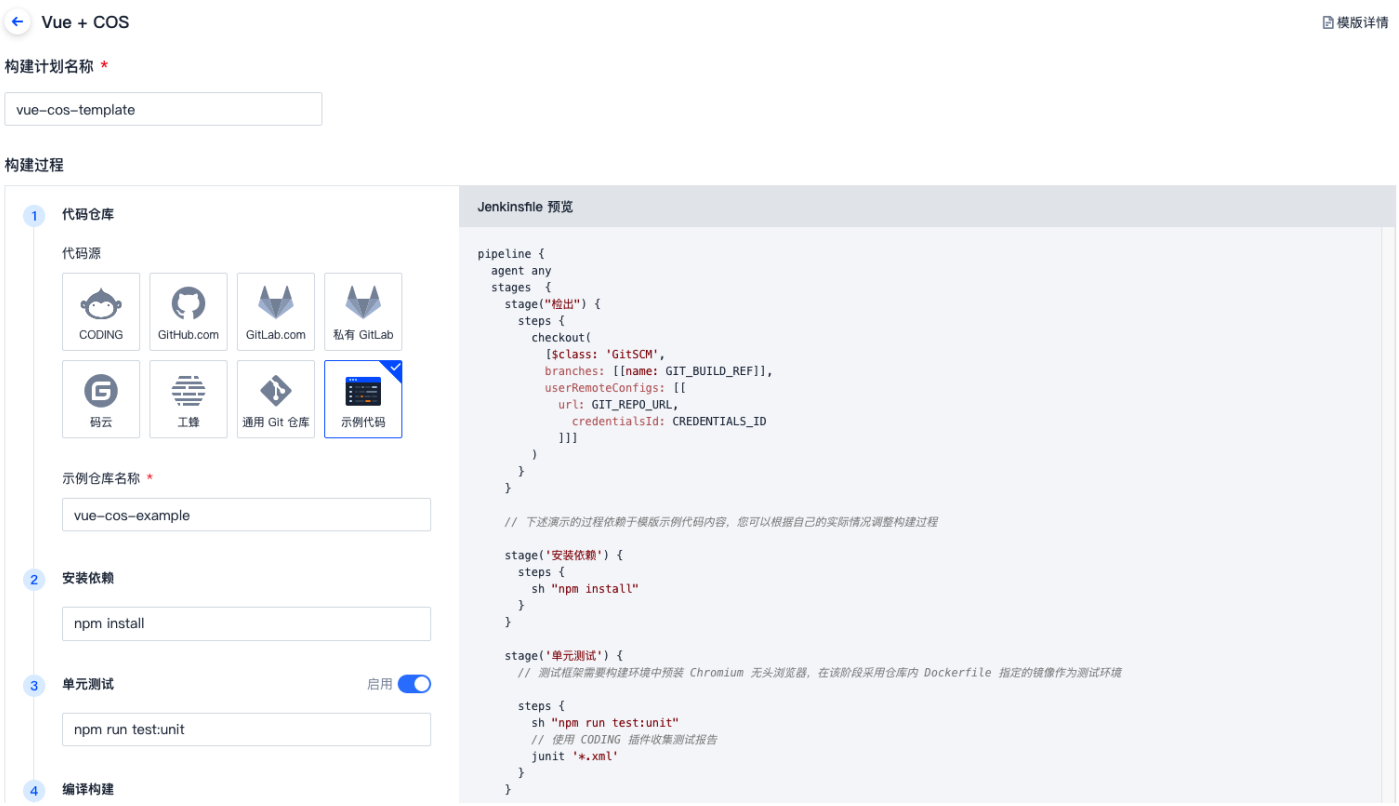
进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

配置构建计划

1. 进入项目后，进入左侧菜单栏的**持续集成**功能，单击右上角的**创建构建计划**并选择 Vue+COS 模板。

2. 自定义构建计划名称、选择示例代码。



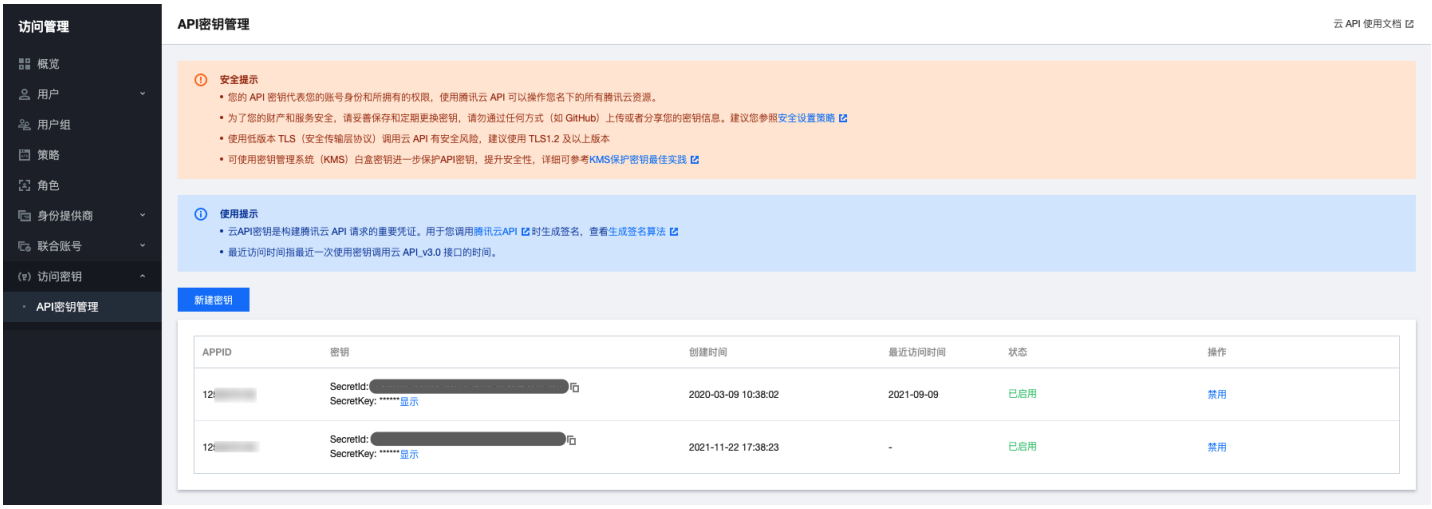
上传到 COS Bucket

此步骤需要将腾讯云 COS 服务填写至构建过程中的四个环境变量中：

- COS_SECRET_ID – 腾讯云访问密钥 **SecretId**
- COS_SECRET_KEY – 腾讯云访问密钥 **SecretKey**
- COS_BUCKET_NAME – 腾讯云 COS 存储桶名称
- COS_BUCKET_REGION – 腾讯云 COS 存储桶地区

获取访问密钥

前往 访问管理 中的**访问密钥 > API 密钥管理**获取相应的密钥 SecretId 与 SecretKey。

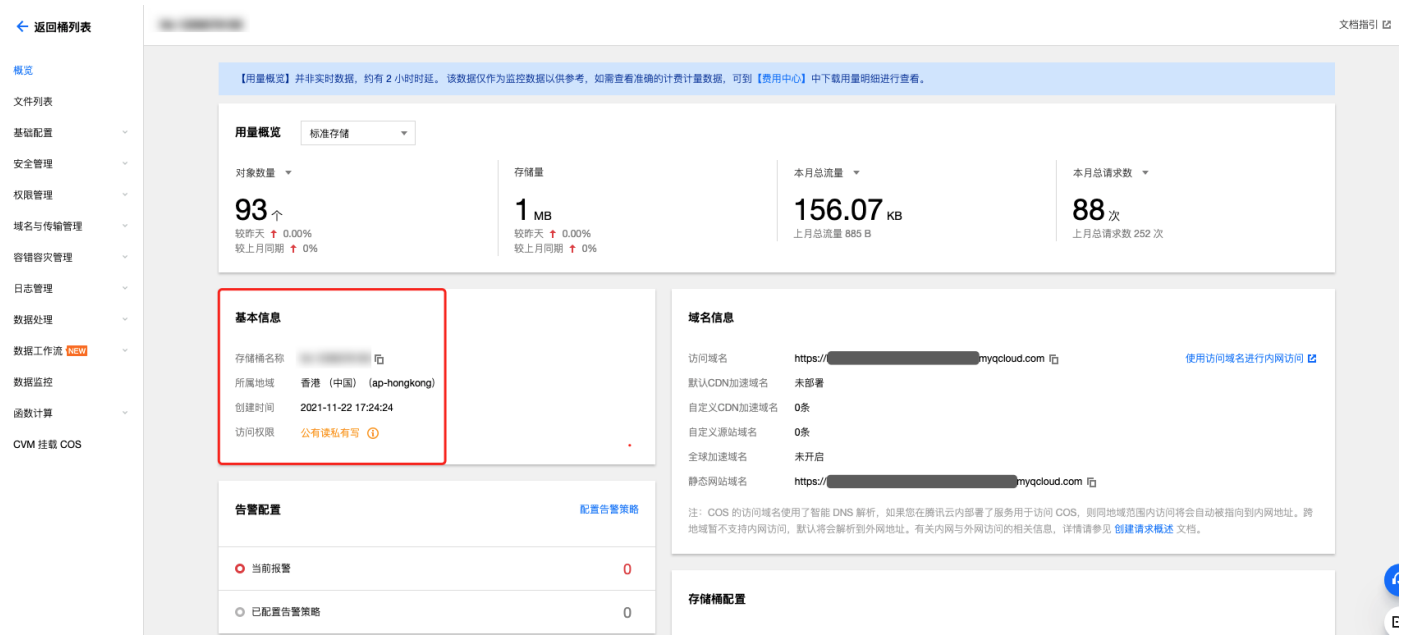


> ⚠ 注意：

将密钥填写至构建过程中将自动隐藏加密，保障数据安全。

获取 COS 基本信息

1. 前往 存储桶列表，单击目标存储桶的概览获取名称与地区信息。



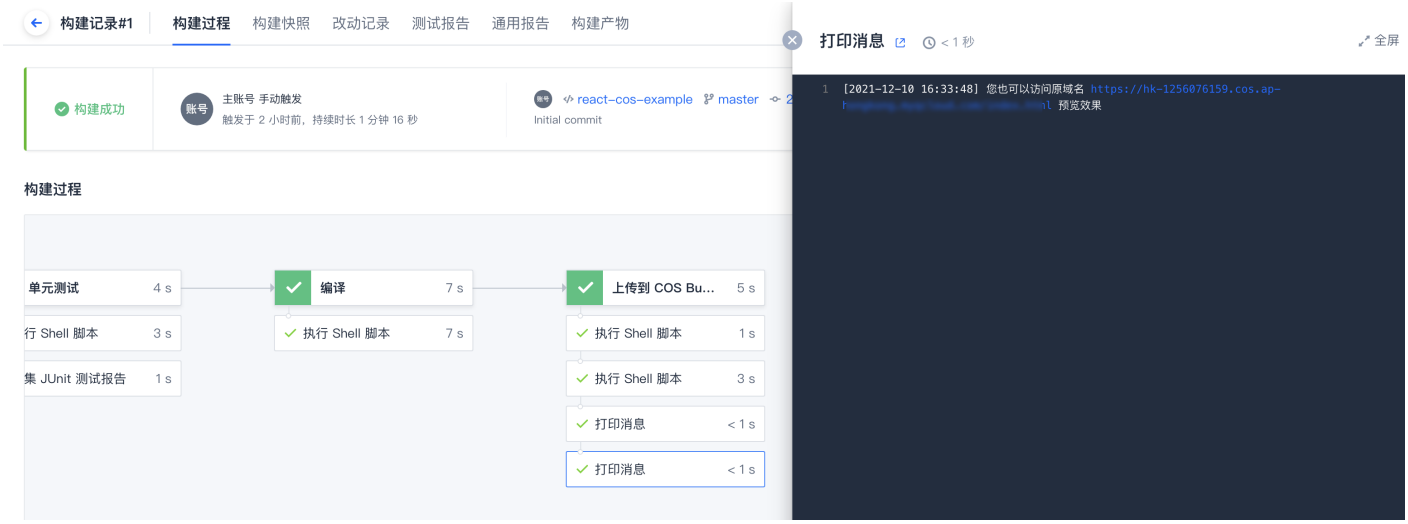
> **说明：**
> 若访问权限不具备公有读权限有可能影响发布结果。

2. 填写完成后，单击**确定**触发构建流程。

查看构建成果

1. 触发构建后，在**构建过程**中可以看到构建的完整日志。

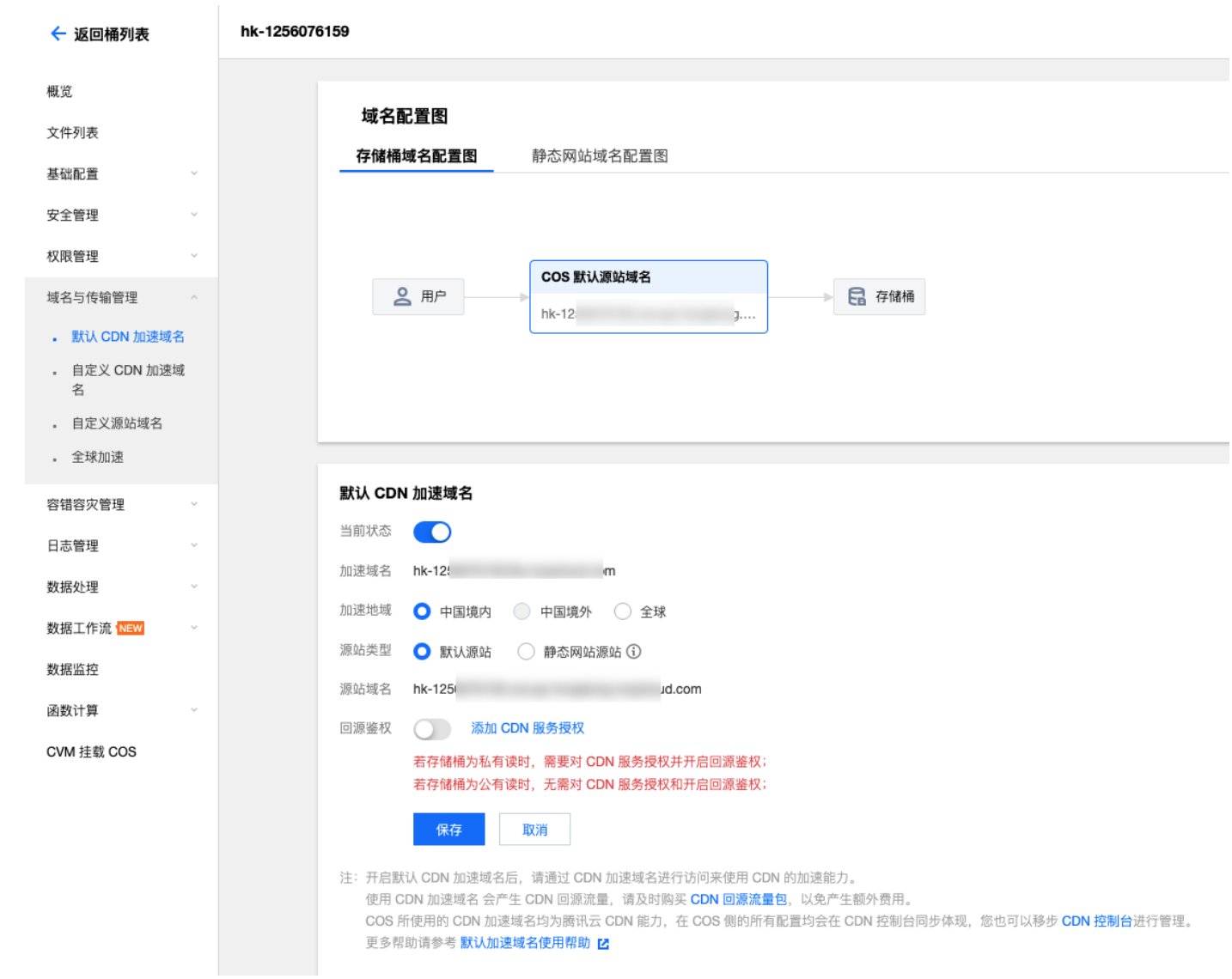
2. 单击**上传到 COS Bucket** 阶段中最后一个**打印消息**处的 COS 地址，即可跳转到构建完毕的 Vue Web 应用。



3. 一个全新的 Vue Demo 页面已构建完成。

配置 CDN 加速（可选）

1. CDN 服务常用于静态网站加速，能够提升网站的访问速度。单击前往 **COS 存储桶** 中的**域名与传输管理 > 默认 CDN 加速域名**，开启默认 CDN 加速域名。



2. 保持默认选项即可。由于我们配置了公有读的存储桶，无需回源鉴权。配置完成后等待部署完成，即可获得 CDN 加速效果。

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。前往**持续集成设置** > **触发规则**进行配置。

若勾选代码源触发，将自动监听符合规则的触发事件，例如 master 分支的代码更新，自动触发持续集成任务完成应用部署。

总结

至此，通过配置持续集成任务，已实现自动化发布 Web 应用至腾讯云 COS 存储桶流程，提升了开发前端工程时的工作效率。

代码规范检查
增量检查

最近更新时间：2022-03-25 15:16:58

本文为您介绍如何通过持续集成实现对检查增量代码。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

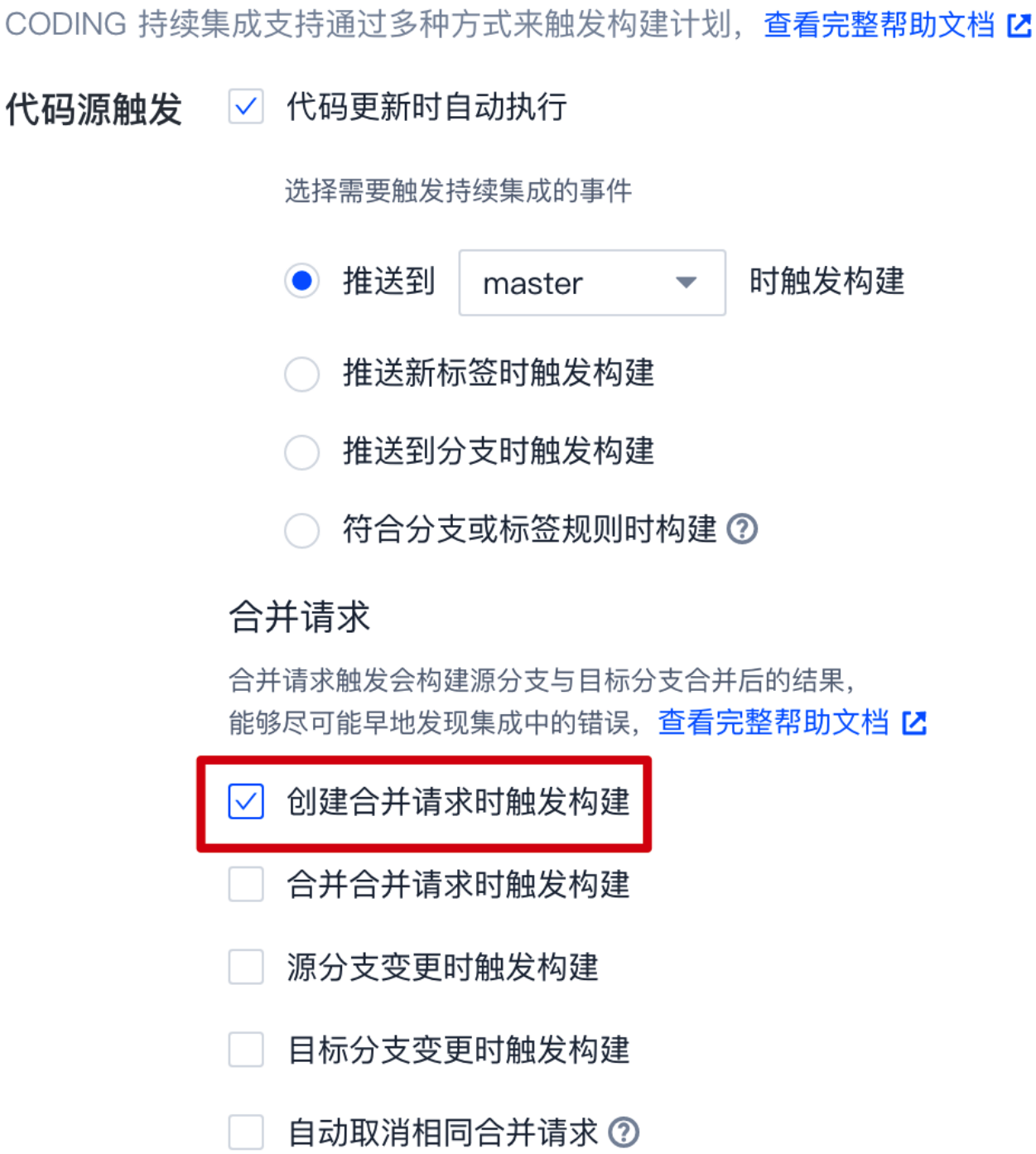
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

功能介绍

增量检查机制多用于代码合并请求场景，当有新的代码变更时将自动触发持续集成任务，对代码的变动情况进行规范性检查。

开始使用

1. 在持续集成的触发规则中开启创建合并请求时触发构建。



2. 参考并使用下述 Jenkinsfile。

Jenkinsfile

```
pipeline {
agent any
stages {
stage('检出') {
steps {
checkout([
$class: 'GitSCM',
```

```
branches: [[name: '*']],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
script {
if ( env.MR_SOURCE_BRANCH ==~ /.*/ ) {
sh "git checkout ${env.MR_TARGET_BRANCH}"
sh "git checkout ${env.MR_SOURCE_BRANCH}"
} else {
sh "git checkout ${env.GIT_COMMIT}"
}
}
}
}
stage('安装依赖') {
steps {
sh 'npm install -g lint-md-cli'
}
}
stage('增量检查代码规范') {
when {
changeRequest()
}
steps {
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... | xargs lint-md"
}
}
}
}
```

