CODING DevOps

持续集成

腾讯云

文档目录

持续集成
快速开始

最近更新时间：2023-09-11 16:05:22

下文将会演示如何利用 CODING 持续集成模板快速发布一个基于 Node.js + Express + Docker 项目。
观看视频

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. 单击团队首页左侧的**项目**，进入项目列表页，选择目标项目。
3. 进入左侧菜单中的**持续集成**。

1. 创建构建计划

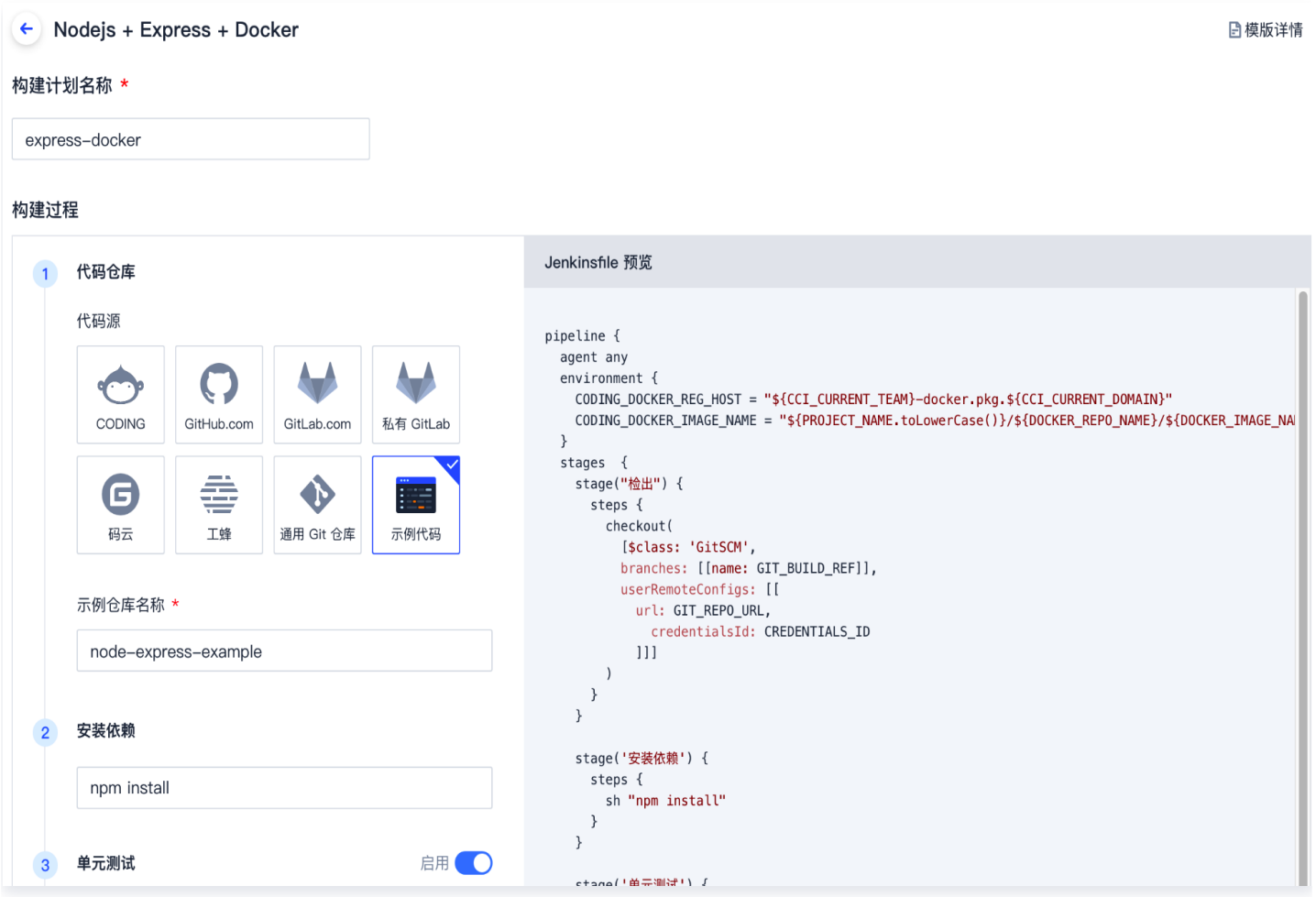
进入项目后，选择左侧的**持续集成 > 构建计划**，单击创建构建计划。该计划将会演示如何基于 Nodejs + Express 实现全自动检出代码 > 单元测试 > 构建 Docker 镜像 > 推送到 Docker 制品库 > 部署到远端服务器（可选步骤）。

2. 选择持续集成模板

选择 Node + Express + Docker 持续集成模版。

3. 选择代码源

在代码仓库栏选择**示例代码**作为代码源，系统将自动在您的项目中新建立一个示例代码仓库。



4. 选择制品库

构建计划结束后会生成一个构建结果，在这里选择拟推送到的 Docker 制品库。若项目内尚未存在制品库，可以利用快捷方式进行新建。

5. 填写远端服务信息（可选步骤）

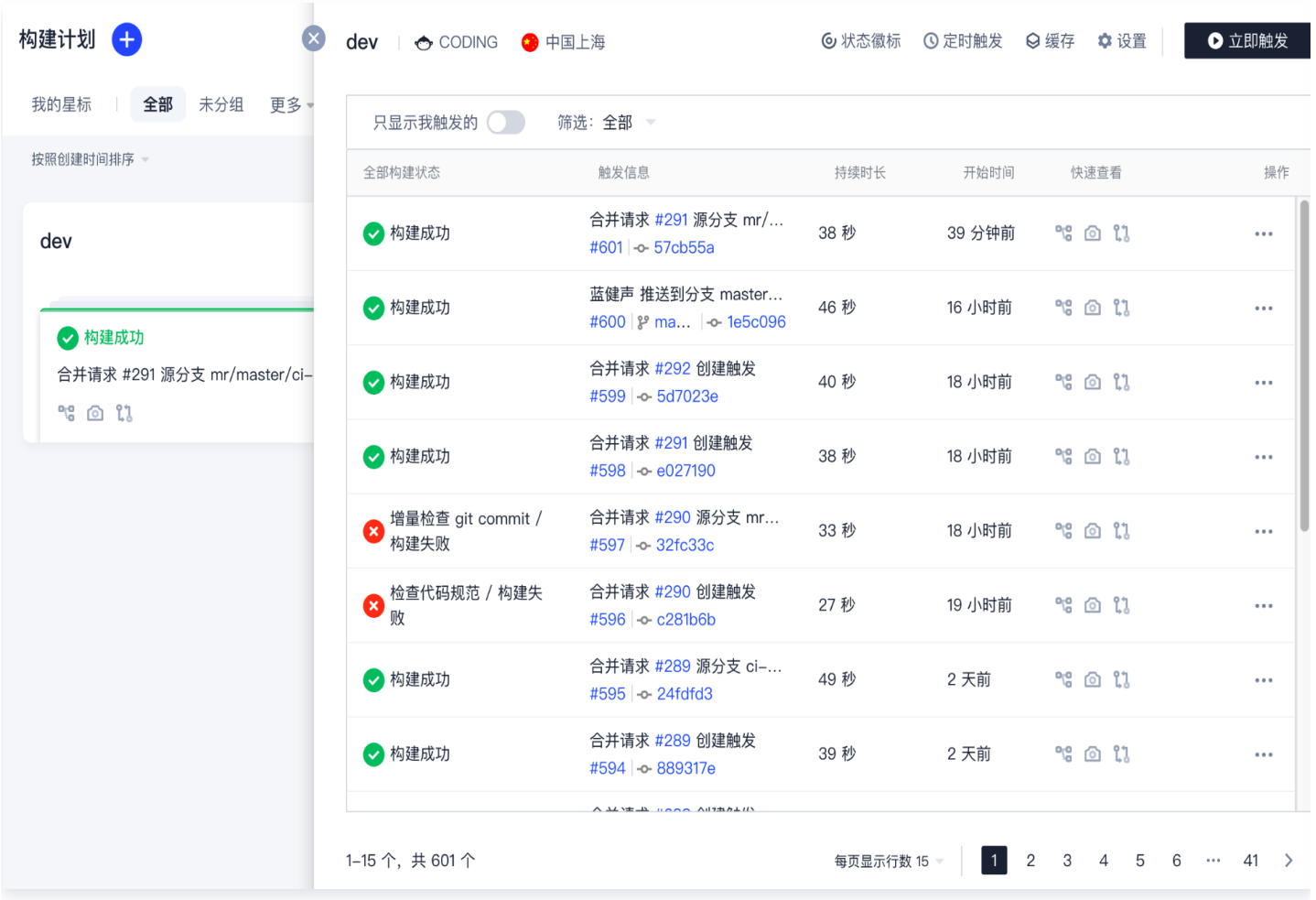
填写拟部署的远端服务器信息，包含 IP 地址及端口等信息并录入服务器 SSH 登录凭据。信息填写正确无误后，待构建计划完成后会将制品发送至远端服务器中，通过一个网址便可预览发布后的效果。如果暂时不需要部署到远端服务器，可跳过此步骤。

单击**录入新凭据并授权**后，若您本身是通过 SSH 私钥登录远程服务器的，在录入方式中则直接单击**手动录入已有 SSH 私钥**。录入完成后可以在**项目设置 > 开发者选项 > 凭据管理**中查看。

若不知道如何使用 SSH 私钥登录远程服务器，请在录入方式中单击**自动创建 SSH 密钥对**，并需要手动将公钥配置到目标远端服务的 `~ssh/authorized_ keys` 文件夹中。