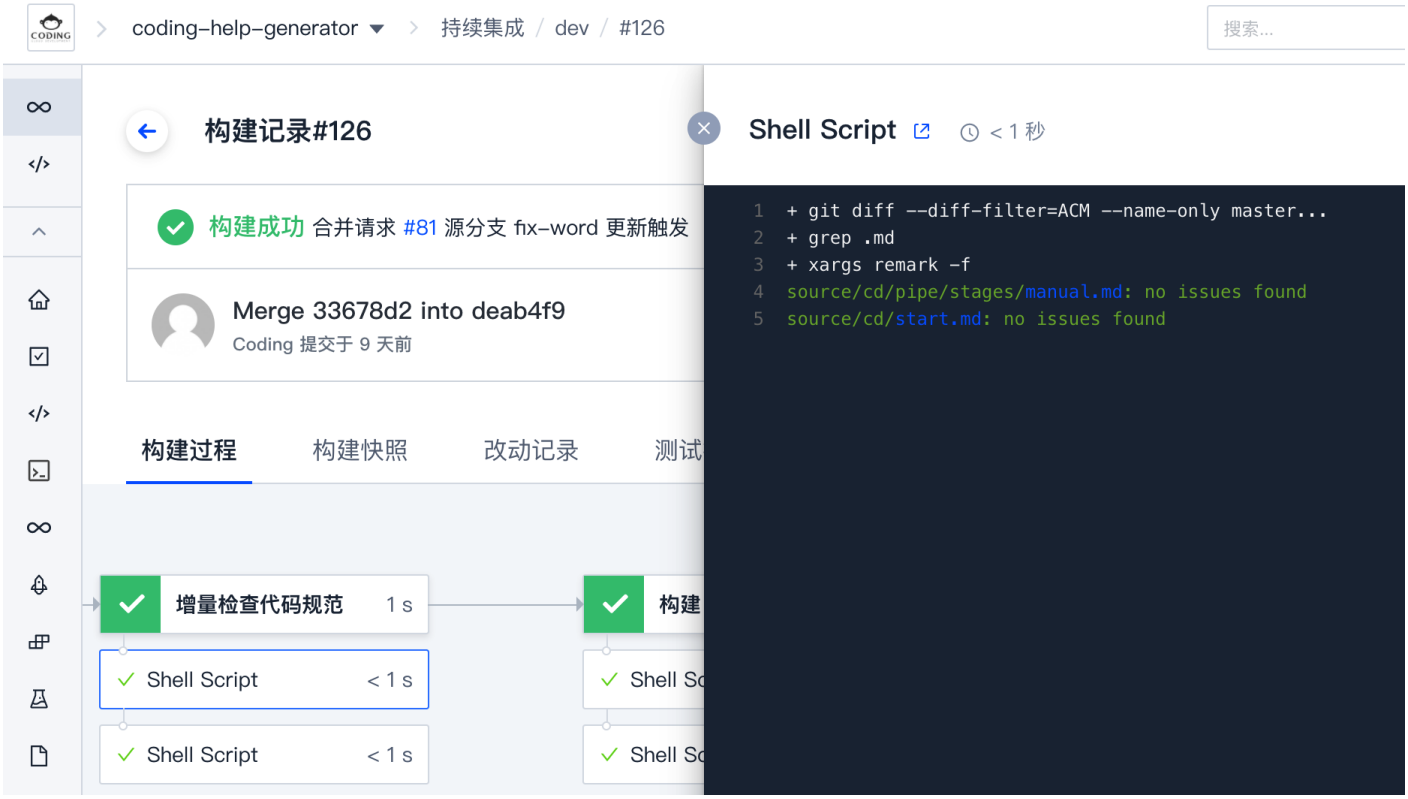
运行结果

- **增量代码未通过状态检查，拦截代码合并：**

- **在构建计划的日志中将展示错误记录：**

- **增量代码通过状态检查，允许代码合并：**

- **增量代码通过状态检查：**



Commit Message

最近更新时间：2022-03-25 15:17:03

本文为您介绍如何通过持续集成检查 Commit Message 是否符合规范。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

书写工具

全局安装 commitizen，即可使用 git cz 命令取代 git commit，提供交互式选择界面，协助书写。

```
npm install -g commitizen cz-conventional-changelog
echo '{ "path": "cz-conventional-changelog" }' > ~/.czrc
```

```
coding-help-generator$ git add source/
coding-help-generator$ git cz
cz-cli@4.2.1, cz-conventional-changelog@3.3.0

? Select the type of change that you're committing: (Use arrow keys)
❯ feat:     A new feature
  fix:      A bug fix
  docs:     Documentation only changes
  style:    Changes that do not affect the meaning of the code (white-space
  refactor: A code change that neither fixes a bug nor adds a feature
  perf:     A code change that improves performance
  test:     Adding missing tests or correcting existing tests
(Move up and down to reveal more choices)
```

项目配置

在项目代码中通过 npm 安装 lint 工具（任何语言的项目都可以安装，例如 PHP、Java）。

```
npm install --save-dev @commitlint/cli @commitlint/config-conventional @commitlint/prompt cz-conventional-changelog
husky
```

修改 package.json，即可实现本地提交前自动检查：

```
{
"config": {
"commitizen": {
"path": "./node_modules/cz-conventional-changelog"
}
},
"husky": {
"hooks": {
"commit-msg": "commitlint -E HUSKY_GIT_PARAMS"
}
},
"devDependencies": {
"@commitlint/cli": "^9.1.2",
"@commitlint/config-conventional": "^11.0.0",
"@commitlint/prompt": "^9.1.2",
"cz-conventional-changelog": "^3.3.0",
"husky": "^4.3.0"
}
}
```

Jenkinsfile

在持续集成中使用下列代码，即可实现：合并请求时，对比当前分支和目标分支的 git log，进行增量检查。

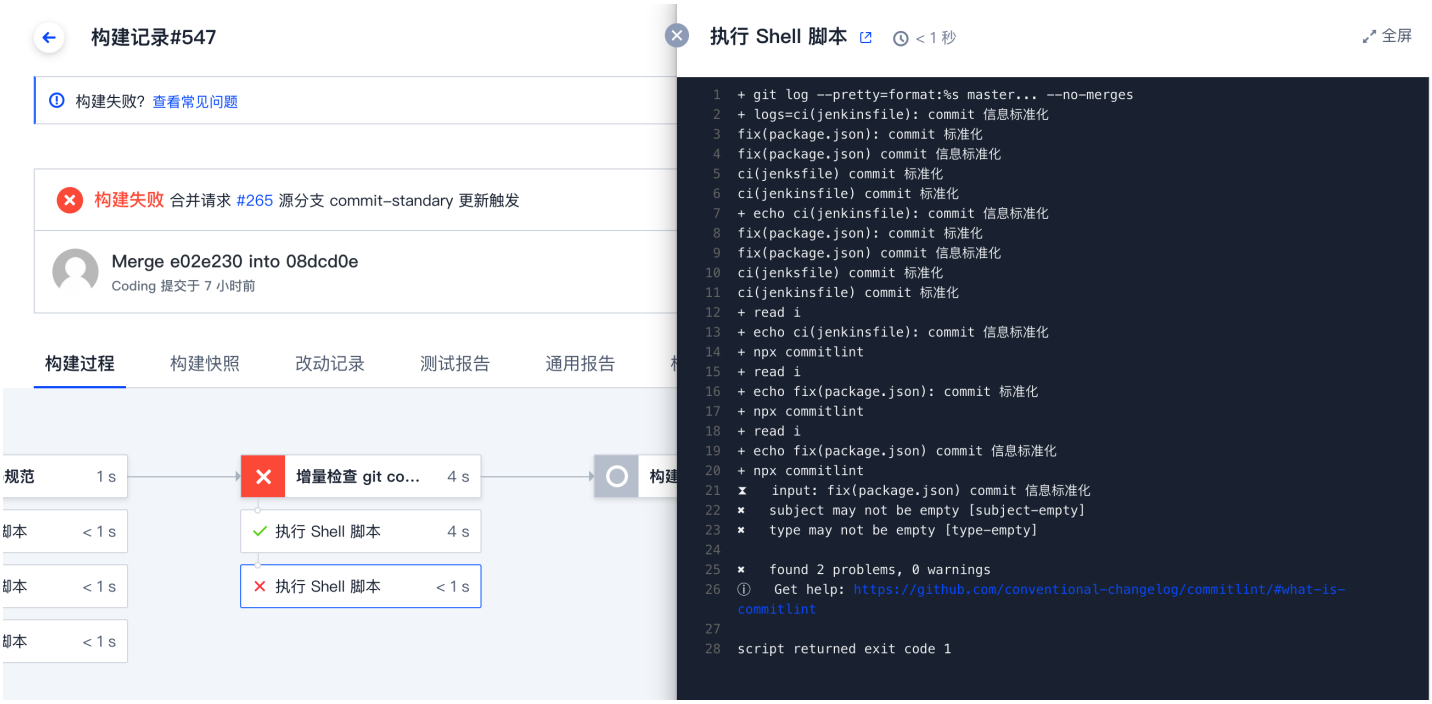
```
pipeline {
agent any
stages {
stage('检出') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: '*']],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
script {
if ( env.MR_SOURCE_BRANCH ==~ /.*/ ) {
sh "git checkout ${env.MR_TARGET_BRANCH}"
sh "git checkout ${env.MR_SOURCE_BRANCH}"
} else {
sh "git checkout ${env.GIT_COMMIT}"
}
}
}
}
stage('安装依赖') {
steps {
sh 'npm install'
```

```
}
}
stage('增量检查 git commit') {
when {
changeRequest()
}
steps {
script {
sh 'npm install'
sh """logs=`git log --pretty=format:'%s' ${env.MR_TARGET_BRANCH}... --no-merges`;
echo "\$logs" | while read i; do echo \$i | npx commitlint; done
"""
}
}
}
}
}
```


dev
基础信息
流程配置
触发规则
变量与缓存
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
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

运行结果

CI 运行不通过时报错：



修改提交信息

1. 如果 Git 提交信息不规范，有两种修改方式：

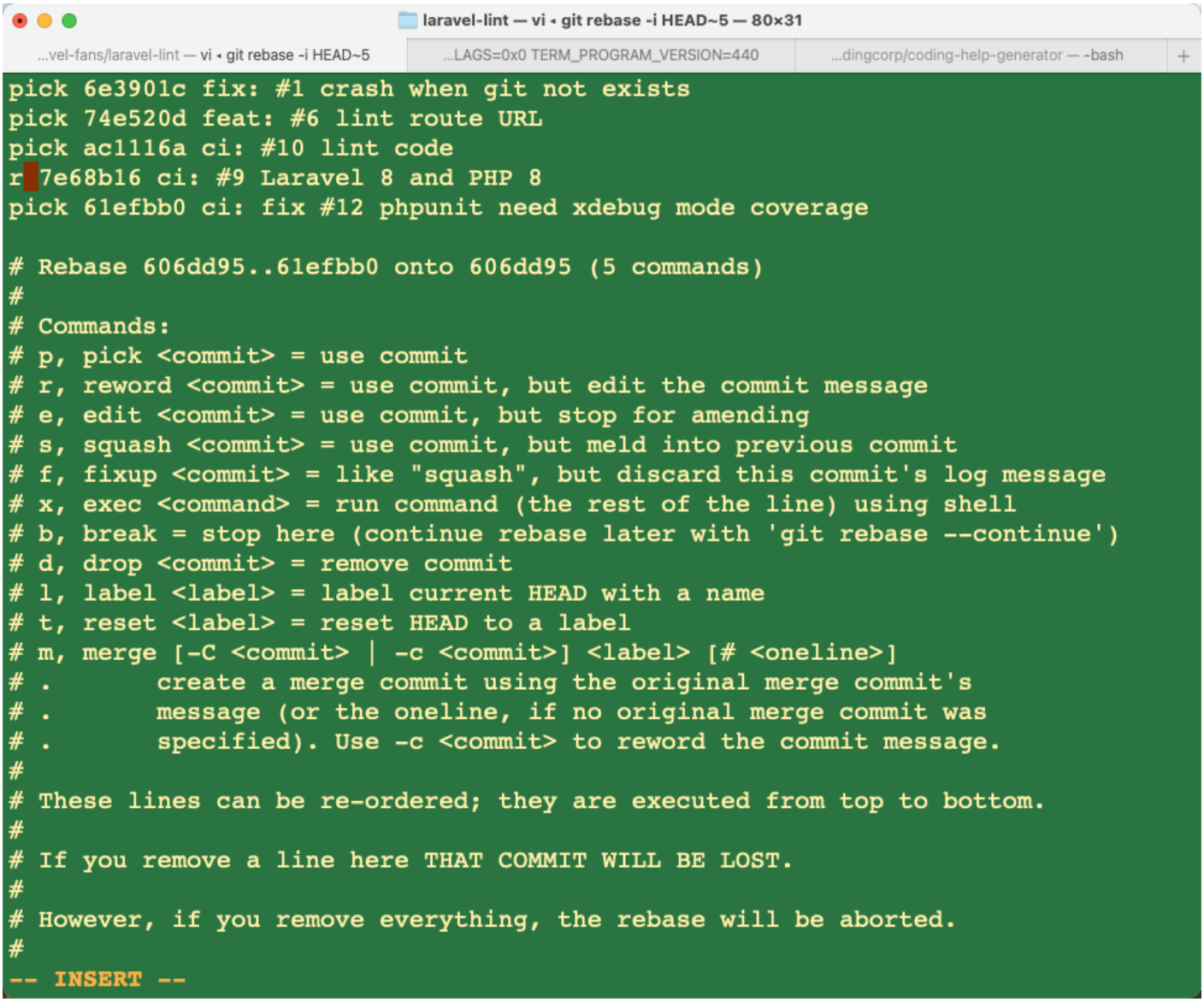
修改最后一次提交信息

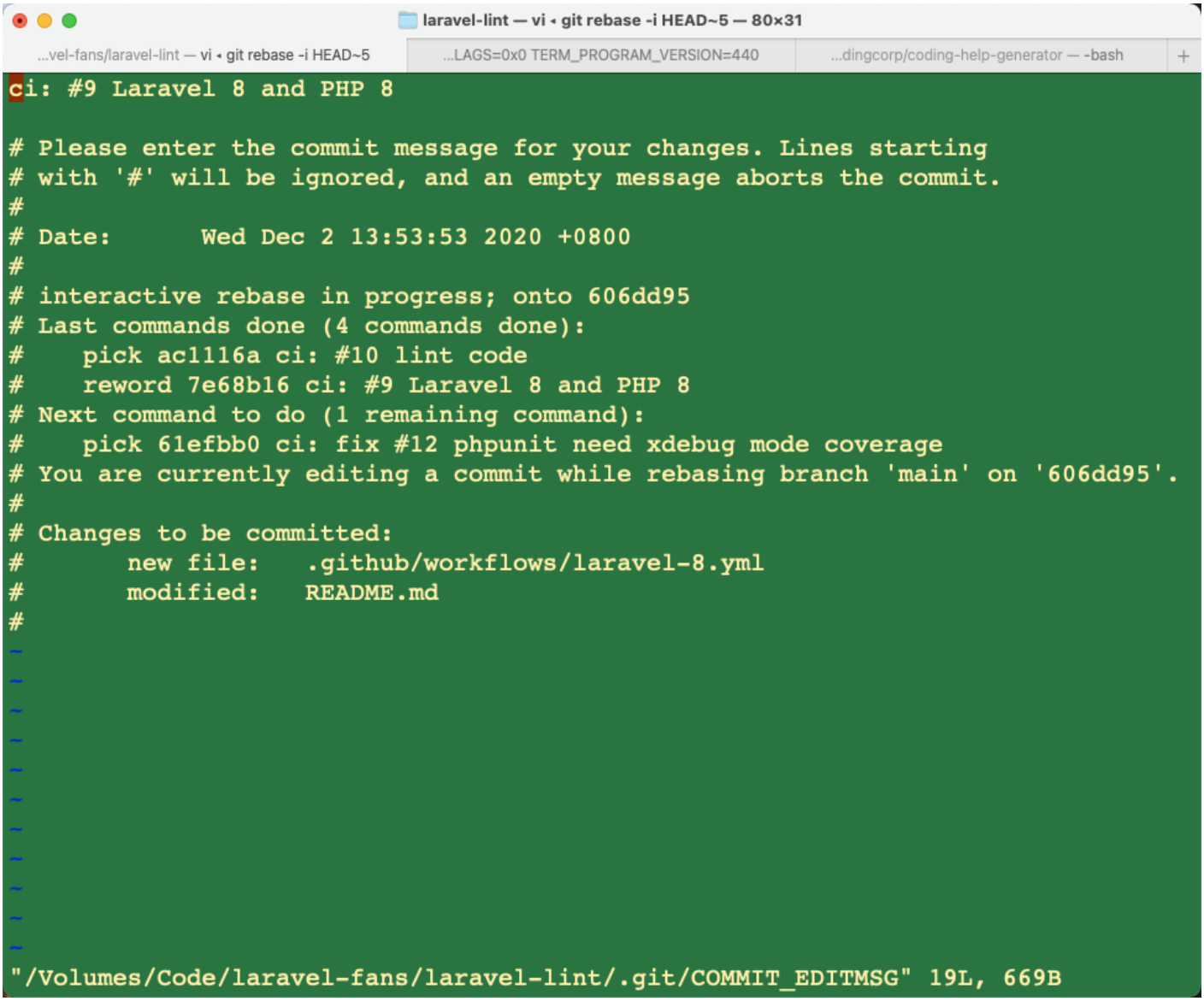
```
git commit --amend
```

修改中间某条提交信息

```
git rebase -i HEAD~5
```

2. 把某条前面的 pick 修改为 r，保存退出（VI 为：x），即可进入编辑界面。





3. 修改完毕，强制推送即可，命令：

```
git push -f origin issue-123
```

Java

最近更新时间：2022-03-25 15:17:07

本文为您介绍如何通过持续集成检查 Java 规范。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

Java 常用代码规范检查工具如下：

工具	lint 命令	支持 IDE
Checkstyle	./gradlew check	VSCode、IDEA
PMD	./gradlew check	VSCode、IDEA
P3C-PMD	./gradlew check	VSCode、IDEA

Checkstyle

Checkstyle 内置2种规范：Google 与 Sun，其中常用的 Google Java Style 规范包括下列规则：

- 每行代码最大长度100个字符。
- 缩进使用2个空格。

安装

本文以 Checkstyle Gradle Plugin 为例，而 Maven 老项目可使用命令一键升级到 Gradle：

```
gradle init --type pom
```

修改 build.gradle：

```
plugins {
id 'checkstyle'
}
checkstyle {
toolVersion = '8.39'
maxWarnings = 0
maxErrors = 0
}
```

下载代码规范 XML 文件，保存到项目中：

```
wget https://raw.githubusercontent.com/checkstyle/checkstyle/checkstyle-8.39/src/main/resources/google_checks.xml -O config/checkstyle/checkstyle.xml
```

全量检查

```
$ ./gradlew check

[WARN] TaskTest.java:543: 本行字符数 101个，最多：100个。 [LineLength]
[WARN] ReportTest.java:206:9: 第 9 个字符 '}'应该与下一部分位于同一行。
[WARN] ProjectRoleTest.java:449:8: 注释应缩进8个缩进符，而不是7个。
```

增量检查

修改 build.gradle：

```
plugins {
id 'checkstyle'
}
checkstyle {
toolVersion = '8.39'
maxWarnings = 0
maxErrors = 0
}

task checkstyleChanged(type: Checkstyle) {
source "${project.rootDir}"
def changedFiles = getChangedFiles()
if (changedFiles) {
// include changed files only
include changedFiles
} else {
// if no changed Java files detected, exclude all
exclude "**/*"
}
classpath = files()
showViolations = true

// Define the output folder for the generated reports
def reportsPath = "${buildDir}/reports/checkstyle"
reports {
html.enabled true
html.destination rootProject.file("${reportsPath}/changed-files.html")

xml.enabled true
xml.destination rootProject.file("${reportsPath}/changed-files.xml")
```

```
}
}

static def getChangedFiles() {
ByteArrayOutputStream systemOutStream = new ByteArrayOutputStream()
def diffFile = ''
def files = []

try {
diffFile = new FileInputStream(".diff").getText();
} catch (FileNotFoundException e) {
("git diff --name-only --diff-filter=d HEAD").execute().waitForProcessOutput(systemOutStream, System.err)
diffFile = systemOutStream.toString()
systemOutStream.close()
}

// Collect only *.java-files from all changed files
for (file in diffFile.trim().split('\n')) {
if (file.endsWith(".java")) {
files.add(file)
}
}
return files
}
```

本地运行：

```
./gradlew checkstyleChanged
```

持续集成合并请求时运行：

```
pipeline {
agent any
stages {
stage('检出') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: '*']],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
script {
sh 'touch .diff'
if (env.MR_SOURCE_BRANCH ==~ /.*/) {
sh "git checkout ${env.MR_TARGET_BRANCH}"
sh "git checkout ${env.MR_SOURCE_BRANCH}"
```

```
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... > .diff"
} else {
sh "git checkout ${env.GIT_COMMIT}"
}
}
}
}
stage('增量检查代码规范') {
when {
changeRequest()
}
agent {
docker {
image 'adoptopenjdk:11-jdk-hotspot'
args '-v /root/.gradle/:/root/.gradle/ -v /root/.m2/:/root/.m2/'
reuseNode true
}
}
steps {
sh './gradlew checkstyleChanged'
}
}
}
}
```

PMD

PMD maven 包中内置了 xml 规则文件，安装即可使用，无需单独下载 xml，也可以自定义规则，例如 Java 规范。

安装

本文以 PMD Gradle Plugin 为例，而 Maven 老项目可使用命令一键升级到 Gradle：

```
gradle init --type pom
```

此处以 Gradle 6.8.3为例，老版本的 Gradle 可能报错，请先升级，修改 gradle/wrapper/gradle-wrapper.properties：

```
distributionUrl=https\://mirrors.cloud.tencent.com/gradle/gradle-6.8.3-bin.zip
```