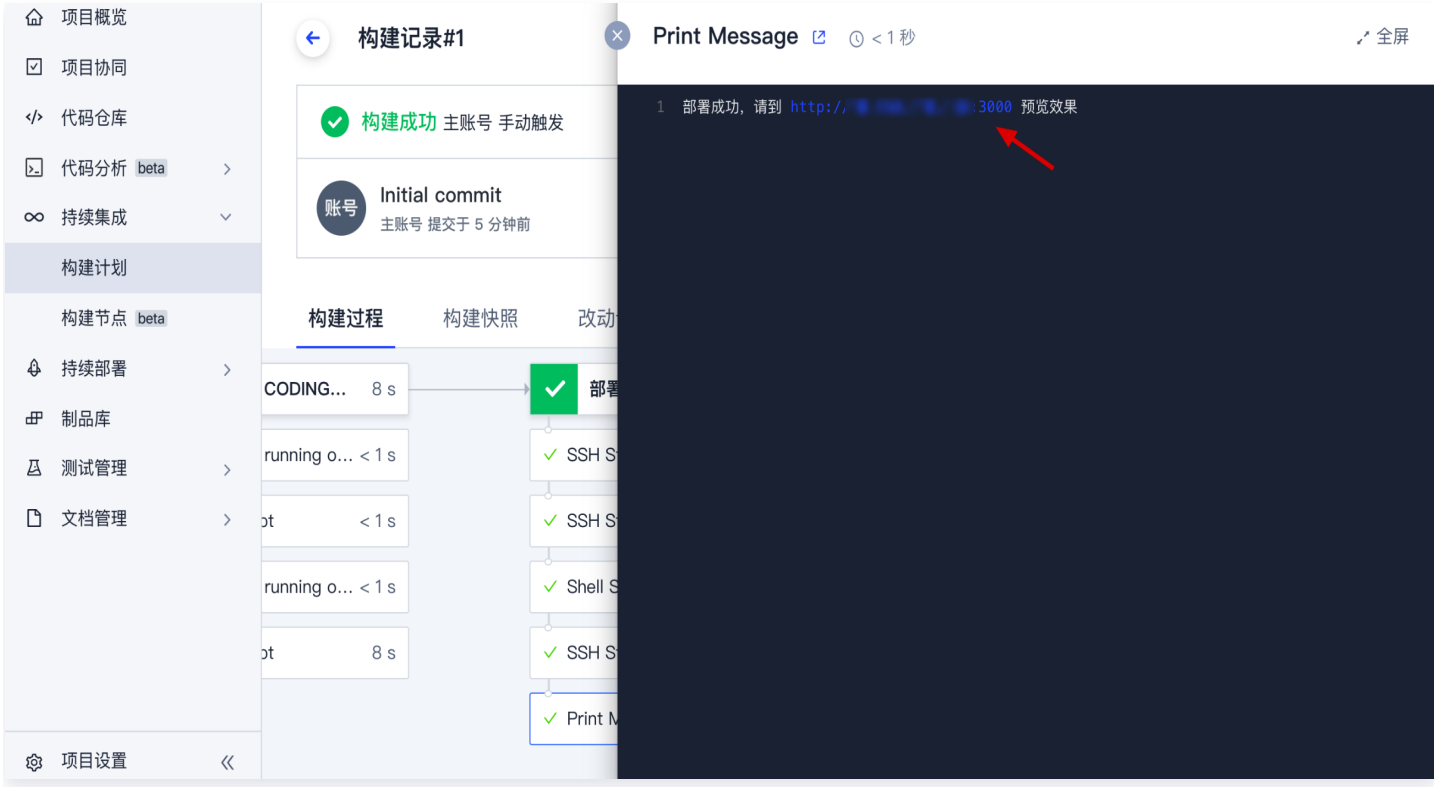
6. 单击创建并查看构建结果

单击**确定**保存构建计划。如果勾选了**创建后触发构建**，构建计划会立即开始执行。在构建计划执行的过程中，可以在构建计划记录列表页查看构建详情。



单击构建记录可以查看流水线上每一个阶段是否运行成功，还可以看到每一个步骤命令的具体的执行效果和日志。

若步骤5运行正常，则可以在构建计划中看到制品输出的网址信息。

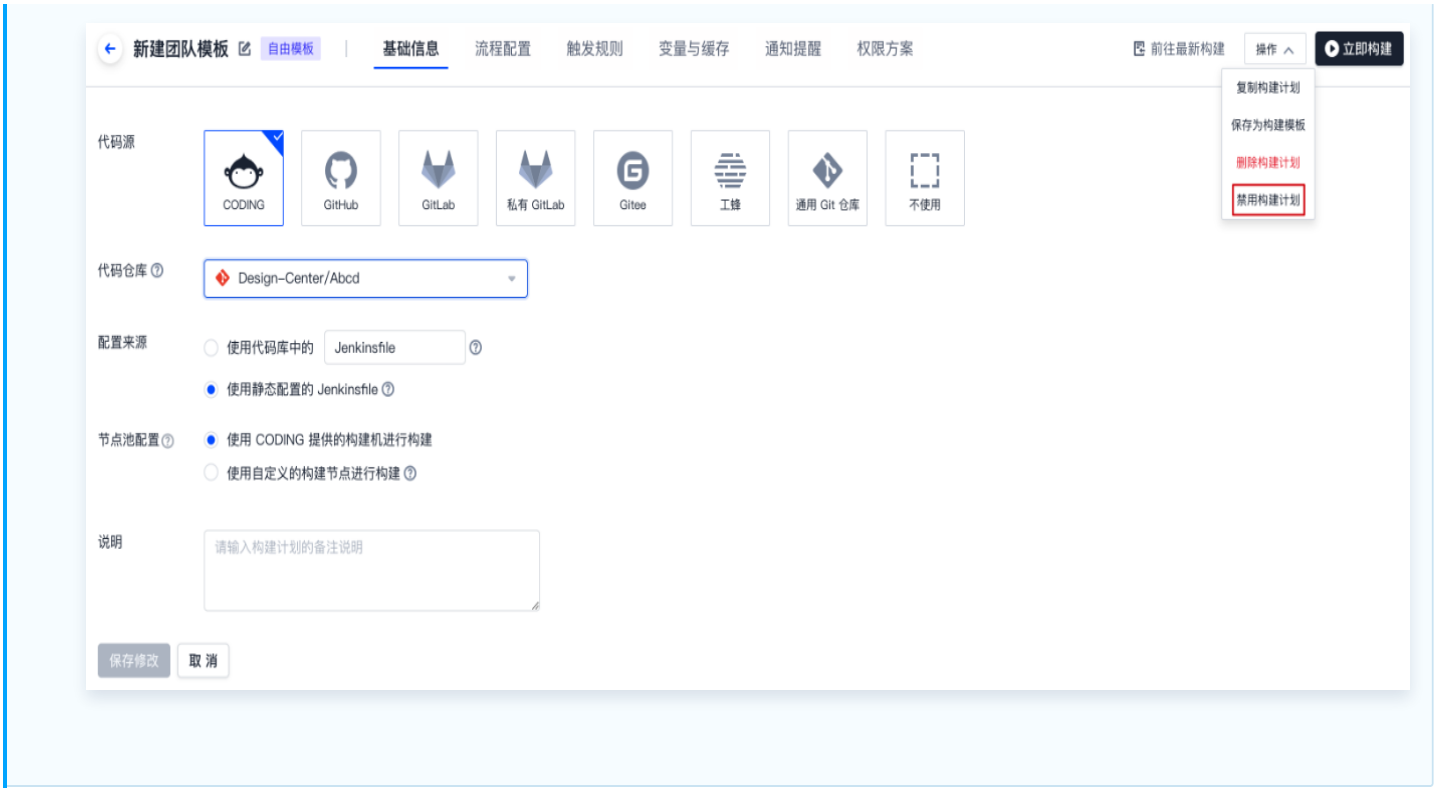


7. 修改远端服务器信息

在步骤5中您已经配置了远端服务器的地址和 SSH 密钥。如果需要更改，可以到持续集成计划 > 设置 > 变量与缓存中更改。

说明：

构建计划创建成功后，当不需要构建该计划时支持禁用构建计划。已禁用的构建计划不会被触发，启用后可正常运行。


新建团队模板
自由模板
基础信息
流程配置
触发规则
变量与缓存
通知提醒
权限方案
前往最新构建
操作
立即构建
复制构建计划
保存为构建模板
删除构建计划
禁用构建计划
代码源
CODING
GitHub
GitLab
私有 GitLab
Gitee
工蜂
通用 Git 仓库
不使用
代码仓库
Design-Center/Abcd
配置来源
使用代码库中的
Jenkinsfile
使用静态配置的 Jenkinsfile
节点池配置
使用 CODING 提供的构建机进行构建
使用自定义的构建节点进行构建
说明
请输入构建计划的备注说明
保存修改
取 消