修改 build.gradle：

```
plugins {
id 'pmd'
}
pmd {
consoleOutput = true
rulesMinimumPriority = 5
```

```
// 官方规范列表（无需下载） https://github.com/pmd/pmd/tree/master/pmd-java/src/main/resources/category/java
// ruleSets = ["category/java/codestyle.xml", "category/java/bestpractices.xml"]
// 规范列表（无需下载） https://github.com/alibaba/p3c/tree/master/p3c-pmd/src/main/resources/rulesets/java
ruleSets = [
"rulesets/java/ali-comment.xml",
"rulesets/java/ali-concurrent.xml",
"rulesets/java/ali-constant.xml",
"rulesets/java/ali-exception.xml",
"rulesets/java/ali-flowcontrol.xml",
"rulesets/java/ali-naming.xml",
"rulesets/java/ali-oop.xml",
"rulesets/java/ali-orm.xml",
"rulesets/java/ali-other.xml",
"rulesets/java/ali-set.xml"
]
}

dependencies {
// 版本号来自 https://mvnrepository.com/artifact/com.alibaba.p3c/p3c-pmd
pmd 'com.alibaba.p3c:p3c-pmd:2.1.1'
}
```

全量检查

```
$ ./gradlew check

> Task :pmdMain FAILED
DemoApplication.java:7: 【DemoApplication】缺少包含@author的注释信息
DemoApplication.java:9: 请不要使用行尾注释
```

Markdown

最近更新时间：2022-04-02 09:27:30

本文为您介绍如何通过持续集成检查 Markdown 规范。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

Markdown 常用代码规范检查工具如下：

工具	用途	lint 命令	支持 IDE
remark	Markdown 通用规范	remark -f docs/	VSCode
lint-md	Markdown 中文规范	lint-md docs/	VSCode
fnlint	文件名 slug 规范	fnlint -c .fnlint.json	不支持

remark

remark 是国际流行的开源项目，用于检查 Markdown 书写规范，内置推荐规范 remark-preset-lint-recommended，常见规则如下：

- 文本文件结尾需要换行符（POSIX 规范）。
- 列表：顶格，符号后3个空格。

安装：

```
$ npm install -save-dev remark-cli remark-preset-lint-recommended
$ vi package.json
"scripts": {
"lint-all": "remark -f docs/"
},
"remarkConfig": {
"plugins": ["remark-preset-lint-recommended"]
}
```

全量检查：

```
$ npx remark -f docs/
docs/start/ci.md: no issues found
docs/open/start.md
1:1 warning Missing newline character at end of file
```

```
16:4 warning Incorrect list-item indent: add 1 space
22:3 warning Incorrect list-item indent: add 2 spaces
⚠ 2422 warnings
```

本地增量检查：

```
git diff --diff-filter=d --name-only HEAD | grep ".md$" | xargs npx remark -f
```

持续集成合并请求增量检查：

```
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... | grep '.md\$' | xargs npx remark -f"
```

lint-md

lint-md 用于检查 Markdown 中文书写规范，内置 技术文档规范，常见规则如下：

- 中文与英文之间需要空格。
- 中文与数字之间需要空格。
- 代码块需声明编程语言（请参见：Markdown 编程语言清单）。
- 禁止使用全角数字。

安装：

```
$ npm install -save-dev lint-md-cli
```

全量检查：

```
$ lint-md docs/
docs/project/open/webhook.md
51:1-51:4 no-empty-code-lang Language of code can not be empty.
74:1-74:123 no-long-code Code Block cannot have too long line. line with 122 characters exceeds code max length 100
docs/practice/agile-development.md
52:27-52:28 space-round-alphabet No space between Chinese and alphabet. '后，Scrum负责人就会
docs/project/basis/settings.md
75:29-75:30 space-round-number No space between Chinese and number. '的缺陷最多显示10条，可
Lint total 247 files, 0 warnings 782 errors
$ echo $?
1
```

本地增量检查：

```
git diff --diff-filter=d --name-only HEAD | grep ".md$" | xargs npx lint-md
```

持续集成合并请求增量检查：

```
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... | grep '.md\$' | xargs npx lint-md"
```

fnlint

fnlint 用于检查文件名规范，内置 4 种规范：

- kebabcase（如 kebab-case.md）
- camelcase（如 camelCase.js）
- pascalcase（如 PascalCase.js）
- snakecase（如 snake_case.rb）

其中 kebabcase 适合用于检查 Markdown 文档的文件名，因为文件名将出现在 URL 中，按照 slug 和域名规范，应使用全小写、连字符，禁止使用大写字母、下划线。

安装：

```
$ npm install --save-dev fnlint
$ vi .fnlint.json
{
"basePath": ".",
"files": "docs/**/*.md",
"format": "kebabcase",
"directories": true
}
```

全量检查：

```
$ npx fnlint -c .fnlint.json
kebabcase: 9 of 257 file(s) failed linting
/docs/coding.net.md
/docs/best-practices/ci/1minute/#C+makefile.md
/docs/cd/question/FAQ.md
/docs/repo/git/commands.md
/docs/ZZZ/aaa/a.md
```

增量检查：不支持。

PHP

最近更新时间：2022-03-25 15:17:17

本文为您介绍如何通过持续集成检查 PHP 规范。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的  ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

PHP 常用代码规范检查工具如下：

工具	lint 命令	支持 IDE
CodeSniffer	phpcs --standard=PSR12 app/	VSCode、IDEA
PHPMD	phpmd app/	VSCode
PHPStan	phpstan analyse app tests	VSCode

CodeSniffer

CodeSniffer 内置多种规范，其中常用的 PSR12 规范包括下列规则：

- 每行代码最大长度120个字符。
- 运算符左右各1个空格。

安装：

```
$ composer require --dev squizlabs/php_codesniffer
```

全量检查：

```
$ ./vendor/bin/phpcs --standard=PSR12 src/

FILE: laravel-wechat/src/ServiceProvider.php
----------------------------------------------------------------------
FOUND 30 ERRORS AND 1 WARNING AFFECTING 12 LINES
----------------------------------------------------------------------
43 | ERROR | [x] Expected at least 1 space before "."; 0 found
43 | ERROR | [x] Expected at least 1 space after "."; 0 found

91 | WARNING | [ ] Line exceeds 120 characters; contains 130 characters
```

```
101 | ERROR | [x] Expected at least 1 space before "."; 0 found
101 | ERROR | [x] Expected at least 1 space after "."; 0 found
```

本地增量检查：

```
git diff --diff-filter=d --name-only HEAD | xargs ./vendor/bin/phpcs --extensions=php --standard=PSR12
```

持续集成合并请求增量检查：

```
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... | xargs ./vendor/bin/phpcs --extensions=php --standa
rd=PSR12"
```

shell

最近更新时间：2022-03-25 15:17:22

本文为您介绍如何通过持续集成检查 Shell 规范。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的  ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

Shell 常用代码规范检查工具如下：

工具	lint 命令	支持 IDE
ShellCheck	shellcheck foo.sh	VSCode、Jetbrains 内置
Shfmt	shfmt -d -i 4 -sr foo.sh	Jetbrains 内置

ShellCheck

ShellCheck 内置规范包括下列规则：

- 子语句标志符。
- 禁止全角引号。
- Useless cat。
- read lines rather than words。

安装：

```
apt-get update
apt-get install shellcheck
```

或下载：

```
wget -nc "https://coding-public-generic.pkg.coding.net/public/downloads/shellcheck-linux-x86-64.tar.xz?version=v0.7.2" -O shellcheck-v0.7.2.linux.x86_64.tar.xz
tar -C /usr/local/bin/ --strip-components=1 -Jxvf shellcheck-v0.7.2.linux.x86_64.tar.xz shellcheck-v0.7.2/shellcheck
```

全量检查：

```
shellcheck foo.sh
```

```
coding-download-center$ shellcheck girl.sh

In girl.sh line 6:
cat index-body.md | sort > index-body-sorted.md
    ^-----------^ SC2002: Useless cat. Consider 'cmd < file | ..' or 'cmd file | ..' instead.


In girl.sh line 9:
for i in `awk '{print $1}' /tmp/index-body.md`
         ^-- SC2013: To read lines rather than words, pipe/redirect to a 'while read' loop.
         ^-- SC2006: Use $(...) notation instead of legacy backticked `...`.

Did you mean:
for i in $(awk '{print $1}' /tmp/index-body.md)


In girl.sh line 11:
  [[ "$i" =~ ^[a-z0-9.-]+$ ]]  || ( echo “$i: Product names shall be all in lower case” && exit 250 )
                                       ^-- SC1110: This is a unicode quote. Delete and retype it (or quo
                                        ^-- SC2086: Double quote to prevent globbing and word splitting.
                                                                                    ^-- SC1110: This is
r quote to make literal).

Did you mean:
  [[ "$i" =~ ^[a-z0-9.-]+$ ]]  || ( echo “"$i": Product names shall be all in lower case” && exit 250 )
```

本地增量检查：

```
git diff --diff-filter=d --name-only HEAD | grep '.sh$' | xargs shellcheck
```

持续集成合并请求增量检查：

```
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... | grep '.sh\$' | xargs shellcheck"
```

Shfmt

Shfmt 内置规范包括下列规则：

- **for/do 应位于同一行。**
- **子语句标志符。**
- **行缩进：默认 tab，可自定义几个空格。**
- **行内缩进：一个空格。**
- **重定向后的空格：默认无空格，可自定义。**

安装：

```
wget -nc "https://coding-public-generic.pkg.coding.net/public/downloads/shfmt-linux-amd64?version=v3.3.1" -O /usr/local/bin/shfmt
chmod +x /usr/local/bin/shfmt
```

全量检查：

```
shfmt -d -i 4 -sr foo.sh
```

```
coding-download-center$ shfmt -d -i 4 -sr foo.sh
--- foo.sh.orig
+++ foo.sh
@@ -6,7 +6,6 @@
 cat index-body.md | sort > index-body-sorted.md
 diff index-body.md index-body-sorted.md

-for i in $(awk '{print $1}' /tmp/index-body.md)
-do
-    [[ "$i" =~ ^[a-z0-9.-]+$ ]] || ( echo “$i: Product names shall be all in lower case” && exit 250 )
+for i in $(awk '{print $1}' /tmp/index-body.md); do
+    [[ "$i" =~ ^[a-z0-9.-]+$ ]] || (echo “$i: Product names shall be all in lower case” && exit 250)
 done
```

本地增量检查：

```
git diff --diff-filter=d --name-only HEAD | grep '.sh$' | xargs shfmt -d -i 4 -sr
```

持续集成合并请求增量检查：

```
sh "git diff --diff-filter=d --name-only ${env.MR_TARGET_BRANCH}... | grep '.sh\$' | xargs shfmt -d -i 4 -sr"
```

自动化测试
功能介绍

最近更新时间：2022-03-25 15:17:28

本文为您介绍如何通过持续集成执行自动化测试任务。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

建议守则

- 开发写测试：测试代码应由开发人员编写，最好先写测试，后写业务即测试驱动开发（TDD），避免代码不可测。
- 一个代码库：测试代码和业务代码放在同一个代码库，使用同一种编程语言，一起提交。
- 自动运行：在合并请求时自动运行测试，全部通过才允许合并。
- 检查覆盖率：在合并请求时计算覆盖率，达到要求才允许合并（50%为中等，80%为良好，90%为优秀）。

工具

各个语言都有开源的测试工具、覆盖率报告工具。持续集成任务结束后支持输出运行结果。

语言	测试工具	覆盖率工具
Java	JUnit	JaCoCo
PHP	PHPUnit	PHPUnit
JS	Jest	Jest

运行持续集成任务后将自动生成测试报告。


CODING Workshop ▾ > 持续集成 / build-laravel-api / #13

构建记录#13

构建成功 sinkcup 推送到分支 main 时触发

feat: user xp
sinkcup 提交于 10 天前

构建过程 | 构建快照 | 改动记录 | 测试报告 | 通用报告

Junit 测试报告

通过的测试用例 13 条

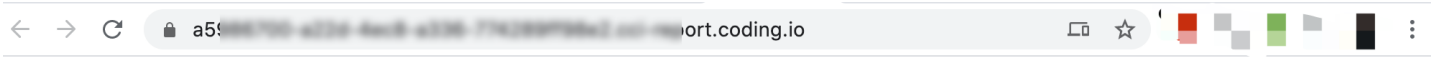
测试报告详情

失败(0) | 成功(13) | 跳过(0) | 全部(13)

testBasicTest — Tests.Feature.ExampleTest

testCreateToken — Tests.Feature.SanctumControllerTest

报告详情：



/root/workspace/app / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		58.18%	64 / 110		56.76%	21 / 37		50.00%	12 / 24
Console		75.00%	3 / 4		50.00%	1 / 2		0.00%	0 / 1
Exceptions		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Http		53.03%	35 / 66		50.00%	12 / 24		40.00%	6 / 15
Models		38.89%	7 / 18		33.33%	1 / 3		50.00%	1 / 2
Providers		85.71%	18 / 21		85.71%	6 / 7		80.00%	4 / 5

Legend

Low: 0% to 50% **Medium**: 50% to 90% **High**: 90% to 100%

Generated by php-code-coverage 9.1.8 using PHP 7.3.12 with Xdebug 2.8.1 and PHPUnit 9.3.9 at Fri Oct 30 10:12:03 UTC 2020.

Java Spring Boot

最近更新时间：2022-03-25 15:17:33

本文为您介绍如何通过持续集成使用 Jva Spring Boot 服务。
Spring Boot 2.2版本开始引入 JUnit 5作为默认的单元测试组件，适用于 Java 8及更高版本。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

安装

修改项目中的 build.gradle 文件，引入 JaCoCo 测试覆盖率工具，并配置**行覆盖率**需达到80%：

```
plugins {
id 'org.springframework.boot' version '2.3.4.RELEASE'
id 'io.spring.dependency-management' version '1.0.10.RELEASE'
id 'java'
id 'jacoco'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
mavenCentral()
}

dependencies {
implementation 'org.springframework.boot:spring-boot-starter'
testImplementation('org.springframework.boot:spring-boot-starter-test') {
exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
}
}

test {
useJUnitPlatform()
finalizedBy jacocoTestCoverageVerification
}
```

```
jacocoTestCoverageVerification {
violationRules {
rule {
limit {
counter = 'LINE'
value = 'COVEREDRATIO'
minimum = 0.8
}
}
}
dependsOn jacocoTestReport
}

jacocoTestReport {
dependsOn test
}
```

编写测试代码

按照 Spring Boot 官方文档，编写代码。
业务代码（ src/main/java/com/example/demo/HomeController.java **）：**

```
package com.example.springboot;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

@RequestMapping("/")
public String index() {
return "Greetings from Spring Boot!";
}

}
```

单元测试代码（ src/test/java/com/example/springboot/HelloControllerTest.java **）：**

```
package com.example.springboot;

import static org.hamcrest.Matchers.equalTo;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.junit.jupiter.api.Test;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

@SpringBootTest
@AutoConfigureMockMvc
public class HelloControllerTest {

@Autowired
private MockMvc mvc;

@Test
public void getHello() throws Exception {
mvc.perform(MockMvcRequestBuilders.get("/").accept(MediaType.APPLICATION_JSON))
.andExpect(status().isOk())
.andExpect(content().string(equalTo("Greetings from Spring Boot!")));
}
}
```

集成测试代码（ src/test/java/com/example/springboot/HelloControllerIT.java **）：**

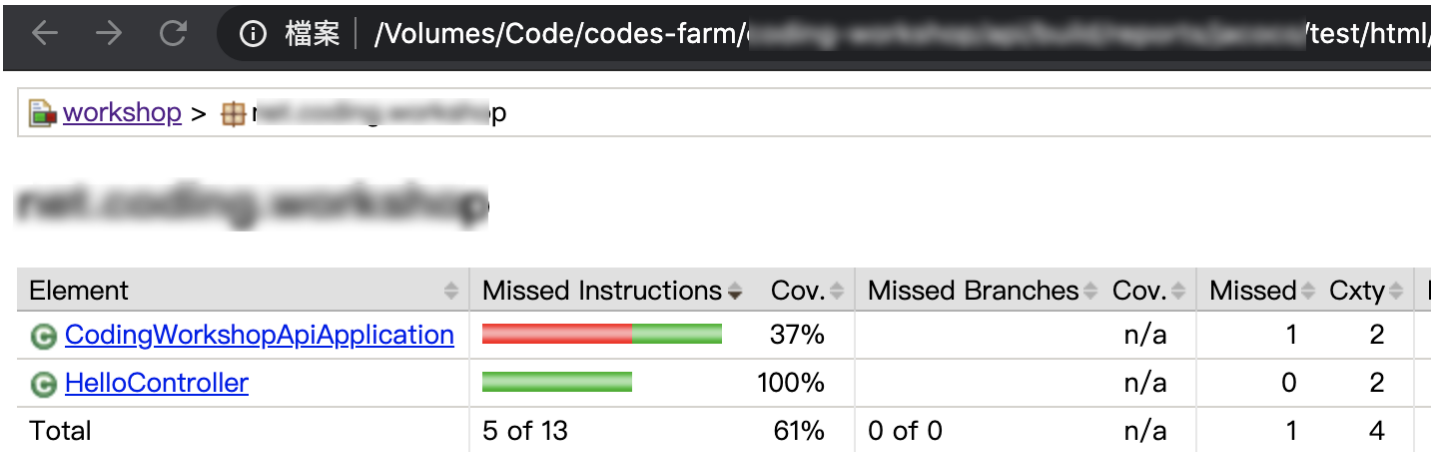
```
package com.example.springboot;

import static org.assertj.core.api.Assertions.*;

import java.net.URL;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.web.server.LocalServerPort;
import org.springframework.http.ResponseEntity;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerIT {

@LocalServerPort
private int port;

private URL base;
```

```
@Autowired
private TestRestTemplate template;

@BeforeEach
public void setUp() throws Exception {
this.base = new URL("http://localhost:" + port + "/");
}

@Test
public void getHello() throws Exception {
ResponseEntity<String> response = template.getForEntity(base.toString(),
String.class);
assertThat(response.getBody()).isEqualTo("Greetings from Spring Boot!");
}
}
```

本地运行

本地运行测试，将会在 build/reports/jacoco/test/html/ **生成 HTML 格式的覆盖率报告。**
如果行覆盖率不达标，将会报错退出。

```
$ ./gradlew clean test

> Task :test
2021-04-06 16:08:03.346 INFO 13478 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down Ex
ecutorService 'applicationTaskExecutor'

> Task :jacocoTestCoverageVerification FAILED
[ant:jacocoReport] Rule violated for bundle workshop: lines covered ratio is 0.6, but expected minimum is 0.8

FAILURE: Build failed with an exception.

$ open build/reports/jacoco/test/html/index.html
```



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty
CodingWorkshopApiApplication		37%		n/a	1	2
HelloController		100%		n/a	0	2
Total	5 of 13	61%	0 of 0	n/a	1	4

持续集成

```
pipeline {
agent any
stages {
stage('检出') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: '*']],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
}
}
stage('测试') {
agent {
docker {
image 'adoptopenjdk:11-jdk-hotspot'
args '-v /root/.gradle/:/root/.gradle/ -v /root/.m2/:/root/.m2/'
reuseNode true
}
}
steps {
sh './gradlew test'
}
post {
// 不管成功失败，都收集简易测试结果
always {
junit 'build/test-results/**/*.xml'
}
// 成功时，才收集测试覆盖率报告
success {
codingHtmlReport(name: '测试覆盖率报告', tag: 'test', path: 'build/reports/jacoco/test/html', entryFile: 'index.html')
}
}
}
}
}
```

PHP 自动化测试

最近更新时间：2022-03-28 10:57:32

本文为您介绍如何通过持续集成执行 PHP 自动化测试任务。
Laravel 8框架使用 PHPUnit 9 作为默认的单元测试组件，适用于 PHP 7.3及更高版本。

PHPUnit 可生成 HTML、Clover 等格式的测试覆盖率报告，JUnit 等格式的测试结果，但 不支持检查测试覆盖率大小，可通过 phpunit-coverage-check 进行检查。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的  ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

安装

在代码目录执行命令进行安装：

```
$ composer require --dev phpunit/phpunit
$ composer require --dev rregeer/phpunit-coverage-check
```

编写测试代码

按照 Laravel 官方文档，编写测试代码。
业务代码（ app/Http/Controllers/Welcome.php ）：

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class Welcome extends Controller
{
/**
* Handle the incoming request.
*
* @param \Illuminate\Http\Request $request
* @return \Illuminate\Http\Response
*/
public function __invoke(Request $request)
{
return view('welcome');
```

```
}
}
```

路由（ routes/web.php ）：

```
use Illuminate\Support\Facades\Route;

Route::get('/', \App\Http\Controllers\Welcome::class);
```

测试代码（ app/Http/Controllers/Welcome.php ）：

```
namespace Tests\Feature;

use Tests\TestCase;

class ExampleTest extends TestCase
{
/**
* A basic test example.
*
* @return void
*/
public function testBasicTest()
{
$response = $this->get('/');

$response->assertStatus(200);
}
}
```

本地运行

本地运行测试，将会生成 JUnit 格式测试结果和 HTML 和 Clover 格式覆盖率报告。

```
$ ./vendor/bin/phpunit --coverage-html storage/reports/tests/ --coverage-clover storage/reports/tests/clover.xml --log-junit storage/test-results/junit.xml tests/

$ head storage/test-results/junit.xml
$ open storage/reports/tests/index.html
```

根据 Clover 报告检查测试覆盖率，如果不达标，将会报错退出。

```
$ ./vendor/bin/coverage-check storage/reports/tests/clover.xml 80
Total code coverage is 53.85 % which is below the accepted 80%
```

```
$ echo $?
1
```

持续集成

带数据库的测试

自动化测试需要临时的基础设施（例如：MySQL、Redis、Elasticsearch），无需购买，启动多个 Docker 后台即可，测试完毕自动删除。

```
node {
stage("检出") {
checkout([
$class: 'GitSCM',
branches: [[name: GIT_BUILD_REF]],
userRemoteConfigs: [[
url: GIT_REPO_URL,
credentialsId: CREDENTIALS_ID
]]])
}
stage('准备数据库') {
sh 'docker network create bridge1'
sh(script:'docker run --net bridge1 --name mysql -d -e "MYSQL_ROOT_PASSWORD=my-secret-pw" -e "MYSQL_DATABASE=test_db" mysql:5.7', returnStdout: true)
sh(script:'docker run --net bridge1 --name redis -d redis:5', returnStdout: true)
}
docker.image('ecoding/php:8.0').inside("--net bridge1 -v \"${env.WORKSPACE}:/root/code\" -e 'APP_ENV=testing' -e 'DB_DATABASE=test_db'" +
" -e 'DB_USERNAME=root' -e 'DB_PASSWORD=my-secret-pw' -e 'DB_HOST=mysql' -e 'REDIS_HOST=redis'" +
" -e 'APP_KEY=base64:tbgOBtYci7i7cdx5RiFE3KZzUkRtJfbU3lbj5uPdL8U='") {
sh 'composer install'

stage('单元测试') {
sh 'XDEBUG_MODE=coverage ./vendor/bin/phpunit --coverage-html storage/reports/tests/ --log-junit storage/test-results/junit.xml --coverage-text tests/'
junit 'storage/test-results/junit.xml'
codingHtmlReport(name: '测试覆盖率报告', path: 'storage/reports/tests/')
}

stage('生成 API 文档') {
sh 'php artisan l5-swagger:generate'
codingReleaseApiDoc(apiDocId: '1', apiDocType: 'specificFile', resultFile: 'storage/api-docs/api-docs.json')
}
}

CODING_DOCKER_REG_HOST = "${CCI_CURRENT_TEAM}-docker.pkg.${CCI_CURRENT_DOMAIN}"
CODING_DOCKER_IMAGE_NAME = "${env.PROJECT_NAME.toLowerCase()}/laravel-docker/laravel-demo"
stage('构建 Docker 镜像') {
if (env.TAG_NAME ==~ /.*/ ) {
```

```
DOCKER_IMAGE_VERSION = "${env.TAG_NAME}"
} else if (env.MR_SOURCE_BRANCH ==~ /.*/ ) {
DOCKER_IMAGE_VERSION = "mr-${env.MR_RESOURCE_ID}-${env.GIT_COMMIT_SHORT}"
} else {
DOCKER_IMAGE_VERSION = "${env.BRANCH_NAME.replace('/', '-')}-${env.GIT_COMMIT_SHORT}"
}
// 本项目内的制品库已内置环境变量 CODING_ARTIFACTS_CREDENTIALS_ID，无需手动设置
docker.withRegistry("https://${env.CCI_CURRENT_TEAM}-docker.pkg.coding.net", "${env.CODING_ARTIFACTS_CREDENTIALS_ID}") {
docker.build("${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION}").push()
}
}
}
```

不带数据库的测试

```
pipeline {
agent {
docker {
image 'ecoding/php:8.0'
reuseNode 'true'
args '-v /var/run/docker.sock:/var/run/docker.sock -v /usr/bin/docker:/usr/bin/docker'
}
}
stages {
stage("检出") {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: GIT_BUILD_REF]],
userRemoteConfigs: [[
url: GIT_REPO_URL,
credentialsId: CREDENTIALS_ID
]]])
}
}
stage('准备依赖') {
steps {
sh 'composer install'
}
}
stage('单元测试') {
post {
always {
junit 'storage/test-results/junit.xml'
}
success {
```

```
codingHtmlReport(name: '测试覆盖率报告', path: 'storage/reports/tests/')
}
}
steps {
sh 'touch database/database.sqlite'
sh 'XDEBUG_MODE=coverage ./vendor/bin/phpunit --coverage-html storage/reports/tests/ --log-junit storage/test-results/junit.xml --coverage-text tests/'
}
}
stage('生成 API 文档') {
steps {
sh 'php artisan l5-swagger:generate'
codingReleaseApiDoc(apiDocId: '1', apiDocType: 'specificFile', resultFile: 'storage/api-docs/api-docs.json')
}
}
stage('构建 Docker 镜像') {
steps {
script {
if (env.TAG_NAME ==~ /.*/ ) {
DOCKER_IMAGE_VERSION = "${env.TAG_NAME}"
} else if (env.MR_SOURCE_BRANCH ==~ /.*/ ) {
DOCKER_IMAGE_VERSION = "mr-${env.MR_RESOURCE_ID}-${env.GIT_COMMIT_SHORT}"
} else {
DOCKER_IMAGE_VERSION = "${env.BRANCH_NAME.replace('/', '-')}-${env.GIT_COMMIT_SHORT}"
}
// 本项目内的制品库已内置环境变量 CODING_ARTIFACTS_CREDENTIALS_ID，无需手动设置
docker.withRegistry("https://${env.CCI_CURRENT_TEAM}-docker.pkg.coding.net", "${env.CODING_ARTIFACTS_CREDENTIALS_ID}") {
docker.build("${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION}").push()
}
}
}
}
}
environment {
CODING_DOCKER_REG_HOST = "${CCI_CURRENT_TEAM}-docker.pkg.${CCI_CURRENT_DOMAIN}"
CODING_DOCKER_IMAGE_NAME = "${env.PROJECT_NAME.toLowerCase()}/laravel-docker/laravel-demo"
}
}
```

运行结果

- 通过持续集成任务：

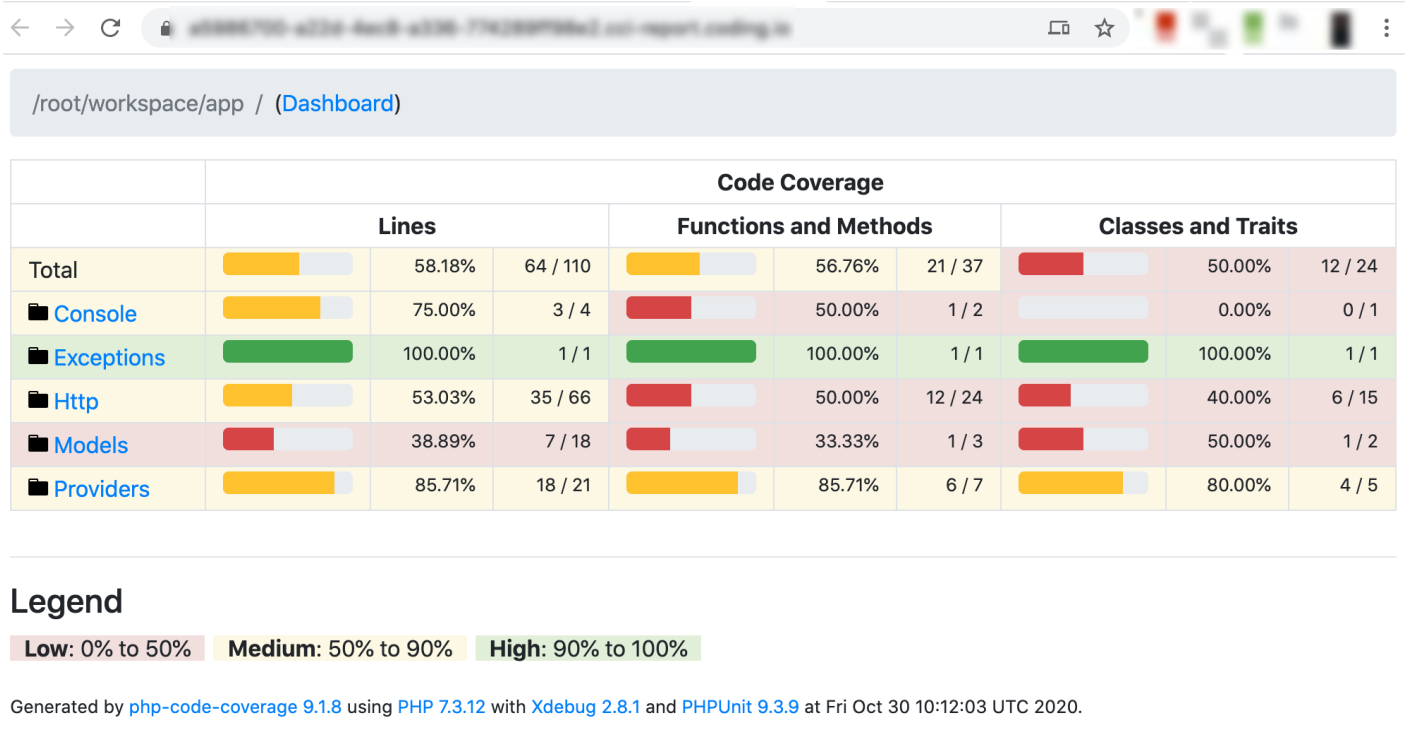

构建记录#251
构建过程
构建快照
改动记录
测试报告
通用报告
构建产物
设置
启动新构建
构建成功
手动触发
触发于 32 分钟前，持续时长 3 分钟 55 秒
laravel-demo
6.x
9193e7f
docs: fix openapi schema
重新构建
构建过程
查看完整日志
准备数据库 30 s
执行 Shell 脚本 1 s
执行 Shell 脚本 8 s
执行 Shell 脚本 2 s
单元测试 9 s
执行 Shell 脚本 1 s
收集 JUnit 测试报告 < 1 s
收集通用报告 7 s
生成 API 文档 47 s
执行 Shell 脚本 < 1 s
Release a new versi... 46 s
构建 Dock... 2 m 9 s
Checks if running o... < 1 s
执行 Shell 脚本 1 m 49 s
Checks if running o... < 1 s

- 输出测试报告：


CODING Workshop
持续集成 / build-laravel-api / #13
项目概览
项目协同
代码仓库
代码扫描 beta
持续集成
构建计划
构建节点
持续部署
制品库
测试管理
文档管理
构建记录#13
构建成功 推送到分支 main 时触发
feat: user xp
提交于 10 天前
构建过程
构建快照
改动记录
测试报告
通用报告
Junit 测试报告
100%
通过率
通过的测试用例 13 条
测试报告详情
失败(0)
成功(13)
跳过(0)
全部(13)
testBasicTest — Tests.Feature.ExampleTest
testCreateToken — Tests.Feature.SanctumControllerTest

- 输出通用报告：

- 报告详情：



/root/workspace/app / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		58.18%	64 / 110		56.76%	21 / 37		50.00%	12 / 24
Console		75.00%	3 / 4		50.00%	1 / 2		0.00%	0 / 1
Exceptions		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
Http		53.03%	35 / 66		50.00%	12 / 24		40.00%	6 / 15
Models		38.89%	7 / 18		33.33%	1 / 3		50.00%	1 / 2
Providers		85.71%	18 / 21		85.71%	6 / 7		80.00%	4 / 5

Legend

Low: 0% to 50% Medium: 50% to 90% High: 90% to 100%

Generated by php-code-coverage 9.1.8 using PHP 7.3.12 with Xdebug 2.8.1 and PHPUnit 9.3.9 at Fri Oct 30 10:12:03 UTC 2020.

自动部署
COS 存储桶

最近更新时间：2023-02-03 14:12:29

本文为您介绍如何通过持续集成将项目一键发布至 COS 存储桶。

前提条件

您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

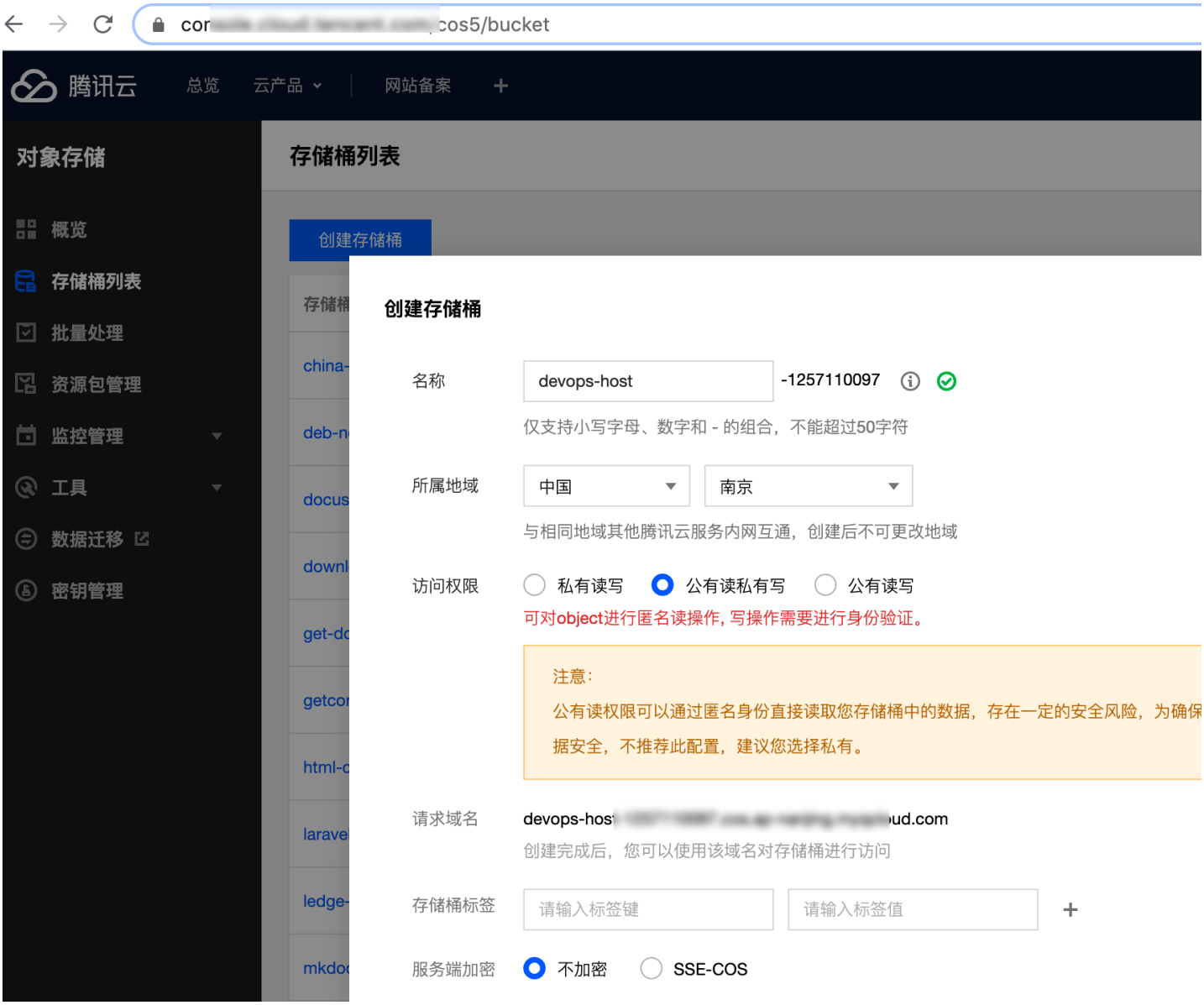
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

功能介绍

您可以将需要存储至云端的项目通过持续集成一键发布至 COS 中，适合搭建静态网站、编译文件后供下载等场景。

新建存储桶

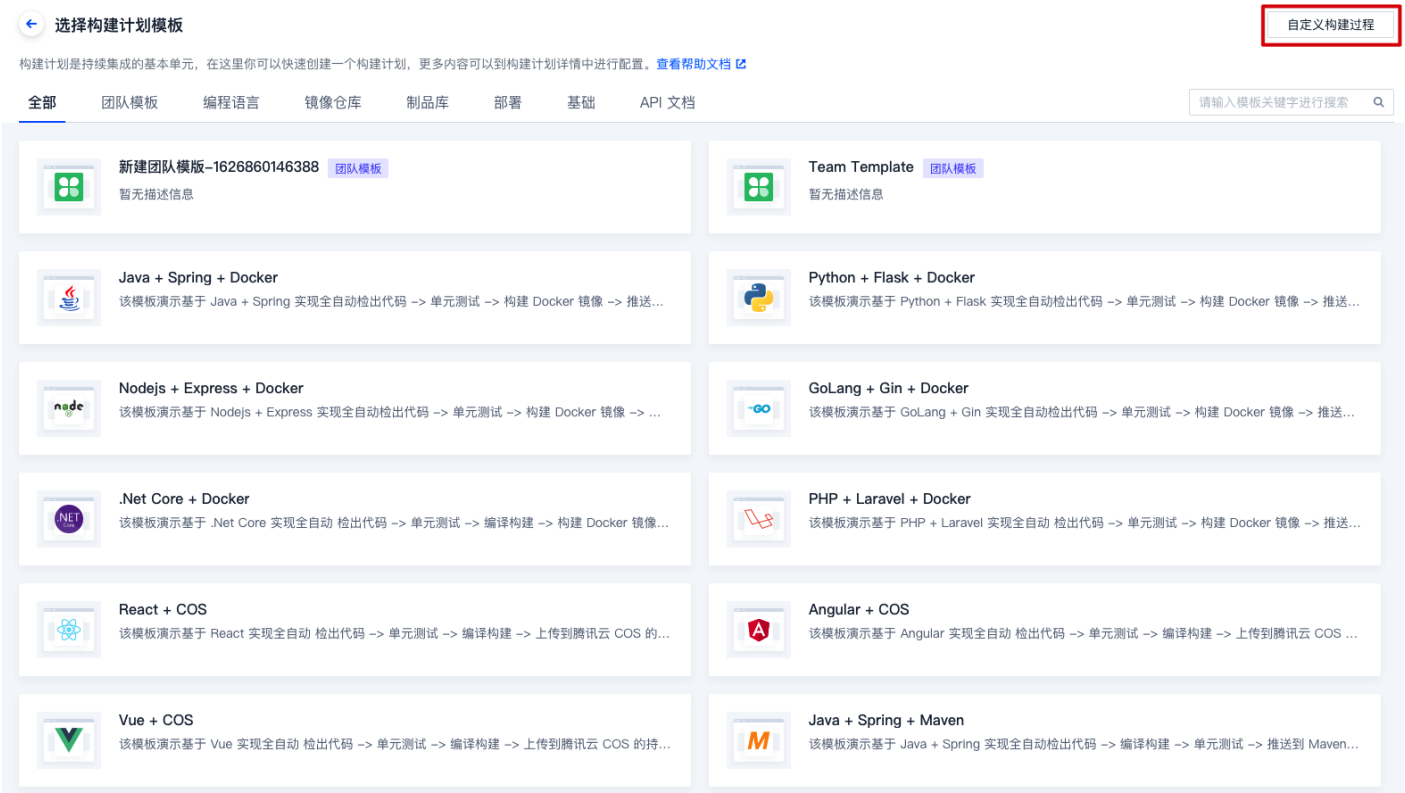
在 腾讯云 COS 对象存储 中创建一个**存储桶**，获取名称、区域、密钥等信息。



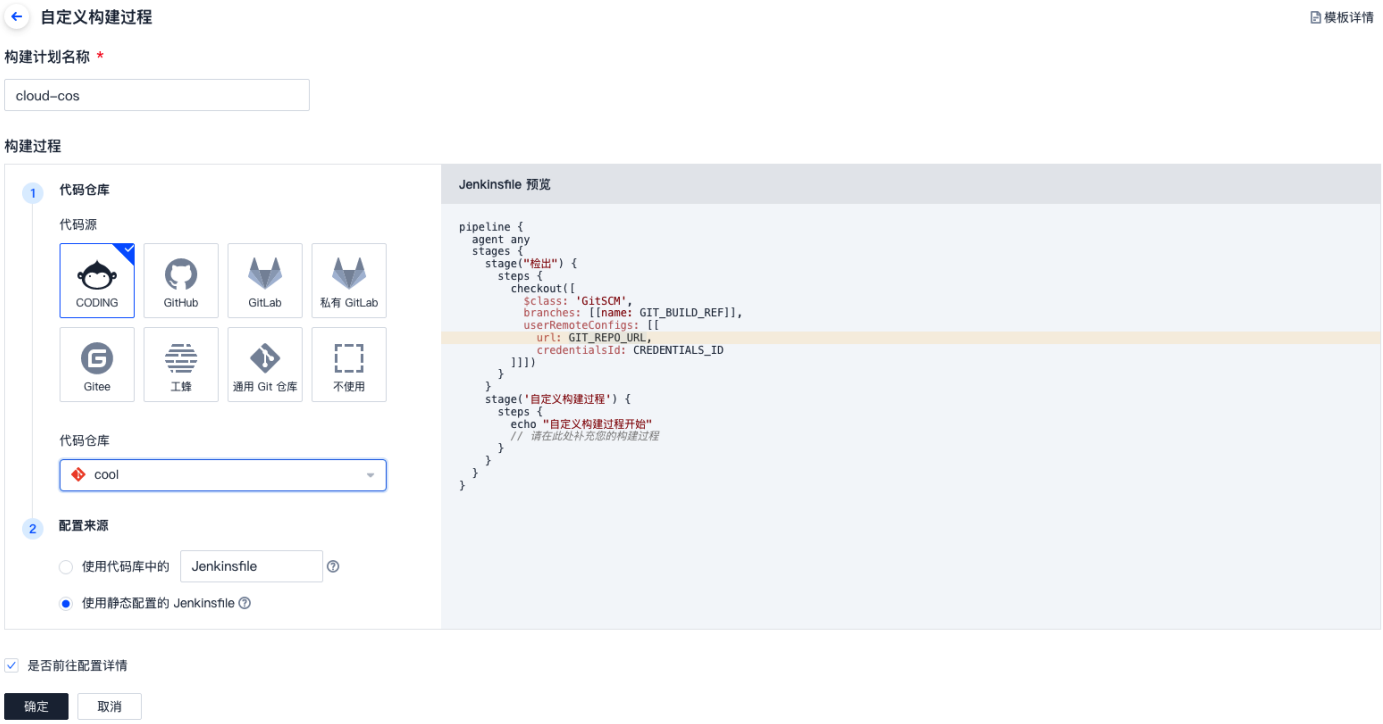
配置持续集成

1. 前往 CODING DevOps 持续集成，单击右上角的**创建构建计划**。

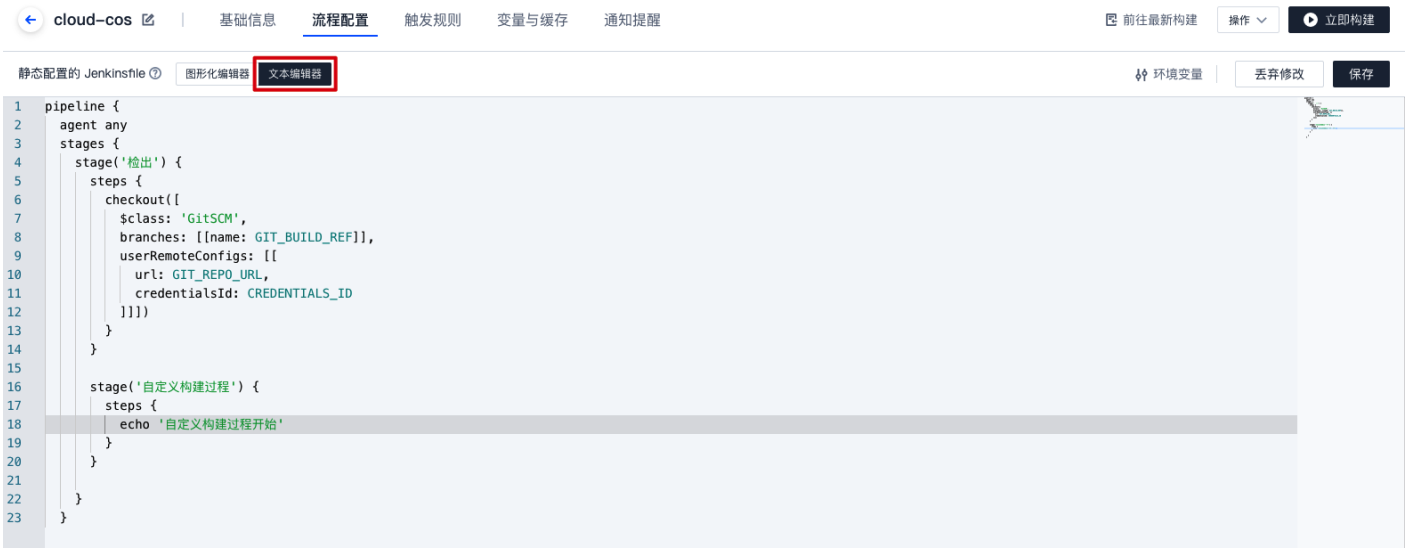
2. 单击右上角的**自定义构建过程**。



3. 选择需要上传至 COS 的代码仓库源。



4. 选择使用**文本编辑器**配置 Jenkinsfile。



Jenkinsfile

在文本编辑器中参考并写入下述 Jenkinsfile，触发构建任务后进行上传。