编写构建流程
文本编辑器

最近更新时间：2023-09-11 16:05:22

本文为您介绍如何使用持续集成中的文本编辑器。

前提条件

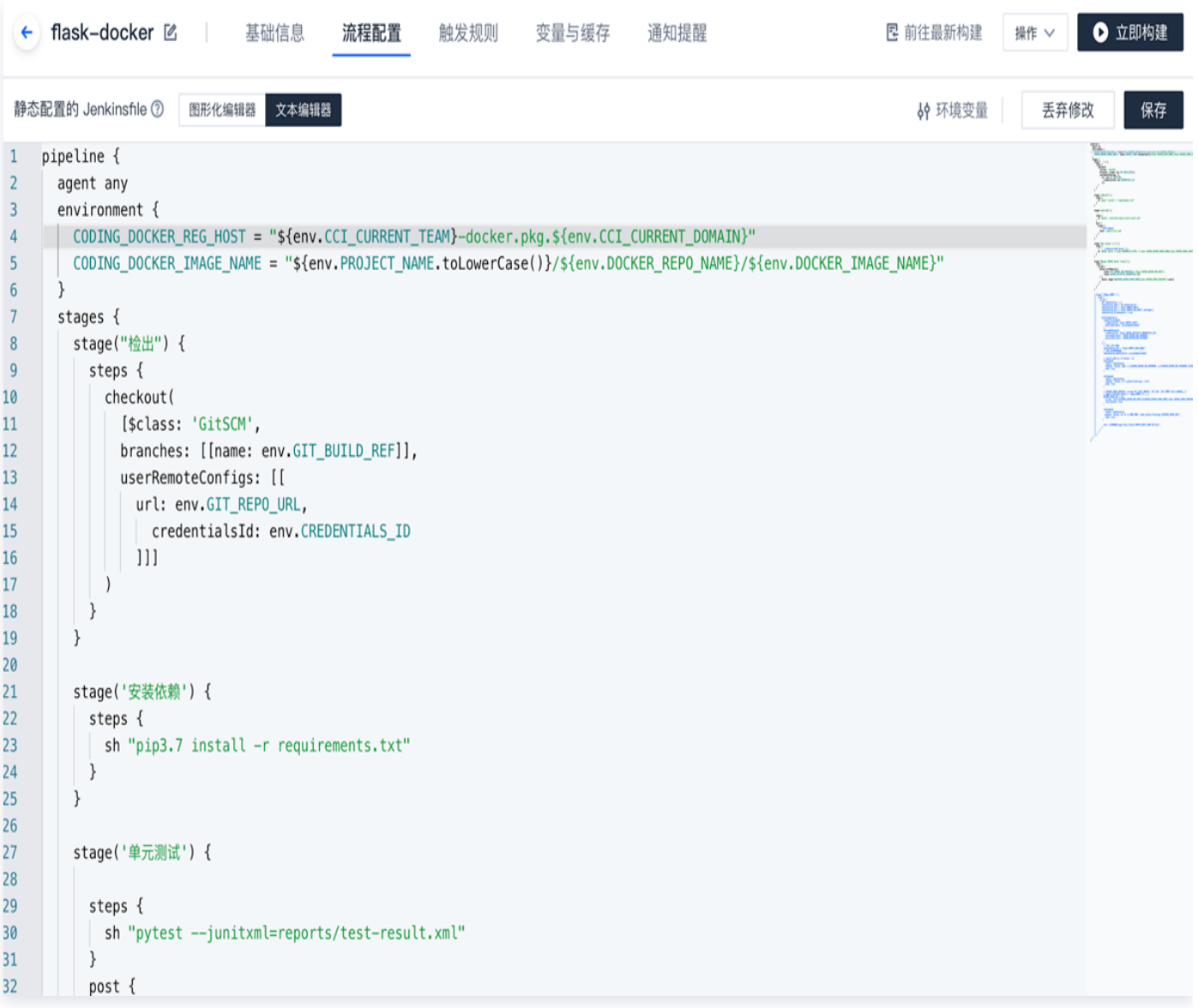
设置 CODING 持续集成中构建环境前，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务 。

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. 进入目标项目后，单击左侧的**持续集成**，单击构建计划的**设置**。

   构建任务的流程本质上是在遵循配置文件中所定义的流程与步骤。CODING 持续集成全面兼容 Jenkinsfile，在文本编辑器中编写的配置文件所需遵循的语法规范与 Jenkinsfile 保持一致即可成功运行。

3. 单击**流程配置**中的**文本编辑器**。



构建过程语法遵循 Jenkinsfile 语法，详情请参考下方文档：

- Jenkinsfile 官方文档
- 配置详情

流程配置详情

最近更新时间：2024-11-18 10:12:32

本文主要用于辅助编写构建过程，以及说明各项步骤的参数详情。

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. 进入目标项目后，单击左侧的**持续集成**，单击**构建计划 > 文本编辑器**并在其中进行编排。

代码仓库

Git

用以检出当前项目的 Git 仓库源代码。本指令是 checkout 指令的一个简易写法。
参数列表：

参数	类型
Git 地址 `url`	string
分支 `branch`	string
变更日志 `changelog`	string
身份认证 ID `credentialsId`	string
Poll `poll`	boolean

从版本控制检出

通用的检出 SCM（Git，SVN）代码。
这个步骤返回一个 Map 格式的内容。例如您用 Git，您可以这样：

```
def scmVars = checkout scm
def commitHash = scmVars.GIT_COMMIT
// or
def commitHash = checkout(scm).GIT_COMMIT
```