```
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('编译') {
steps {
// Markdown 转成 HTML
sh 'pip install mkdocs && mkdocs build'
// React/VUE SPA 生成 HTML
sh 'npm run build'
// Android 打包
sh './gradlew assembleDebug'
}
}
stage('上传到腾讯云 COS 对象存储') {
steps {
sh "coscmd config -a ${env.COS_SECRET_ID} -s ${env.COS_SECRET_KEY}" +
" -b ${env.COS_BUCKET_NAME} -r ${env.COS_BUCKET_REGION}"
sh "rm -rf .git"
sh 'coscmd upload -r ./ /'
//sh 'coscmd upload -r ./dist /'
}
}
```

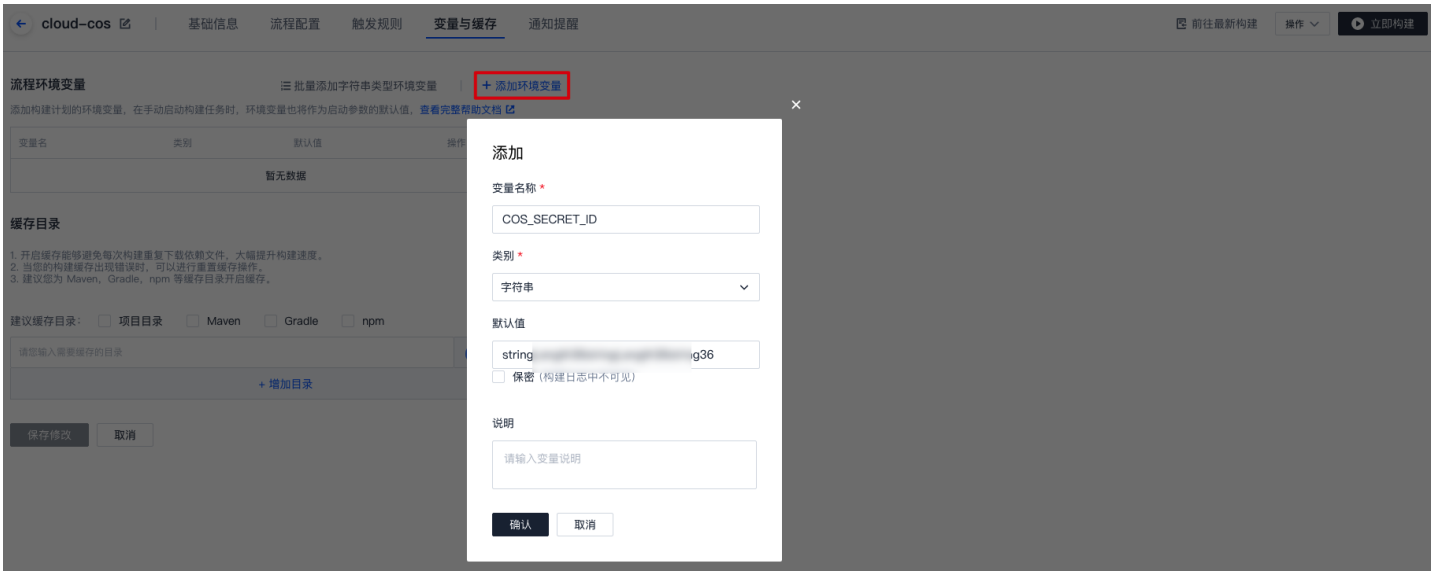
```
}
}
```

环境变量

变量名	含义	参考值
COS_SECRET_ID	腾讯云访问密钥 ID	stringLength36stringLength36string36
COS_SECRET_KEY	腾讯云访问密钥 KEY	stringLength32stringLength323232
COS_BUCKET_NAME	腾讯云对象存储桶	devops-host-1257110097
COS_BUCKET_REGION	腾讯云对象存储区域	ap-nanjing

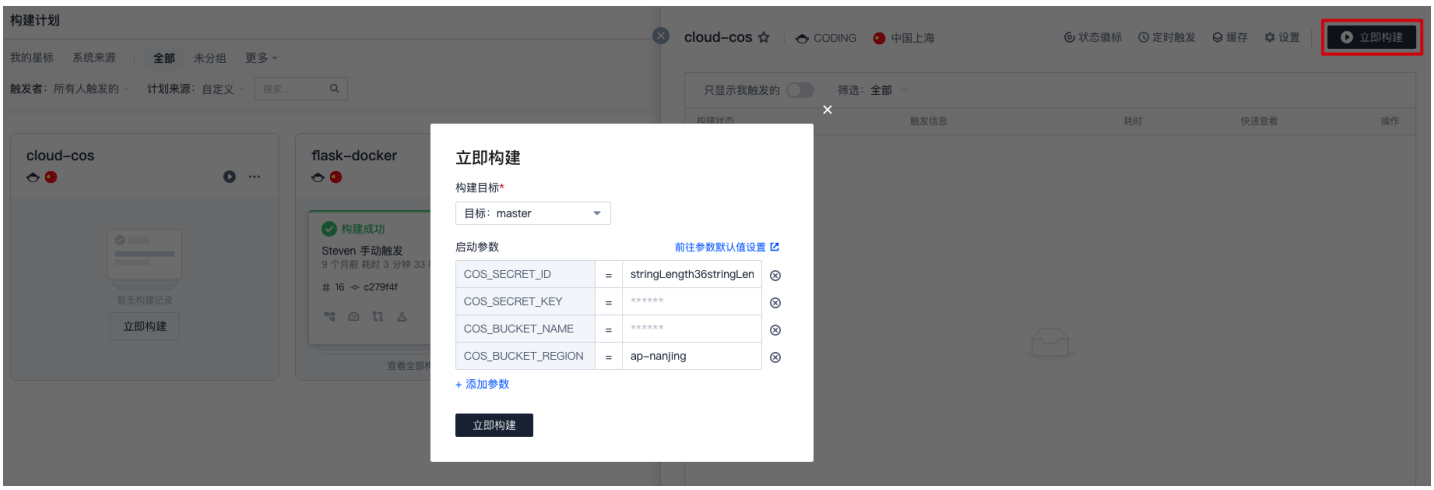
添加环境变量

访问构建计划中的**变量与缓存**，添加 COS 相关的环境变量。



触发持续集成

添加环境变量后，单击右上角的**立即构建**触发持续集成任务。



Docker 服务器

最近更新时间：2022-04-29 15:02:28

本文为您介绍如何通过持续集成将项目发布至 Docker 服务器。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

获取 SSH 密钥对

登录服务器控制台，创建 SSH 密钥对。获取私钥对后将其录入至 CODING 中的 凭据管理 中，将公钥 id_rsa.pub 的内容复制到服务器的 ~/.ssh/authorized_keys 中。

获取制品仓库信息

1. 按照提示一键获取 Docker 仓库的用户名与密码，并将其录入持续集成的环境变量中。

2. 在构建计划详情中的变量与缓存处填写。

Jenkinsfile

在构建计划设置中的流程配置中参考以下配置进行填写。

```
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 '构建中...'
```

```
script {
// 请修改 dockerServer、dockerPath、imageName
dockerServer = 'codes-farm-docker.pkg.coding.net'
dockerPath = '/laravel-demo/laravel-docker'
imageName = "${dockerServer}${dockerPath}/laravel-demo:1.0.0"
def customImage = docker.build(imageName)

// 推送 Docker 镜像到仓库
docker.withRegistry("https://${dockerServer}", CODING_ARTIFACTS_CREDENTIALS_ID) {
customImage.push()
}
}
}
}
stage('部署') {
steps {
echo '部署中...'
script {
// 声明服务器信息
def remote = [:]
remote.name = 'web-server'
remote.allowAnyHosts = true
remote.host = 'x.x.x.x'
remote.port = 22
remote.user = 'ubuntu'

// 把「CODING 凭据管理」中的「凭据 ID」填入 credentialsId，而 id_rsa 无需修改
withCredentials([sshUserPrivateKey(credentialsId: "c4af855d-xxxx-xxxx-xxxx-xxxx6ae3441c", keyFileVariable: 'id_rsa')]) {
remote.identityFile = id_rsa

// SSH 登录到服务器，拉取 Docker 镜像
// 请在持续集成的环境变量中配置 DOCKER_USER 和 DOCKER_PASSWORD
sshCommand remote: remote, sudo: true, command: "apt-get install -y gnupg2 pass"
sshCommand remote: remote, command: "docker login -u ${env.DOCKER_USER} -p ${env.DOCKER_PASSWORD} $DOCK
ER_SERVER"
sshCommand remote: remote, command: "docker pull ${imageName}"
sshCommand remote: remote, command: "docker stop web | true"
sshCommand remote: remote, command: "docker rm web | true"
sshCommand remote: remote, command: "docker run --name web -d ${imageName}"
}
}
}
}
}
}
```

Docker Compose

Docker Compose 的代码和上述类似，仅有少许不同：

```
sshCommand remote: remote, sudo: true, command: "apt-get install -y gnupg2 pass"
sshCommand remote: remote, command: "docker login -u ${env.DOCKER_USER} -p ${env.DOCKER_PASSWORD} $DOCKER_SERVER"
sshCommand remote: remote, sudo: true, command: "mkdir -p /var/www/site/"
sshCommand remote: remote, sudo: true, command: "chmod 777 /var/www/site/"
sshPut remote: remote, from: 'docker-compose.yml', into: '/var/www/site/'
sshCommand remote: remote, command: "cd /var/www/site/ && echo IMAGE=${imageName} > .env && echo APP_KEY=${env.APP_KEY} >> .env && echo DB_CONNECTION=sqlite >> .env"
sshCommand remote: remote, command: "cd /var/www/site/ && docker-compose down --remove-orphans"
sshCommand remote: remote, command: "cd /var/www/site/ && docker-compose up -d --no-build"
```

docker-compose.yml 代码：

```
version: '2.1'
services:
web:
env_file: .env
build: .
image: ${IMAGE:-laravel-demo:dev}
ports:
- "80:80"
links:
- redis
redis:
image: "redis:5"
```

K8s 集群

最近更新时间：2022-04-29 14:43:02

本文为您介绍如何通过持续集成将项目发布至 K8s 集群。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

持续集成可简易部署项目到 K8s 集群，步骤如下：

1. 获取 Docker 仓库的用户名和密码（CODING 制品库一键创建访问令牌即可获得），录入持续集成的环境变量中。
2. 构建 Docker 镜像并上传到仓库。
3. 在云计算服务商（例如腾讯云）创建一个 K8s 集群和 deployment，获得 Kubeconfig，录入 CODING 凭据管理。
4. 在持续集成中使用下述 Jenkinsfile：执行 kubectl 进行部署。

Jenkinsfile

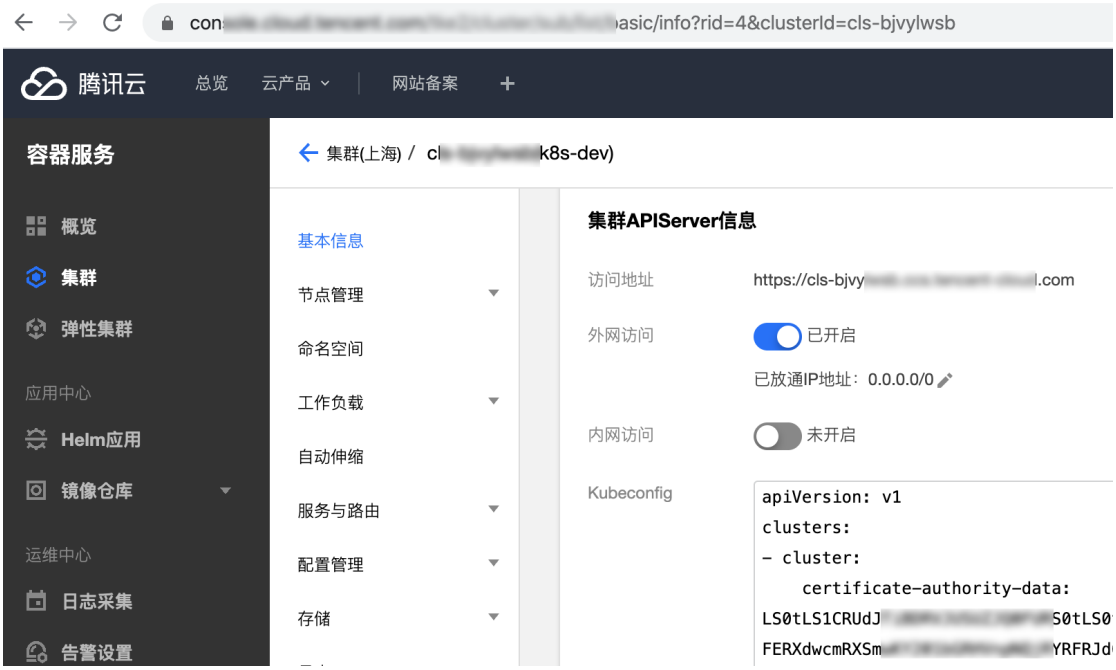
```
pipeline {
agent any
stages {
stage('检出') {
steps {
checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]])
}
}
stage('构建') {
steps {
echo '构建中...'
script {
// 请修改 dockerServer、dockerPath、imageName
dockerServer = 'codes-farm-docker.pkg.coding.net'
dockerPath = '/laravel-demo/laravel-docker'
imageName = "${dockerServer}${dockerPath}/laravel-demo:1.0.0"
def customImage = docker.build(imageName)

// 推送 Docker 镜像到仓库
docker.withRegistry("https://${dockerServer}", CODING_ARTIFACTS_CREDENTIALS_ID) {
customImage.push()
}
```

```
}
}
}
stage('部署到 K8s') {
steps {
echo '部署中...'
script {
// 请修改 credentialsId：填入 k8s 凭据 ID
withKubeConfig([credentialsId: 'f23cc59c-xxxx-xxxx-xxxx-2c9b6909e908']) {
// 使用 kubectl 创建 K8s 密钥：来自环境变量的 DOCKER_USER 和 DOCKER_PASSWORD
sh(script: "kubectl create secret docker-registry coding --docker-server=${dockerServer} --docker-username=${env.DOCKER_USER} --docker-password=${env.DOCKER_PASSWORD} --docker-email=support@coding.net", returnStatus: true)

// 使用 kubectl 修改 K8s deployment：指定 Docker 镜像链接和密钥
// 请修改 laravel-demo、web 为你的 deployment 中的值
sh "kubectl patch deployment laravel-demo --patch '{\"spec\": {\"template\": {\"spec\": {\"containers\": [{\"name\": \"web\", \"image\": \"${imageName}\"}], \"imagePullSecrets\": [{\"name\": \"coding\"}]}}}}'"
}
}
}
}
}
}
```

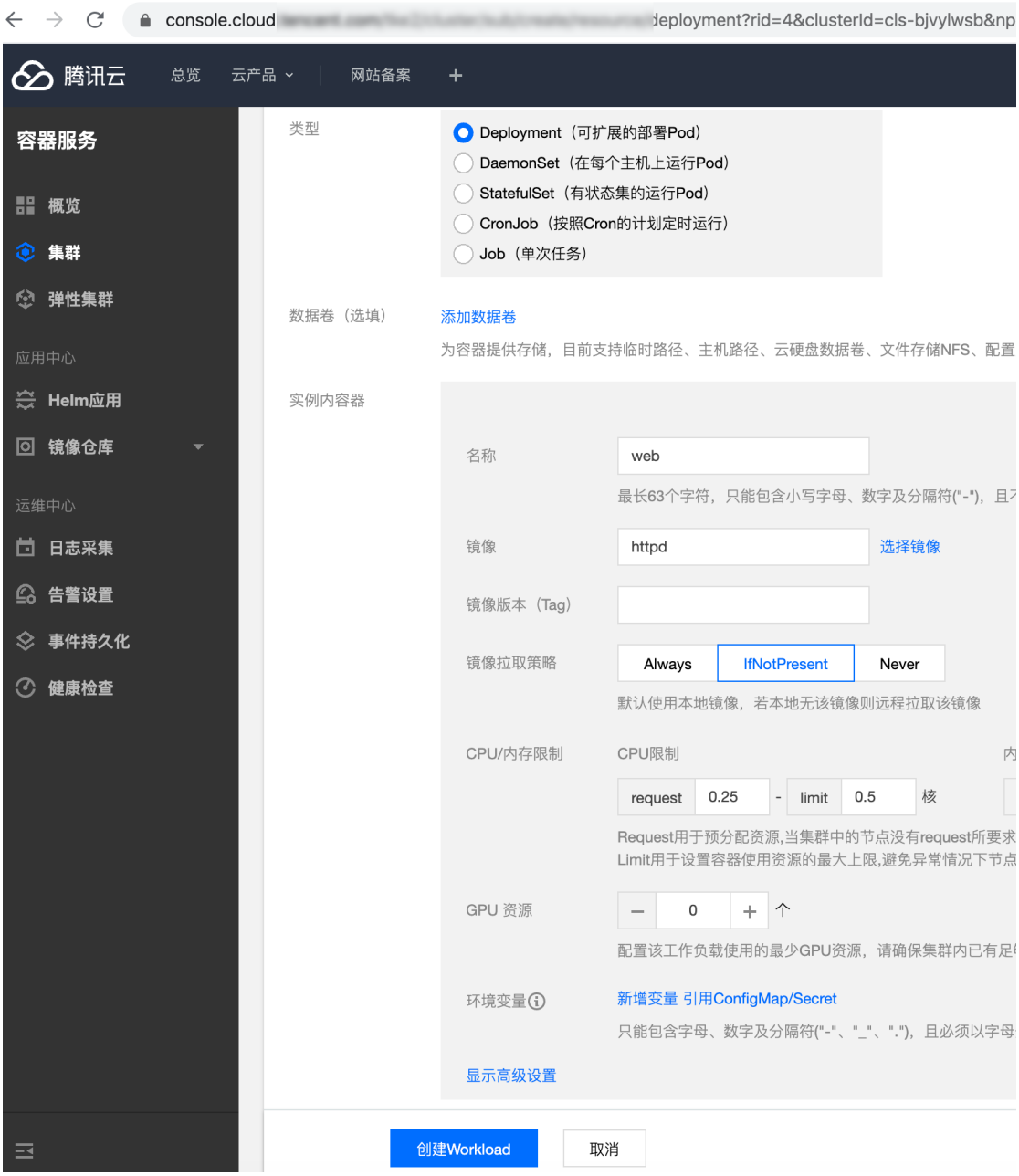
截图


制品库 / laravel-docker
搜索...
制品库
my-npm
npm 仓库 | 项目内
my-maven
Maven 仓库 | 项目内
my-composer
Composer 仓库 | 项目内
laravel-docker
Docker 仓库 | 项目内
laravel-docker
类型 Docker | 权限 项目内
指引
镜像列表
设置凭证
自动生成凭证
推荐使用访问令牌生成命令。 查看所有认证方式
使用访问令牌生成配置
1
请在命令行中执行以下命令，管理请到 个人访问令牌。
2
docker login -u laravel-docker-
-p 81
a91337


持续集成 / laravel-demo-ci / 修改配置
搜索...
laravel-demo-ci
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
+ 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_USER
字符串
laravel-docker-1
DOCKER_PASSWORD
字符串
******

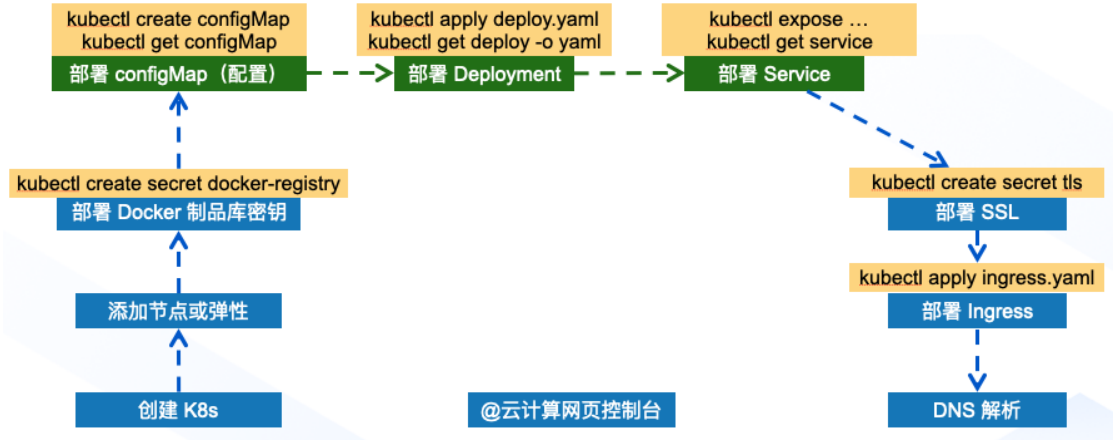

腾讯云
总览
云产品
网站备案
容器服务
概览
集群
弹性集群
应用中心
Helm应用
镜像仓库
运维中心
日志采集
告警设置
集群(上海) / cl
k8s-dev)
基本信息
节点管理
命名空间
工作负载
自动伸缩
服务与路由
配置管理
存储
集群APIServer信息
访问地址
https://cls-bjvy
.com
外网访问
已开启
已放通IP地址：0.0.0.0/0
内网访问
未开启
Kubeconfig
apiVersion: v1
clusters:
- cluster:
certificate-authority-data:
LS0tLS1CRUdJ
S0tLS0
FERXdwcmRXSm
YRFRJd


coding-public.
nection/create/KUBERNETES
MkDocs COS Demo
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
Kubernetes 凭据
凭据名称*
k8s-dev
选择认证方式*
Kubeconfig
Service Account
KubeConfig*
apiVersion: v1
clusters:
- cluster:
certificate-authority-data:
LS0tLS1C
RS0tLS0
tCk1JSUN
ICQURB



提醒

K8s 部署包括5步甚至更多，如果都写在持续集成里难以维护，建议使用 持续部署。



Serverless

最近更新时间：2022-03-25 15:18:10

本文为您介绍如何通过持续集成将项目发布至 Serverless。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

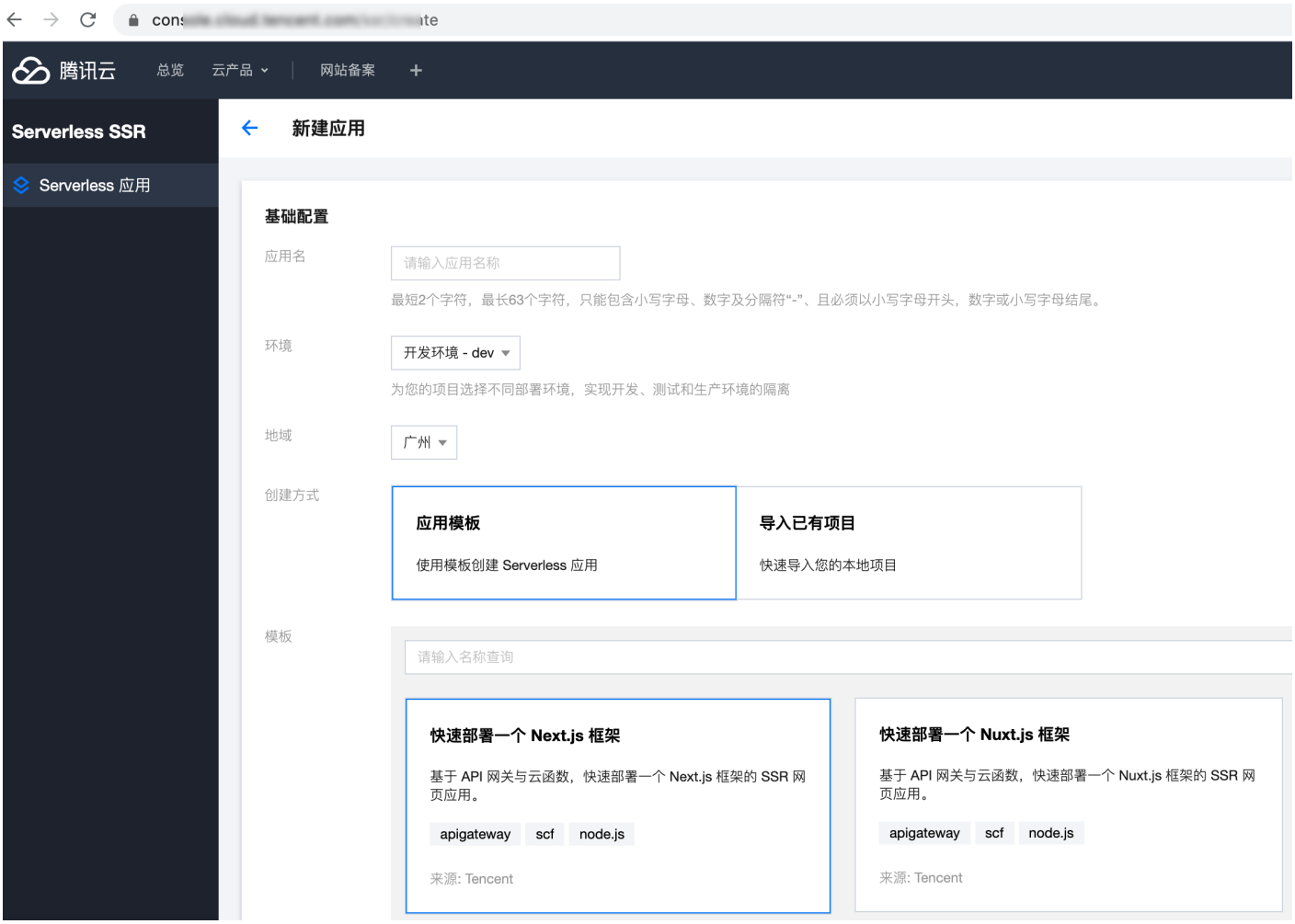
进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

功能介绍

持续集成可自动部署项目到 Serverless，适用于需要 SEO 的动态网站等场景，步骤如下：

1. 在 Serverless（例如：腾讯云 Serverless）中创建一个**应用**，获取名称、区域、密钥。



2. 在持续集成中使用下述 Jenkinsfile：执行命令进行部署。

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('编译') {
steps {
sh 'npm install'
sh 'npm run build'
}
}
stage('部署') {
steps {
sh 'npm install -g serverless'
sh 'serverless deploy'
}
}
}
}
```