参数 `scm` 是一个可配置 SCM 类型的对象，目前支持的有如下：

- GitSCM 示例写法：

```
checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
                userRemoteConfigs: [[url: env.GIT_REPO_URL]]])
```

userRemoteConfigs 参数列表：

参数	类型
locations	对象数组。
remote	string。
credentialsId	string。
local	string，指明一个本地目录（相对于 workspace）作为检出代码的位置。
depthOption	string，对应 --depth 的内容。默认值是无穷大，详情请参见 Subversion。
ignoreExternalsOption	boolean。

- SubversionSCM 从 SVN 服务器检出代码。示例写法：

```
checkout([$class: 'SubversionSCM', remote: 'http://sv-
server/repository/trunk'])
```

参数列表：

参数	类型
locations	对象数组。
remote	string。
credentialsId	string。
local	string，指明一个本地目录（相对于 workspace）作为检出代码的位置。
depthOption	string，对应 --depth 的内容。默认值是无穷大，详情请参见 Subversion。
ignoreExternalsO	boolean。

ption	

构建过程

子节点

参数列表：

`label`：类型 string 环境标签名称，例如 java-8。

收集构建物

把构建结果（例如 jar，war，apk 等）收集起来。请注意，这里收集的构建产物会跟随这个构建历史一起保存和删除，这里只是一个临时的保存空间，更建议用户使用“构建物管理”来进行构建结果的版本化管理。

参数列表：

参数	类型
artifacts	string，可以使用通配符 * 来指定要收集的文件的路径模式，符合 Apache Ant 的路径规则 ，只允许指定工作空间内的文件。
allowEmptyArchive	boolean，可选；通常情况下，本指令在找不到符合收集模式的文件时会导致构建失败。这个选项如果设置为 true ，那么当没有构建产物的时候，构建过程只会发出一个警告，不会导致失败。
caseSensitive	boolean，可选；默认对文件路径规则的匹配是大小写敏感的，如果设置为 false，则不区分大小写。
defaultExcludes	boolean，可选。
excludes	boolean，可选；在设定的路径模式内可以排除一部分文件，同样支持 Apache Ant 的路径规则。
fingerprint	boolean，可选；收集的时候同时计算文件的 hash 信息。
onlyIfSuccessful	boolean，可选；只有当构建成功的时候才收集。

执行 Shell 脚本

执行一段 Shell 脚本，示例：

```
pipeline {
    agent any
    stages {
        stage('Example') {
```

```
            steps {
                echo 'Hello World'
                sh 'ls -al'
            }
        }
    }
}
```

收集 JUnit 测试报告

收集 JUnit 和 TestNG 的测试报告（xml 格式），您可以指定收集哪些 xml 文件，例如 `**/build/test-reports/*.xml` 需要注意的是不要把那些不是报告文件的 xml 包含在内了，您可以用逗号隔开写多个规则。
参数列表：

参数	类型
testResults	string。
allowEmptyResults	boolean，可选，允许测试报告文件不存在或为空。
keepLongStdio	boolean，可选，所有测试日志都会保留，即便是那些通过的测试用例。

其他

变更目录子步骤

变更目录子步骤。可以在 dir 块内填充若干子步骤，这些子步骤将会在指定的路径目录内执行。
参数列表：
`path`：类型 string。

睡眠

即暂停一段时间，直至所设定的截止时间。与 Unix 的 `sleep xxx` 类似。
参数列表：

- `time`：类型 int。
- `unit`：只能在 `NANOSECONDS`，`MICROSECONDS`，`MILLISECONDS`，`SECONDS`，`MINUTES`，`HOURS`，`DAYS` 中选择一个。

错误信号

发送一个错误信号，往往用在需要根据条件终止部分执行过程的时候。您也可以使用 throw new Exception（），但使用 error 步骤可以避免打印过长的异常栈。

参数列表：

`message`：类型 string。

当前目录

把当前目录路径以字符串类型作为返回结果。

参数列表：

`tmp`：类型 boolean 可选参数，如果选择了，则返回一个跟工作空间关联的临时目录。通常用于在不想污染工作空间目录，又想存放一些临时文件之类的场景。

写文件

把指定内容写入文件。参数列表：

- `file`：类型 string。
- `text`：类型 string。
- `encoding`：类型 string 文件的编码，如果留空将会根据当前运行环境平台默认编码处理，如果遇到 Binary 文件，则会自动被以 Base64 编码后的结果返回。

读文件

从相对路径读取文件，并把文件内容作为字符串返回。参数列表：

- `file`：类型 string 相对路径地址（相对于工作空间目录）。
- `encoding`：类型 string 文件的编码，如果留空将会根据当前运行环境平台默认编码处理，如果遇到 Binary 文件，则会自动被以 Base64 编码后的结果返回。

重试子步骤

重试一个指定的块直至达到设定的最多重试次数。如果在执行过程中正常结束，则不再重试，如果执行过程中出现异常，则会不断重试直至达到指定的最大重试次数，如果最后一次尝试出现异常，则构建过程会被终止。

参数列表：

`count`：类型 int。

限时子步骤

限定时间执行块内的过程。如果时间到了，将会抛出一个异常 `org.jenkinsci.plugins.workflow.steps.FlowInterruptedException`。单位参数是可选的，默认是分钟。参数列表：

- `time`：类型 int。
- `activity`：类型 boolean，以日志没有新的内容来计时，而不是以绝对执行时间来计时。
- `unit`：只能在 `NANOSECONDS`，`MICROSECONDS`，`MILLISECONDS`，`SECONDS`，`MINUTES`，`HOURS`，`DAYS` 中选择一个。

捕获错误子步骤

这里指定的子步骤中的错误将会被捕获。

计时子步骤

这里指定的子步骤的执行时间将会以 Unix 时间戳的形式被记录。

循环子步骤

这里指定的子步骤将会被循环执行指定次数。

条件循环子步骤

这里指定的子步骤将会被循环执行，直到子步骤的结果返回 `true` 。

打印消息

在日志中打印消息。
参数列表：
`message`：类型 `string`

执行任意的 Pipeline 脚本

添加此步骤可以执行任意的 Pipeline 脚本。

执行 Groovy 源文件

构建过程将在此位置执行 Groovy 源文件。
示例：

```
pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
                load 'test.groovy'
            }
        }
    }
}
```

执行 Yarn 审计

此步骤在指定的目录里执行 yarn audit，并且可以在持续集成结果页面看到对 yarn 依赖审查的漏洞情况。
参数列表：

- `directory`：类型 string，可选。填写 yarn.lock 所在目录，默认在项目根目录执行。

- `collectResult`：类型 boolean，可选。收集 Yarn Audit 报告。

执行 Npm 审计

此步骤在指定的目录里执行 npm audit，并且可以在持续集成结果页面看到对 npm 依赖审查的漏洞情况。
参数列表：

- `directory`：类型 string，可选。填写 package.json 所在目录，默认在项目根目录执行。
- `collectResult`：类型 boolean，可选。收集 Npm Audit 报告。

合并合并请求

合并代码。您可以在此步骤合并一个指定的合并请求。
参数列表：

- `token`：类型 string。项目令牌。
- `depot`：类型 string。仓库名称。
- `mrResourceId`：类型 string。指定资源 ID。
- `commitMessage`：类型 string。合并提交信息模板。
- `deleteSourceBranch`：类型 boolean，可选。删除源分支。
- `fastForward`：类型 boolean，可选。尝试 Fast-Forward 模式合并。

评论合并请求

评论合并请求。您可以在此步骤评论一个指定的合并请求。
参数列表：

- `token`：类型 string。项目令牌。
- `depot`：类型 string。仓库名称。
- `mrResourceId`：类型 string。指定资源 ID。
- `commentContent`：类型 string。评论内容模板。

图形化编辑器

最近更新时间：2024-11-18 10:12:32

本文为您介绍如何在构建计划设置中使用图形化编辑器。

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. 进入目标项目后，单击左侧的**持续集成**，单击**构建计划 > 文本编辑器**并在其中进行编排。

功能介绍

通过命令行文编辑 Jenkinsfile（构建过程描述文件）是最基础人机交互模式，CODING 在基于文本编辑的核心功能上设计了图形化编辑视图，能够兼容命令行编辑中的大部分自定义操作。创新性实现了构建过程边写边看、能够为用户带来所见即所得的直观构建过程编辑体验。
在**构建计划设置 > 流程配置**中选择**图形化编辑器**进行使用。

构建流程中的概念

不论图形化或文本类型，编辑器本质上都是用来方便用户查看与编辑构建流程的核心—— Jenkinsfile（过程描述文件），因此在讨论编辑器之前需理解“过程描述文件”的几个重要概念。
本篇文档主要聚焦于声明式文件的语法规则。

流水线

流水线是可自定义的工作模型，它定义了交付软件的完整流程，一般包含构建、测试和部署等阶段。

执行环境

执行环境描述了整个流水线执行过程或者某个阶段的执行环境，必须出现在描述文件顶格或者每一个阶段里。

是否必须	是
参数列表	见下文
允许的位置	必须出现在描述文件顶格或者每一个阶段里

阶段

一个阶段定义了一系列紧密相关的步骤。每个阶段在整条流水线中各自承担了独立、明确的责任。例如“构建阶段”、“测试阶段”或“部署阶段”。通常来讲，所有的实际构建过程都放置在阶段里面。

是否必须	至少一个
参数列表	一个强制的字符串类型参数，用以指明阶段名称
允许的位置	在阶段（stage）区块内部

阶段列表

阶段列表包含了一系列的阶段，一个阶段列表最少包含一个阶段。流水线里必须要有且仅有一个阶段列表。

是否必须	是
参数列表	无
允许的位置	在流水线（pipeline）内只能出现一次

步骤列表

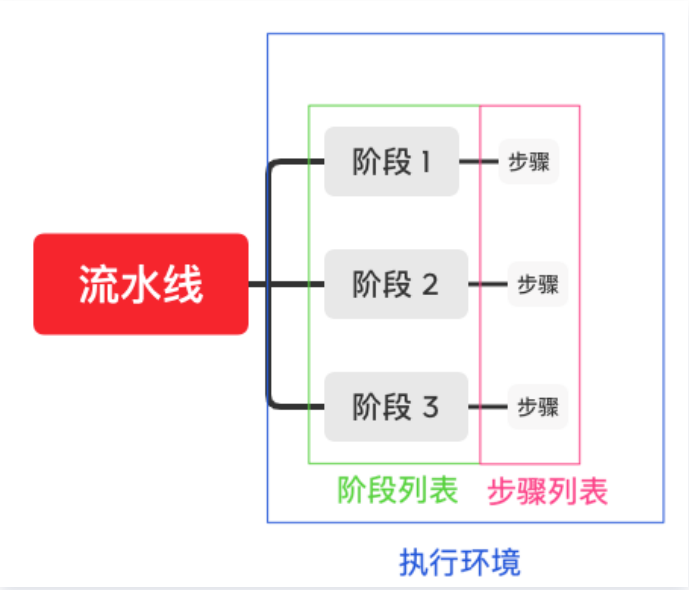
步骤列表描述了一个阶段内具体要做什么事，具体要执行什么命令。例如有一个步骤（step）需要系统打印一条“构建中...”的消息，即执行命令 echo '构建中...'。