持续集成环境变量

变量名	含义	参考值
SERVERLESS_PLATFORM_VENDOR	Serverless 厂商	tencent
SERVERLESS_REGION	Serverless 区域	ap-guangzhou
SERVERLESS_STAGE	Serverless 环境	dev、test、prod
TENCENT_SECRET_ID	腾讯云访问密钥 ID	AKIDFooBar
TENCENT_SECRET_KEY	腾讯云访问密钥 KEY	jgaYd123456

Linux 服务器

最近更新时间：2022-03-25 15:18:14

本文为您介绍如何通过持续集成将项目发布至 Linux 服务器。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

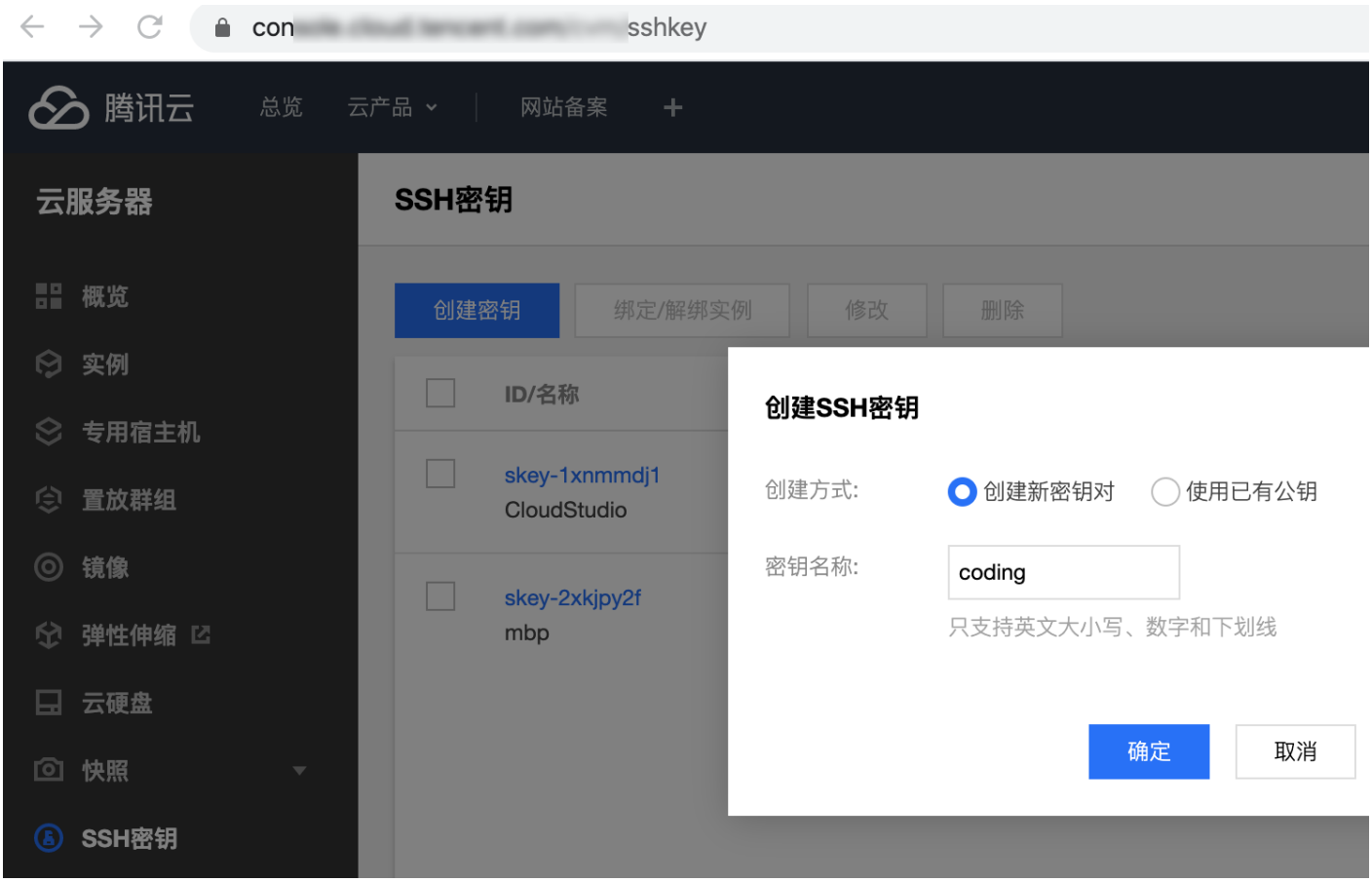
进入项目

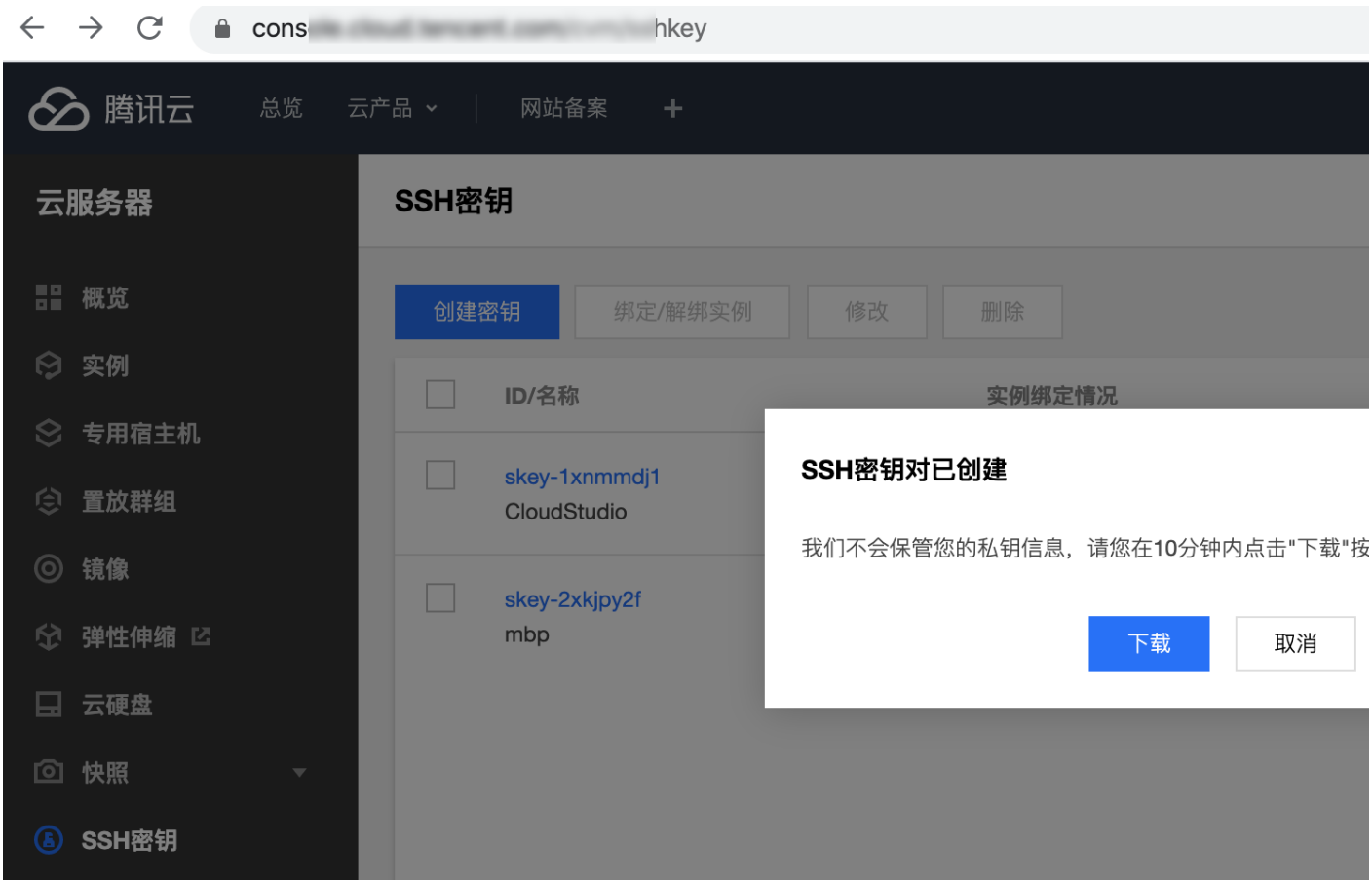
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

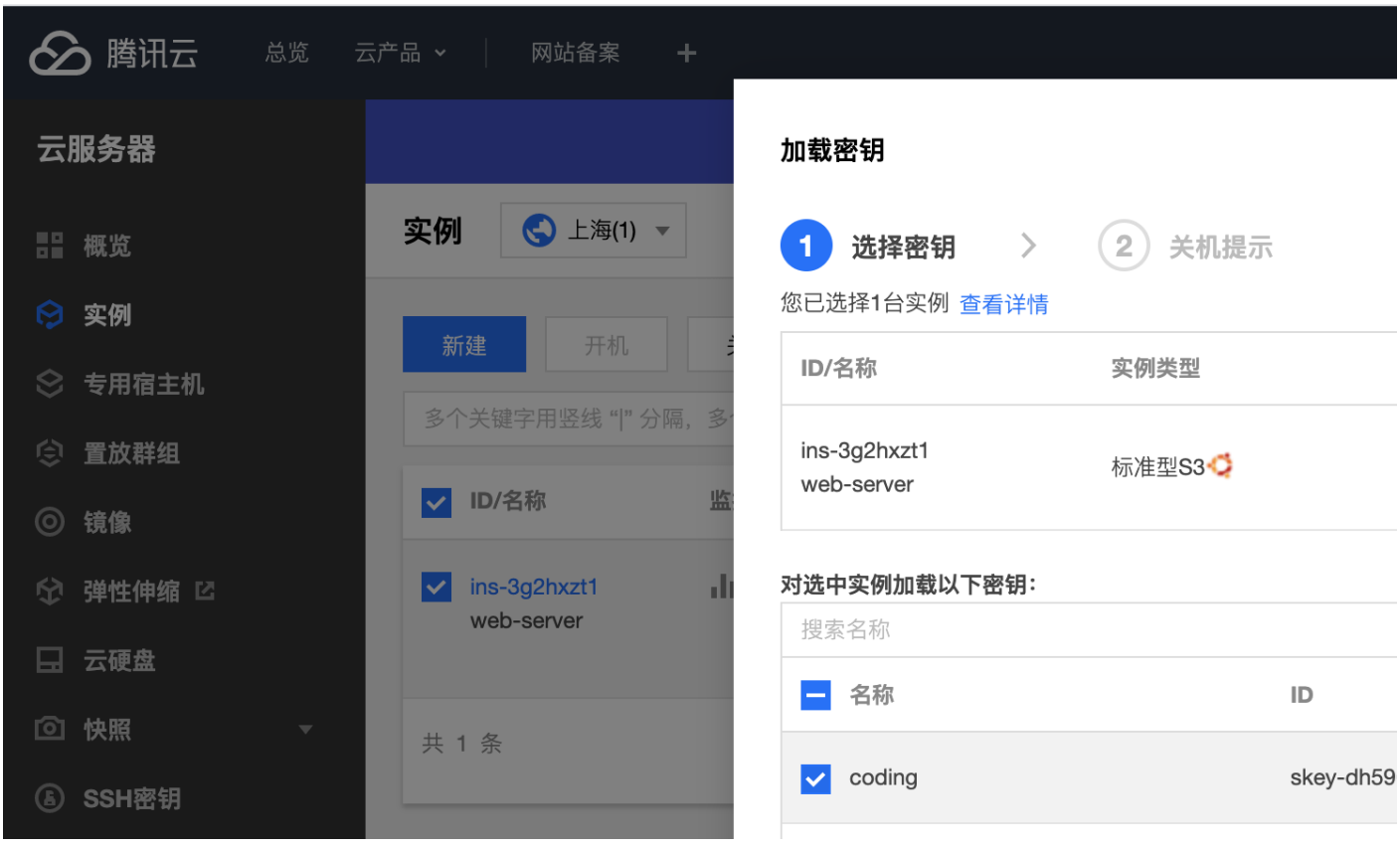
功能介绍

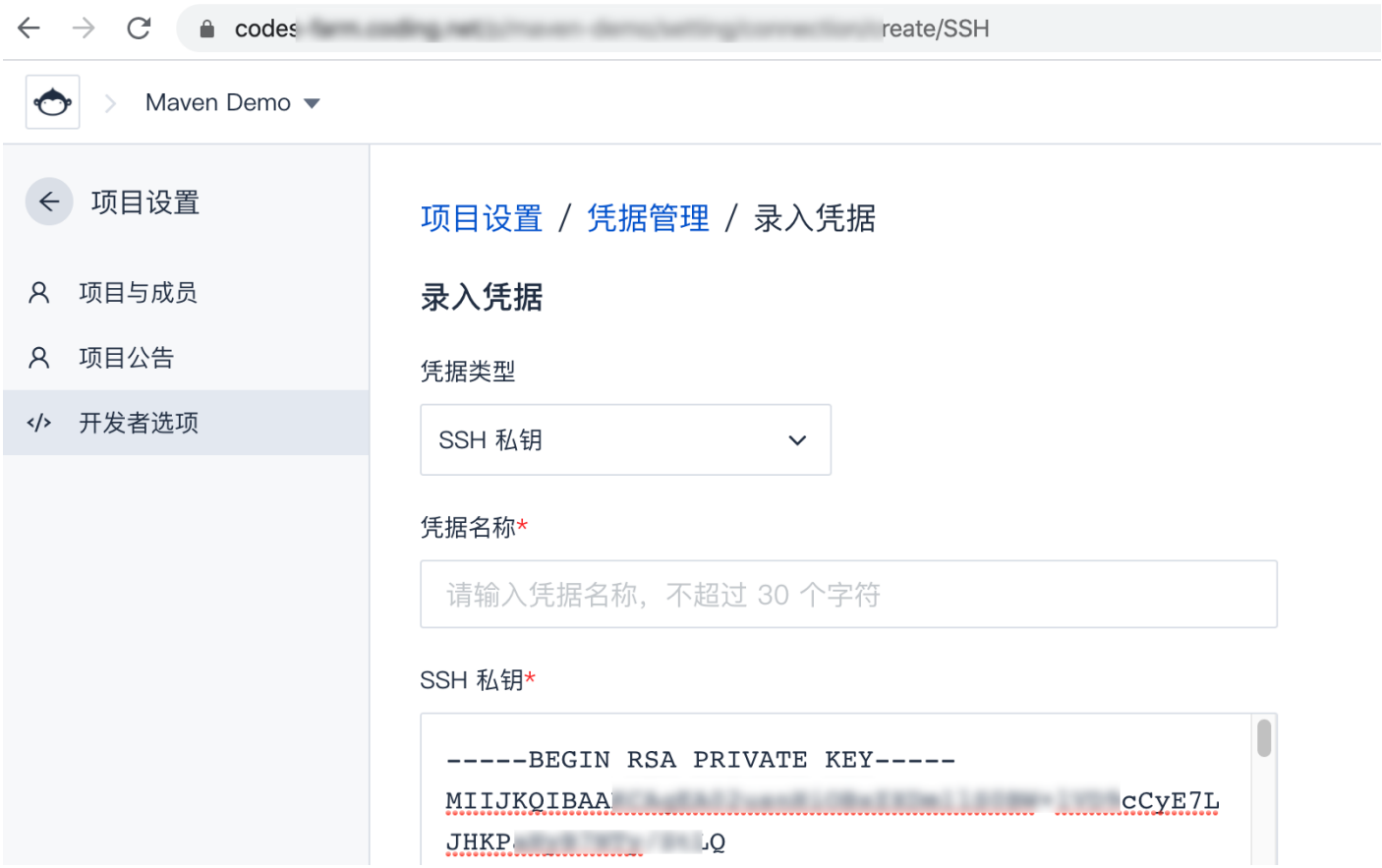
持续集成可简易部署项目到单台 Linux 服务器（多台建议使用 持续部署），步骤如下：

1. 在云计算（例如：腾讯云）的网页控制台创建 SSH 密钥对，把私钥录入 CODING 凭据管理，把公钥加载到服务器。也可以使用命令手动创建，把私钥 id_rsa 录入 CODING 凭据管理，把公钥 id_rsa.pub 的内容复制到服务器的 ~/.ssh/authorized_keys，参考文档：凭据管理。




腾讯云
总览
云产品
网站备案
云服务器
概览
实例
专用宿主机
置放群组
镜像
弹性伸缩
云硬盘
快照
SSH密钥
SSH密钥
创建密钥
绑定/解绑实例
修改
删除
ID/名称
实例绑定情况
skey-1xnmmdj1
CloudStudio
skey-2xkjpy2f
mbp
SSH密钥对已创建
我们不会保管您的私钥信息，请您在10分钟内点击"下载"按
下载
取消


腾讯云
总览
云产品
网站备案
云服务器
概览
实例
专用宿主机
置放群组
镜像
弹性伸缩
云硬盘
快照
SSH密钥
实例
上海(1)
新建
开机
多个关键字用竖线 "|" 分隔，多
ID/名称
ins-3g2hxzt1
web-server
共 1 条
加载密钥
1 选择密钥
2 关机提示
您已选择1台实例 查看详情
ID/名称
实例类型
ins-3g2hxzt1
web-server
标准型S3
对选中实例加载以下密钥：
搜索名称
名称
ID
coding
skey-dh59



2. 在持续集成中使用下述 Jenkinsfile：执行 SSH 进行部署。

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 '构建中...'
// Java Spring Boot Gradle Jar
// sh './gradlew bootJar'

// Java Spring Boot Gradle War
// sh './gradlew bootWar'
```

```
// Java Maven
// sh 'mvn package'
// sh 'ls target/'

// PHP
// sh 'composer install --optimize-autoloader --no-dev'
// sh 'tar -zcf /tmp/tmp.tar.gz .'

// 静态资源
sh 'tar -zcf /tmp/tmp.tar.gz apache2/ site/'
echo '构建完成.'
}
}
stage('部署') {
steps {
echo '部署中...'
script {
// 声明服务器信息
def remote = [:]
remote.name = 'web-server'
remote.allowAnyHosts = true
remote.host = '0.0.0.0'
remote.port = 22
remote.user = 'ubuntu'

// 把「CODING 凭据管理」中的「凭据 ID」填入 credentialsId，而 id_rsa 无需修改
withCredentials([sshUserPrivateKey(credentialsId: "c4af855d-xxxx-xxxx-xxxx-f6226ae3441c", keyFileVariable: 'id_rsa')]) {
remote.identityFile = id_rsa

// SSH 上传文件到远端服务器
sshPut remote: remote, from: '/tmp/tmp.tar.gz', into: '/tmp/'
// 解压缩
sshCommand remote: remote, command: "tar -zxf /tmp/tmp.tar.gz -C /tmp/"
sshCommand remote: remote, sudo: true, command: "mkdir -p /var/www/example-site"
sshCommand remote: remote, sudo: true, command: "cp -R /tmp/site/* /var/www/example-site/"
sshCommand remote: remote, sudo: true, command: "cp -R /tmp/apache2/ /etc/"
// 重启 apache2
sshCommand remote: remote, sudo: true, command: "a2ensite example.com"
sshCommand remote: remote, sudo: true, command: "a2enmod headers rewrite ssl"
sshCommand remote: remote, sudo: true, command: "systemctl reload apache2"
}
}

echo '部署完成'
}
}
}
}
```

建议服务器关闭密码登录，因为可能会被暴力破解，而只允许 SSH 私钥方式登录，更安全。
如果一定要使用密码登录服务器，则使用下述 Jenkinsfile 代码：

```
// 把「CODING 凭据管理」中的「凭据 ID」填入 credentialsId，而 username 和 password 无需修改
withCredentials([usernamePassword(credentialsId: "xxx", usernameVariable: 'username', passwordVariable: 'password')])
{
remote.user = username
remote.password = password

sshPut remote: remote, from: '/tmp/tmp.tar.gz', into: '/tmp/'
}
```

微信小程序

最近更新时间：2023-06-15 11:50:10

本文为您介绍如何通过持续集成自动更新微信小程序。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

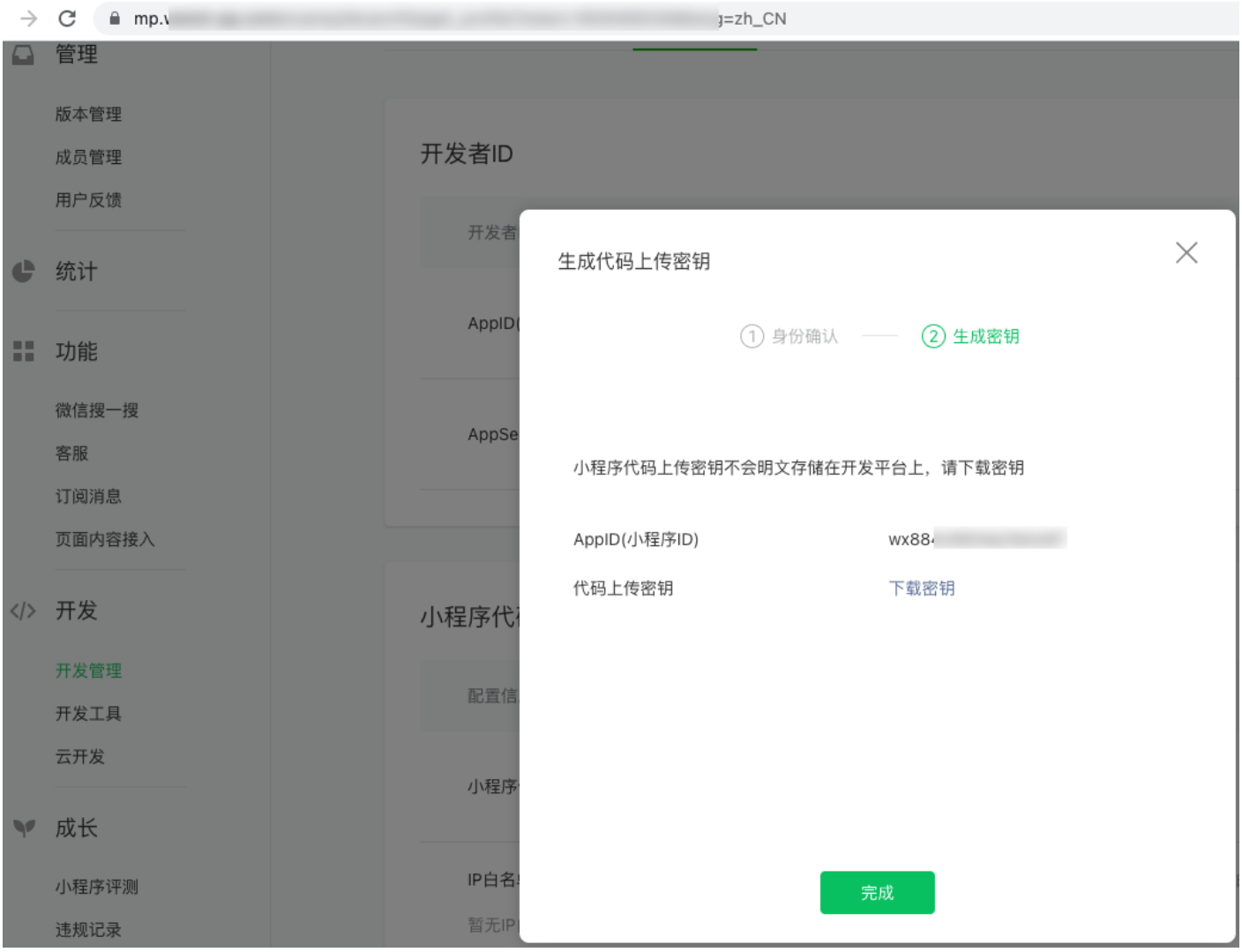
进入项目

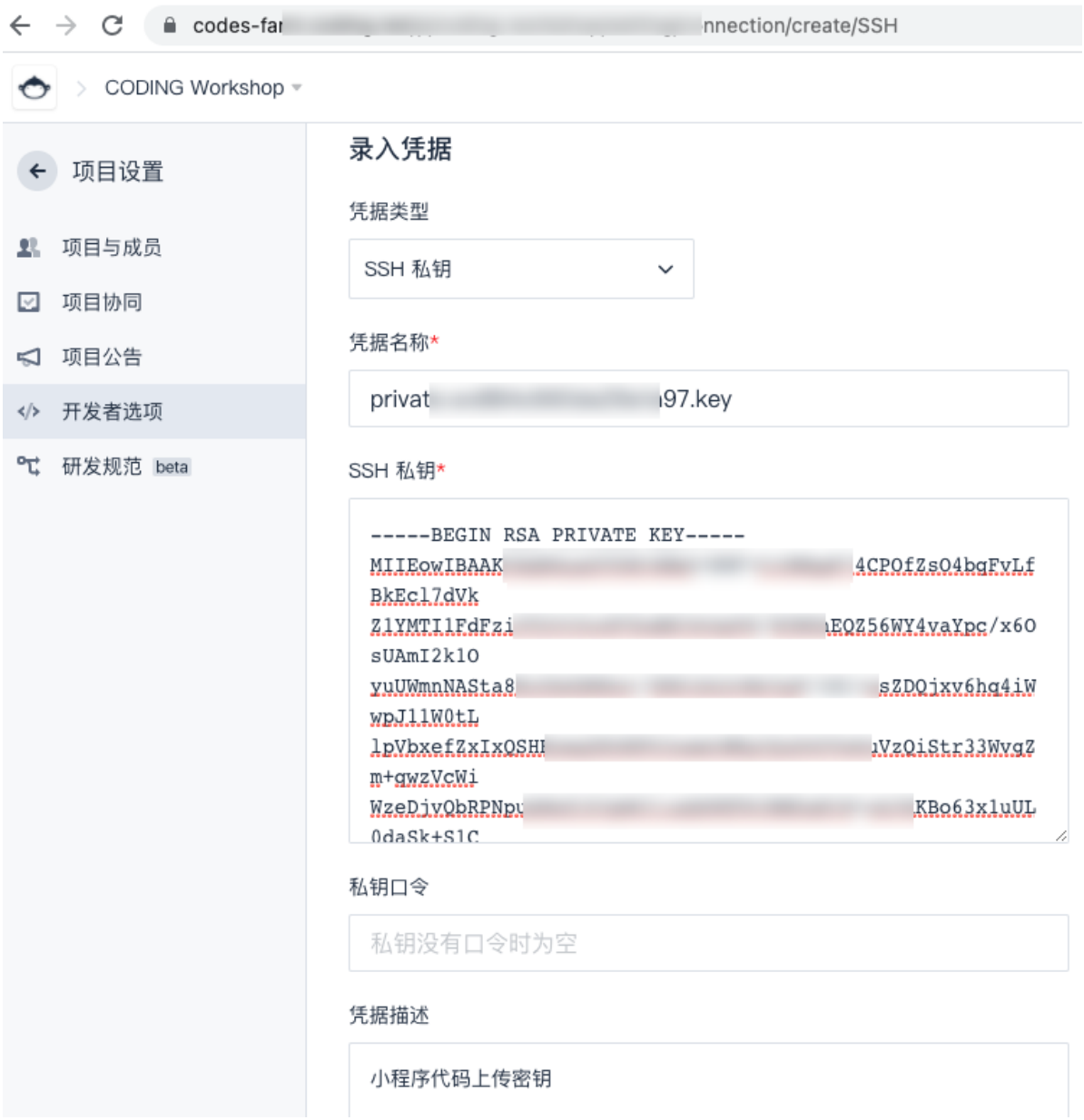
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

功能介绍

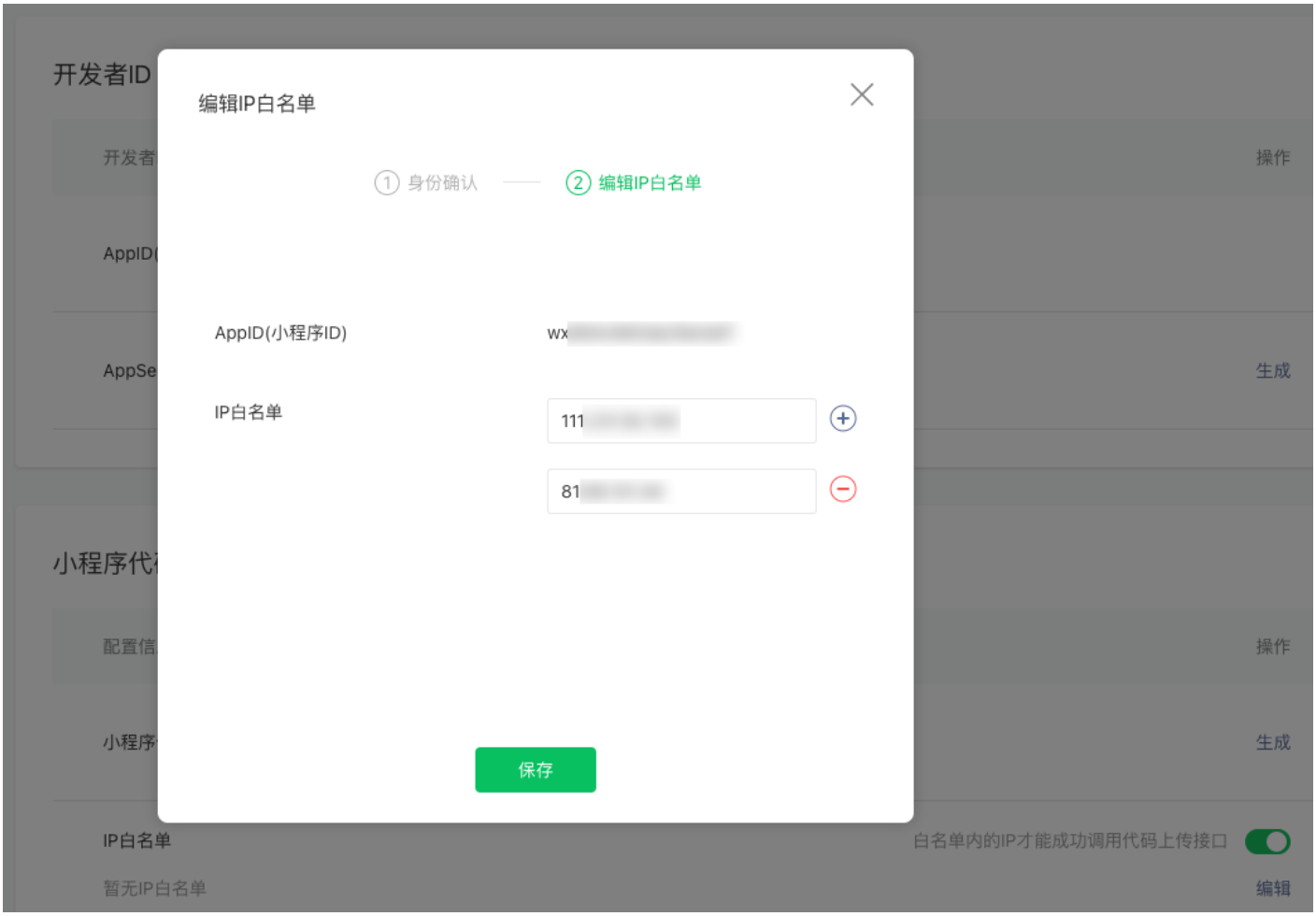
持续集成可自动上传微信小程序，步骤如下：

1. 在 微信小程序管理后台 生成代码上传密钥，录入 CODING 项目凭据管理。




codes-far nnection/create/SSH
CODING Workshop
项目设置
项目与成员
项目协同
项目公告
开发者选项
研发规范 beta
录入凭据
凭据类型
SSH 私钥
凭据名称*
privat 97.key
SSH 私钥*
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAK 4CPOfZsO4bqFvLf
BkEcl7dVk
Z1YMTI1FdFzi EQZ56WY4vaYpc/x6O
sUAmI2k1O
yuUWmnNASta8 sZDQjxv6hq4iW
wpJ11W0tL
lpVbxefZxIxOSHI VzQiStr33WvqZ
m+qwzVcWi
WzeDjvQbRPNpu KBo63x1uUL
0daSk+S1C
私钥口令
私钥没有口令时为空
凭据描述
小程序代码上传密钥

2. 把 CODING 持续集成的 IP 录入微信小程序管理后台的上传 IP 白名单。



3. 在持续集成中使用下述 Jenkinsfile 即可自动上传。

Jenkinsfile

```
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('准备依赖') {
steps {
sh 'npm install -g miniprogram-ci'
sh 'npm install'
```

```
}
}
stage('预览小程序') {
steps {
withCredentials([ sshUserPrivateKey(credentialsId: "$MP_PRIVATE_CREDENTIALS_ID",keyFileVariable:'SSH_PRIVATE_KEY_PATH')
]) {
sh "miniprogram-ci preview --pp ./miniprogram/ --pkp $SSH_PRIVATE_KEY_PATH --appid $MP_APP_ID --uv 1.0.0 --enable-es6 true --qrcode-format image --qrcode-output-dest qrcode.jpg"
archiveArtifacts(artifacts: 'qrcode.jpg', allowEmptyArchive: false)
}
}
}
stage('上传小程序') {
steps {
withCredentials([ sshUserPrivateKey(credentialsId: "$MP_PRIVATE_CREDENTIALS_ID",keyFileVariable:'SSH_PRIVATE_KEY_PATH')
]) {
sh "miniprogram-ci upload --pp ./miniprogram/ --pkp $SSH_PRIVATE_KEY_PATH --appid $MP_APP_ID --uv 1.0.0 --enable-es6 true"
}
}
}
}
}
```

持续集成环境变量

变量名	含义	参考值
MP_PRIVATE_CREDENTIALS_ID	小程序上传密钥的凭据 ID	abcdef00-1234-5678-bc0c-c57eddd2d123
MP_APP_ID	小程序 App ID	wx886c660da29a1234

运行结果截图

持续集成 / 微信小程序
构建记录#7
构建过程
构建快照
改动记录
测试报告
通用报告
构建产物
构建成功
sinkcup 手动触发
触发于 3 分钟前，持续时长 1 分钟 33 秒
构建过程
预览小程序 12 s
执行 Shell 脚本 12 s
收集构建物 < 1 s
上传小程序 14 s
执行 Shell 脚本 14 s
执行 Shell 脚本
14 秒
[2021-06-04 18:09:01] processing: upload
[2021-06-04 18:09:05] done: upload

同步代码库
定时同步开源代码库

最近更新时间：2023-06-15 11:49:42

本文为您介绍如何定时同步开源代码库。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

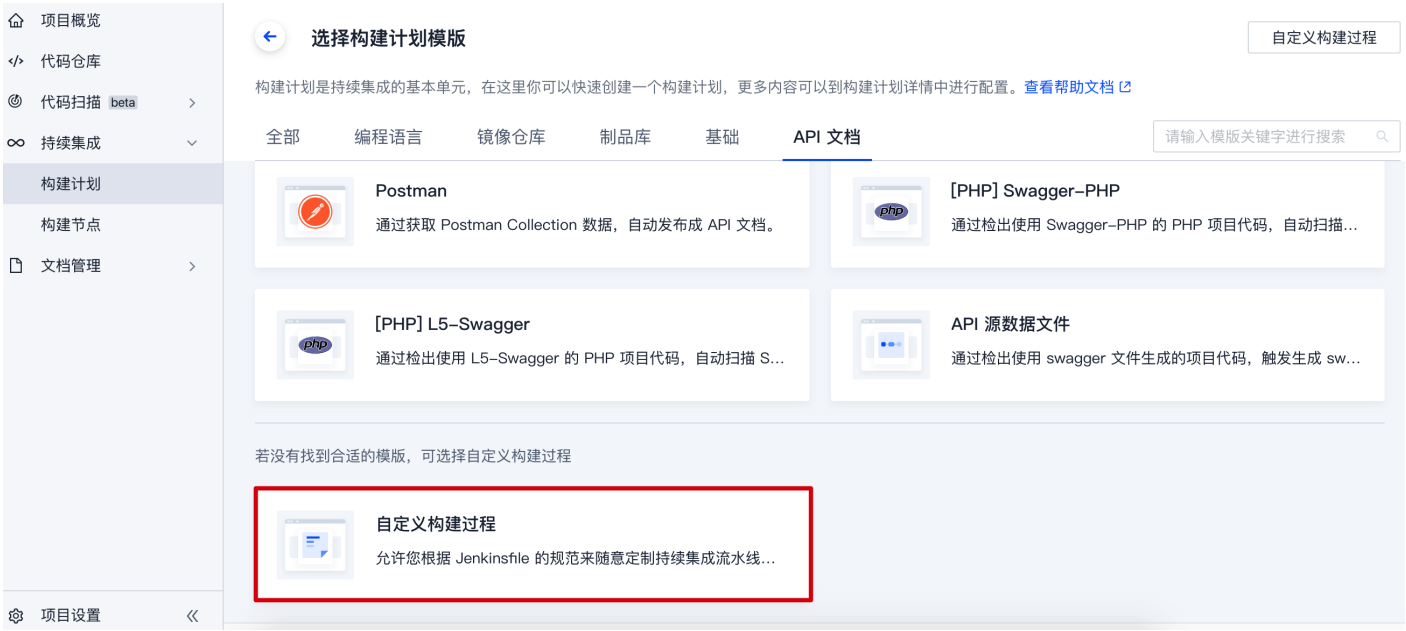
申请项目令牌

1. 进入项目后，轻点左下角的**项目设置** > **开发者选项** > **项目令牌**，点击**新建项目令牌**，键入令牌名称并勾选文件及持续集成权限。

2. 创建完成后便会获取到一串用户名和密码。请注意该项目令牌的过期时间，以免影响到持续集成构建任务。

创建持续集成任务

1. 在项目里的持续集成中新建构建计划，选择自定义构建过程。



2. 键入构建计划名称，代码源选择拟进行同步的 CODING 代码仓库，配置来源勾选使用静态配置的 Jenkinsfile 后前往配置详情。在流程配置中选择文本编辑器，可以参考以下配置文件进行编写。

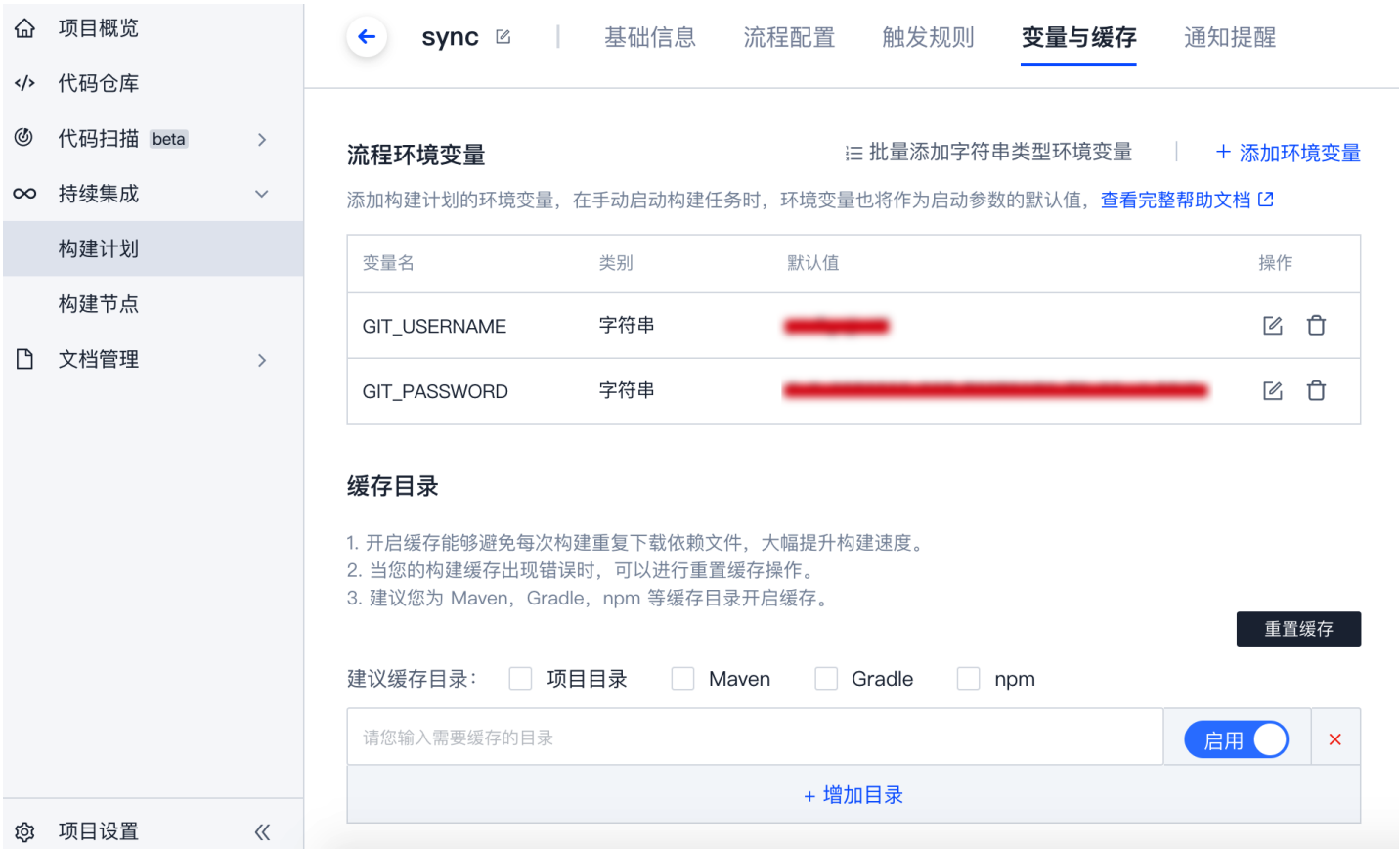
Jenkinsfile