是否必须	是

参数列表	无
允许的位置	在每一个阶段（stage）块内

并行

并行用来声明一些并行执行的阶段，通常适用于阶段与阶段之间不存在依赖关系的情况下，用来加快执行速度。注意任何含并行区块下的阶段不能再设置执行环境。



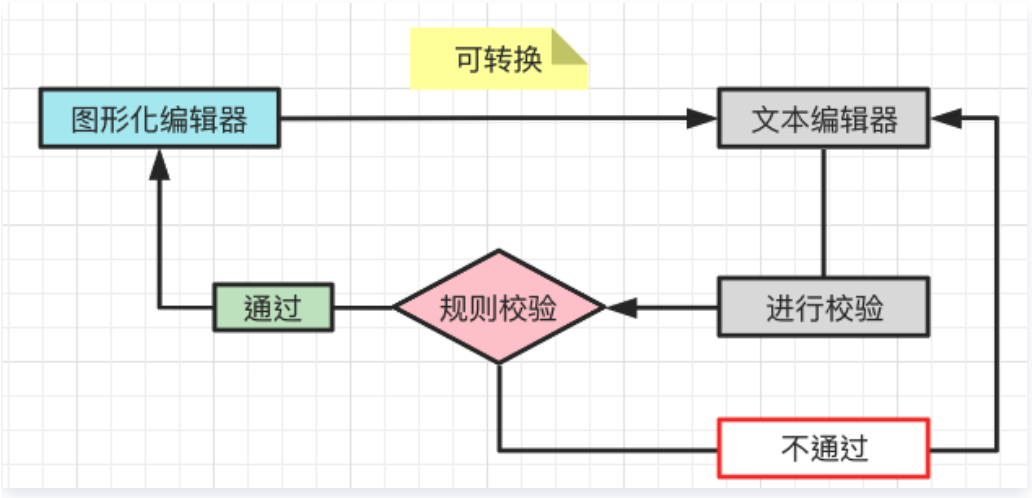
示例文件

```
pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
        sh 'ci-init'
        checkout([$class: 'GitSCM', branches: [[name:
env.GIT_BUILD_REF]],
                                    userRemoteConfigs: [[url:
env.GIT_REPO_URL]]])
      }
    }
    stage('构建') {
      steps {
        echo '构建中...'
        sh 'make'
        echo '构建完成.'
      }
    }
```

```
    stage('测试') {
      steps {
        echo '单元测试中...'
        sh 'make check'
        junit 'reports/**/*.xml'
        echo '单元测试完成.'
      }
    }
    stage('部署') {
      steps {
        echo '部署中...'
        sh 'make publish'
        echo '部署完成'
      }
    }
  }
}
```

编辑器间转换

图形化编辑器本质上是预设好的代码文本，所以能够无缝转变为文本编辑器。反之则不行，因为文本编辑器上增删的代码文本必须经过“规则校验”的判定程序，通过后才能转换为可编辑视图。



就自定义操作范围而言，文本编辑器所支持的范围比图形化编辑器更大。因图形化编辑器预设了大量常用的步骤，因此适用模式、标准化的工作；而文本编辑器没有限制，仅需满足 Jenkins 语法要求即可，适用于执行具体而特定的任务。

配置构建计划
触发规则

最近更新时间：2024-08-02 15:23:01

本文为您介绍如何在构建计划设置中设置触发规则。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

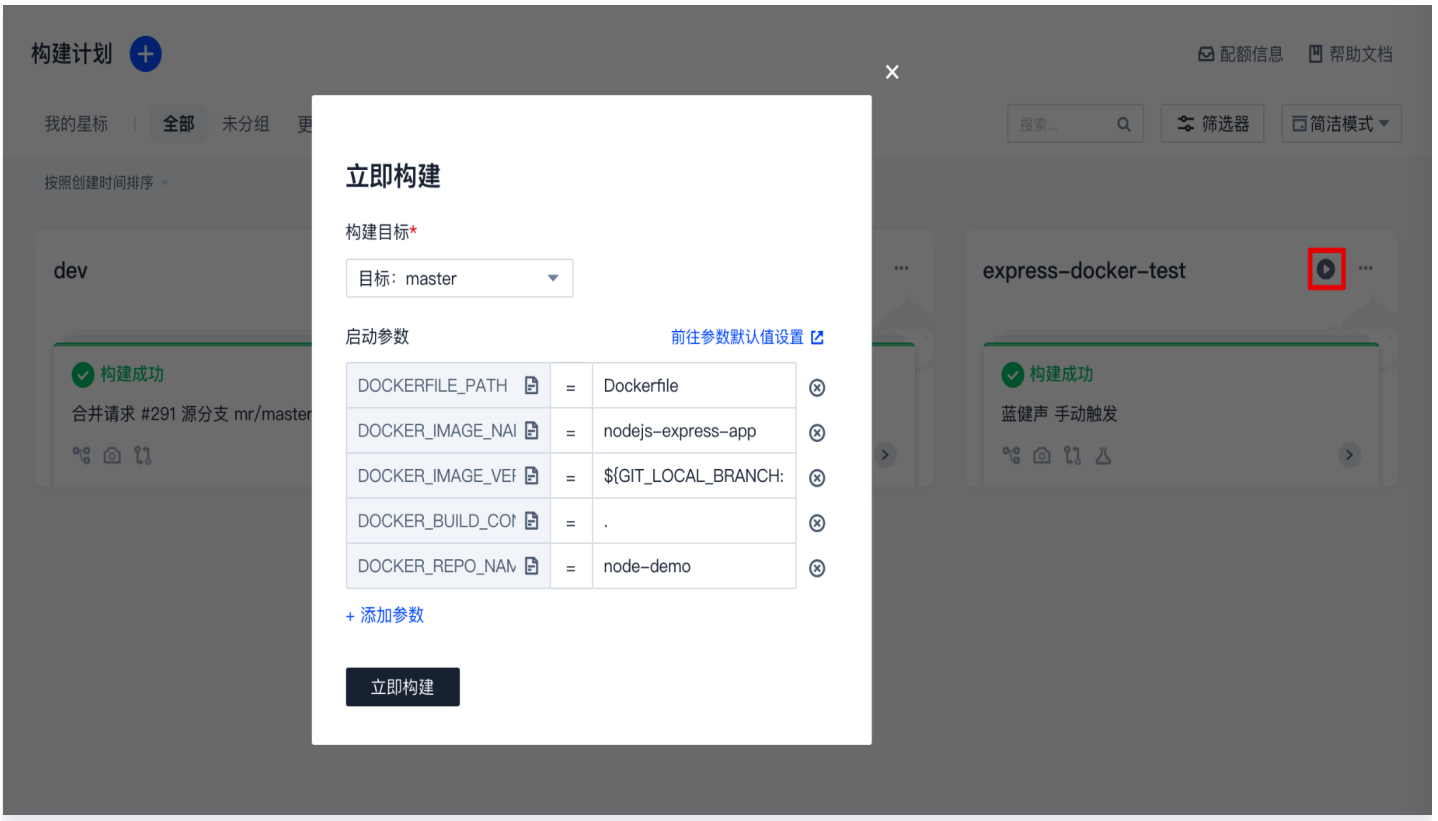
在持续集成计划的配置过程中，您可以按需设置构建计划运行的触发规则，触发规则中包含了构建计划运行的频率与触发的条件。每一个持续集成的构建计划，都会支持以下几种触发方式：