```
pipeline {
agent any
stages {
stage('检出 CODING') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
}
}
stage('拉取 GitHub') {
```

```
steps {
sh "git remote add github https://github.com/phodal/ledge.git" // 此处需替换为您需要同步的 GitHub 代码仓库地址。
sh "git remote update github"
sh "git merge github/master"
}
}
stage('推送到 CODING') {
steps {
// 使用项目令牌环境变量 PROJECT_TOKEN_GK 和 PROJECT_TOKEN 来作为推送至 CODING 代码仓库所需的用户名和密码。
// 若希望推送到非本项目的代码仓库或第三方平台的代码仓库，需要自行更换为有效的凭据信息
sh "git push https://${PROJECT_TOKEN_GK}:${PROJECT_TOKEN}@e.coding.net/coding-public/ledge.git HEAD:master"
}
}
}
}
```

添加环境变量

此步骤的目的是在构建任务中使用项目令牌，以通过代码仓库的账号密码推送验证。可以使用 GIT_USERNAME 和 GIT_PASSWORD 作为变量名称，默认值为 上文 中所申请到的用户名和密码。



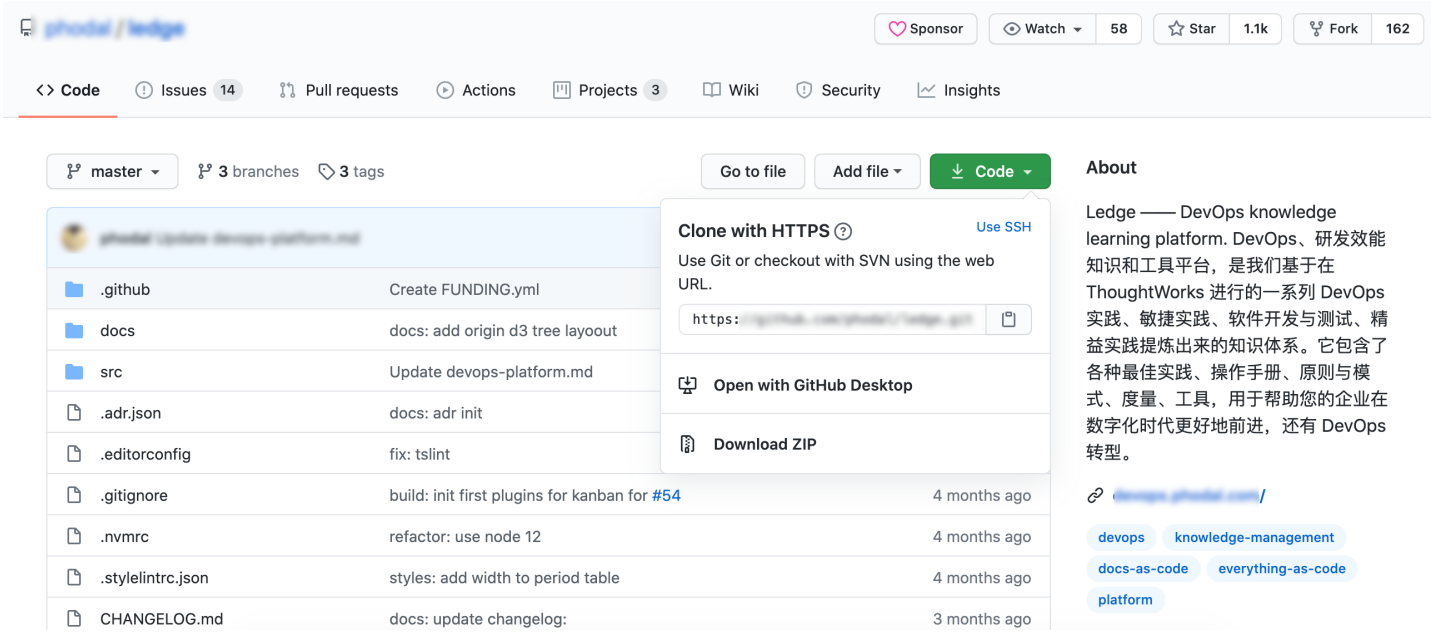
设置触发规则

在持续集成设置中的触发规则中添加定时触发，您可以按照所需要的频率进行设置，还可以在该页面设置其他的触发规则，将 CI 任务的触发无缝融入至您的工作流之中。

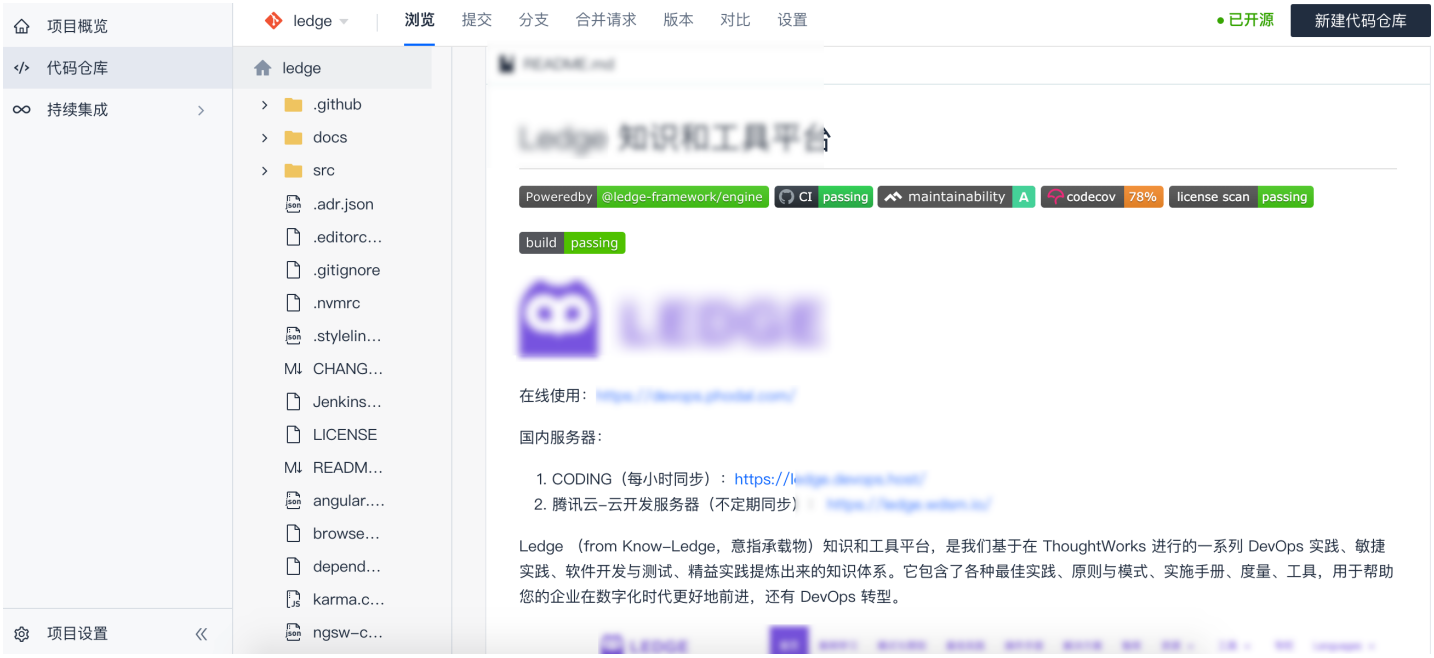
触发持续集成任务

完成上述步骤后，点击立即构建便可以看到构建过程。待构建成功后，便可以看到 CODING 代码仓库已和 GitHub 代码仓库保持一致了。

GitHub 代码仓库



CODING 代码仓库



定时同步私有代码库

最近更新时间：2023-06-15 11:49:17

本文为您介绍如何定时同步私有代码库。

前提条件

设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

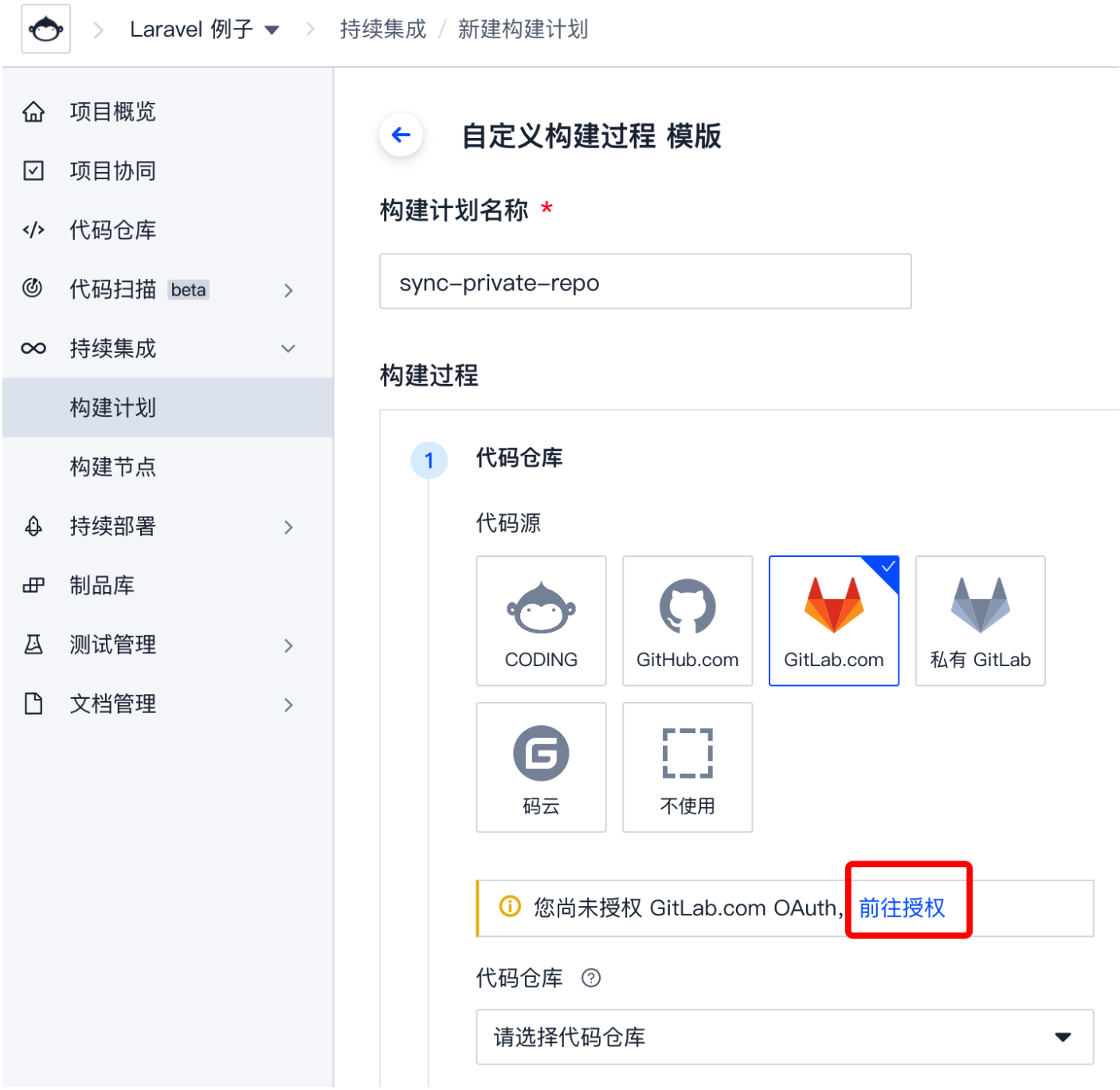
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的  ，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

如果您在使用外部私有代码库（例如：GitHub、GitLab.com、自建 GitLab、码云），希望迁移至 CODING，可通过持续集成定时同步。

1. 在 CODING 中创建空代码仓库，把您的代码手动推送一次。

```
git pull
git remote add coding git@e.coding.net:your-team/project/repo.git
git push coding main
```

2. 在 CODING 持续集成中创建**自定义构建过程**，授权绑定您的外部账号，并选择代码仓库。



3. 在 CODING 持续集成中修改**流程配置**，使用下方 Jenkinsfile 代码：

```
pipeline {
agent any
stages {
stage('检出 GitHub') {
steps {
checkout([
$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]
])
```

```
}
}
stage('推送到 CODING') {
steps {
// 无需修改 PROJECT_TOKEN_GK 和 PROJECT_TOKEN，它们为 CODING 内置环境变量
// 请修改为你的代码库链接
sh "git push https://${PROJECT_TOKEN_GK}:${PROJECT_TOKEN}@e.coding.net/your-team/project/repo.git HEAD:master"
}
}
}
}
```

4. 在 CODING 持续集成中修改**触发规则**，关闭**代码更新时自动执行**，开启**定时触发**，最高频率为5分钟一次。

调取已录入的凭据

最近更新时间：2021-08-25 16:38:21

本文为您详细介绍如何使用持续集成调取已录入的凭据。

前提条件

使用 CODING 持续集成的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击页面右上角的 ，进入项目列表页面，单击项目图标进入目标项目。
3. 进入项目后单击左下角的**项目设置**。

功能介绍

在持续部署中，有时候会需要用到第三方供应商所提供的服务，这时候就需要调用相关的凭据来获取权限。目前持续部署功能已集成内置插件，支持快速调取相关凭据。

使用插件快速调取已录入的凭据

在 CODING 持续集成任务构建过程当中，如果将 Github 的账号密码等重要信息硬编码在配置文件内，将会有极大的安全隐患。通过 CODING 的 凭据管理 功能，将凭据 ID 写入配置文件中进行服务调用。在接下来的插件功能使用中，请确保您已将凭据托管至 CODING 中。

快速开始

下面以调取凭据管理中的云 API 密钥为例，演示如何使用 Jenkinsfile 配置快速调取已录入的凭据。

1. 将获取到的 API 密钥上传至 CODING 进行托管以获得凭据 ID。

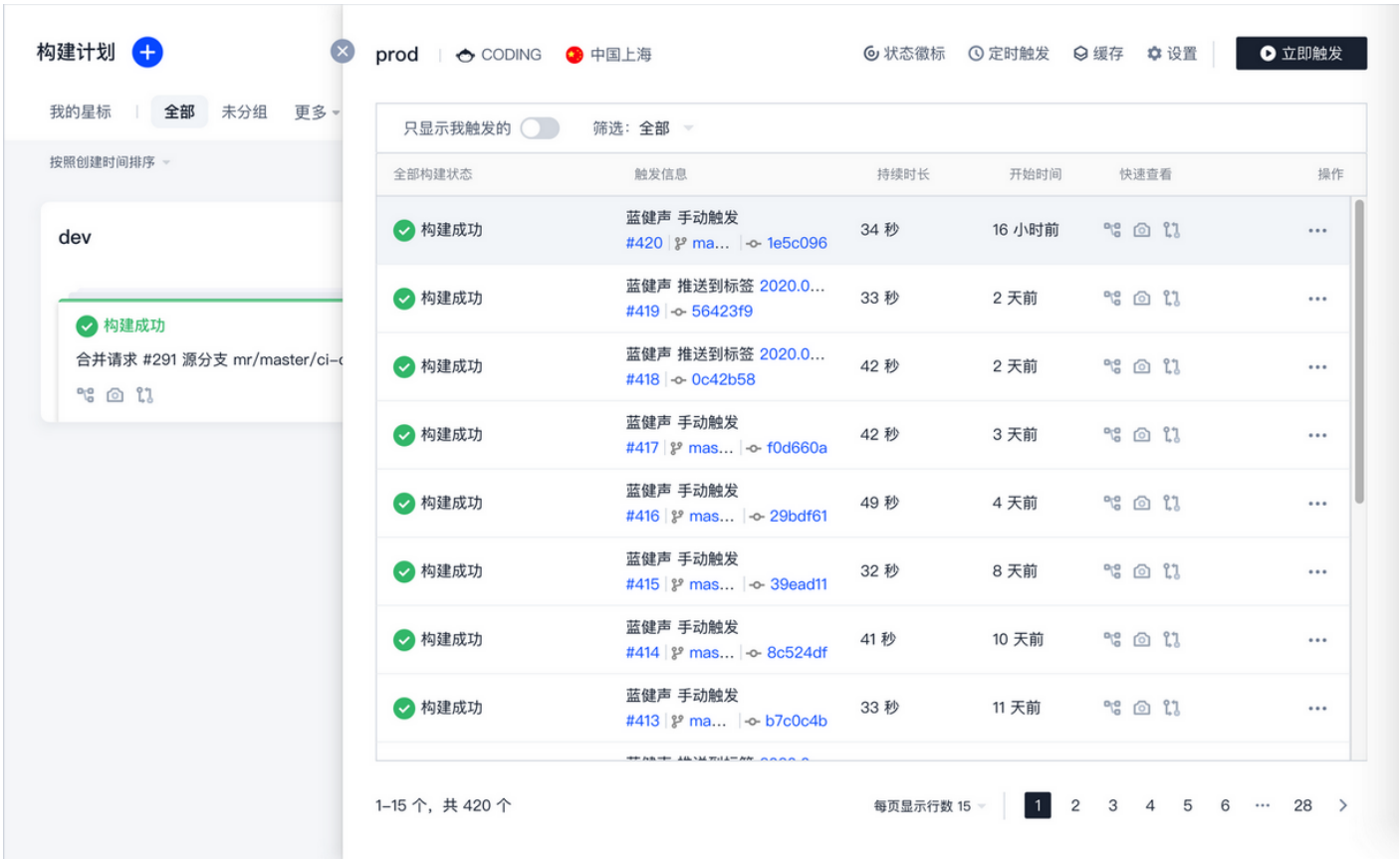
2. 在变量与缓存中选择**增加环境变量 > 类别选择 CODING 凭据 > 选择您需调取的凭据**。

3. 在构建于部署中新建计划列表，并填写相应的 Jenkinsfile 配置

Jenkinsfile 配置

```
pipeline {
agent any
stages {
stage('获取云 API 密钥') {
steps {

withCredentials([cloudApi(credentialsId: '此处填写您上传凭据后所生成的凭据 ID', secretIdVariable: 'CLOUD_API_SECRET_ID', s
ecretKeyVariable: 'CLOUD_API_SECRET_KEY')]) {
sh 'CLOUD_API_SECRET_ID=${CLOUD_API_SECRET_ID}'
sh 'CLOUD_API_SECRET_KEY=${CLOUD_API_SECRET_KEY}'
}
withCredentials([[$class: 'CloudApiCredentialsBinding', credentialsId: '此处填写您上传凭据后所生成的凭据 ID', secretIdVariabl
e: 'CLOUD_API_SECRET_ID', secretKeyVariable: 'CLOUD_API_SECRET_KEY']]) {
sh 'CLOUD_API_SECRET_ID=${CLOUD_API_SECRET_ID}'
sh 'CLOUD_API_SECRET_KEY=${CLOUD_API_SECRET_KEY}'
}
}
}
}
}
```

4. 构建完成



参数说明

参数名称	是否必填	默认值	说明
credentialsId	是	-	需要获取的凭据 ID，仅支持云 API 类型的凭据
secretIdVariable	是	-	secretId 环境变量的名称，会用配置名称注入环境变量
secretKeyVariable	是	-	secretKey 环境变量的名称，会用配置名称注入环境变量