- 手动触发。
- 代码源触发。
- 定时触发。
- API 触发。

上述的多种触发方式可以同时使用。

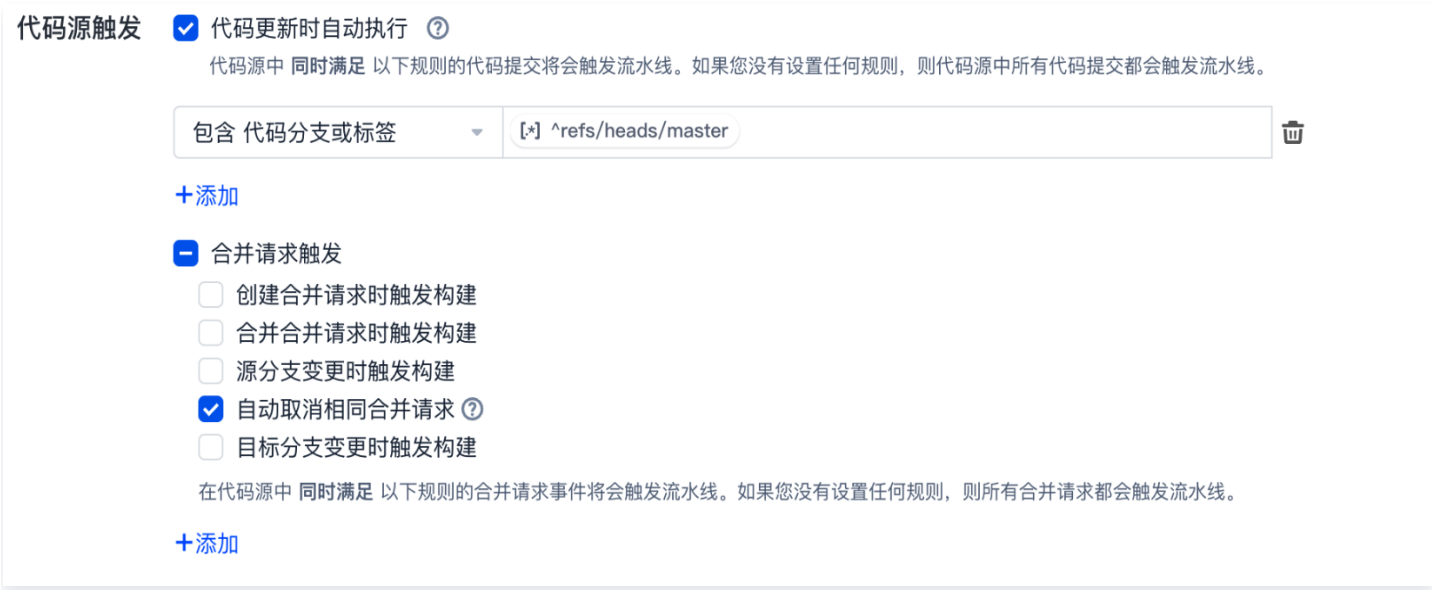
手动触发

您可以主动触发一个构建计划，手动触发时，可输入对应的构建参数，构建参数将以环境变量的形式加入到构建环境中。
在构建计划页面，单击**立即构建**，在弹框中按需选择构建目标（标签、分支、修订版本），输入需要的构建参数完成触发构建。



代码源触发

配置了代码源触发的构建计划，会自动监听本构建计划中所选择的代码仓库，根据其变化来自动触发构建计划。



代码更新

若选择了**代码更新时自动执行**，却未指定任何代码源规则，则任何代码提交均会触发流水线。

如需设置精细化的构建触发条件，可通过组合规则进行约束。多个规则之间为「and」的关系，同时满足才会触发构建。目前持续集成提供以下类型代码提交触发规则：

- **包含代码分支或标签**：只有指定代码分支或标签更新代码时，才会触发构建。
- **排除代码分支或标签**：除指定代码分支或标签以外的情况更新代码时，才会触发构建。
- **包含文件路径**：只有指定文件路径更新代码时，才会触发构建。
- **排除文件路径**：除指定文件路径以外的路径更新代码时，才会触发构建。
- **包含创建人**：只有指定成员更新代码时，才会触发构建。
- **排除创建人**：除指定成员以外的成员更新代码时，才会触发构建。

其中，指定代码分支或标签支持正则匹配 ref 全称或者简称：

1. 如 refs/heads/master 和 master 都能匹配 master 分支触发。
2. 如希望 master 和 dev 更新时才触发构建可使用：`^refs/heads/(master|dev)` 。

合并请求

通过合并请求触发持续集成可以模拟源分支合并入目标分支后可能发生的场景，尽早暴露问题。您可以选择在何种情况下的合并请求会执行构建：

- 发起合并请求时触发。
- 归并合并请求时触发。
- 合并请求的源分支发生变更。

  表示持续集成任务将监听源分支的变动情况。假设一个合并请求的源分支是 `issue/new` 分支，那么构建计划将监听 `issue/new` 分支，有新的代码提交时将自动触发持续集成任务。
- 自动取消相同合并请求。
- 合并请求的目标分支发生变更。

  表示持续集成任务将监听目标分支的变动情况。假设一个合并请求的目标分支为 `master` 分支，那么构建计划将监听 `master` 分支，代码发生变更时将会自动触发持续集成任务。

若勾选了上述配置，却未指定任何代码源规则，则任何合并请求均会触发流水线。

如需设置精细化的构建触发条件，可通过组合规则进行约束。多个规则之间为「and」的关系，同时满足才会触发构建。目前持续集成提供以下类型合并请求触发规则：

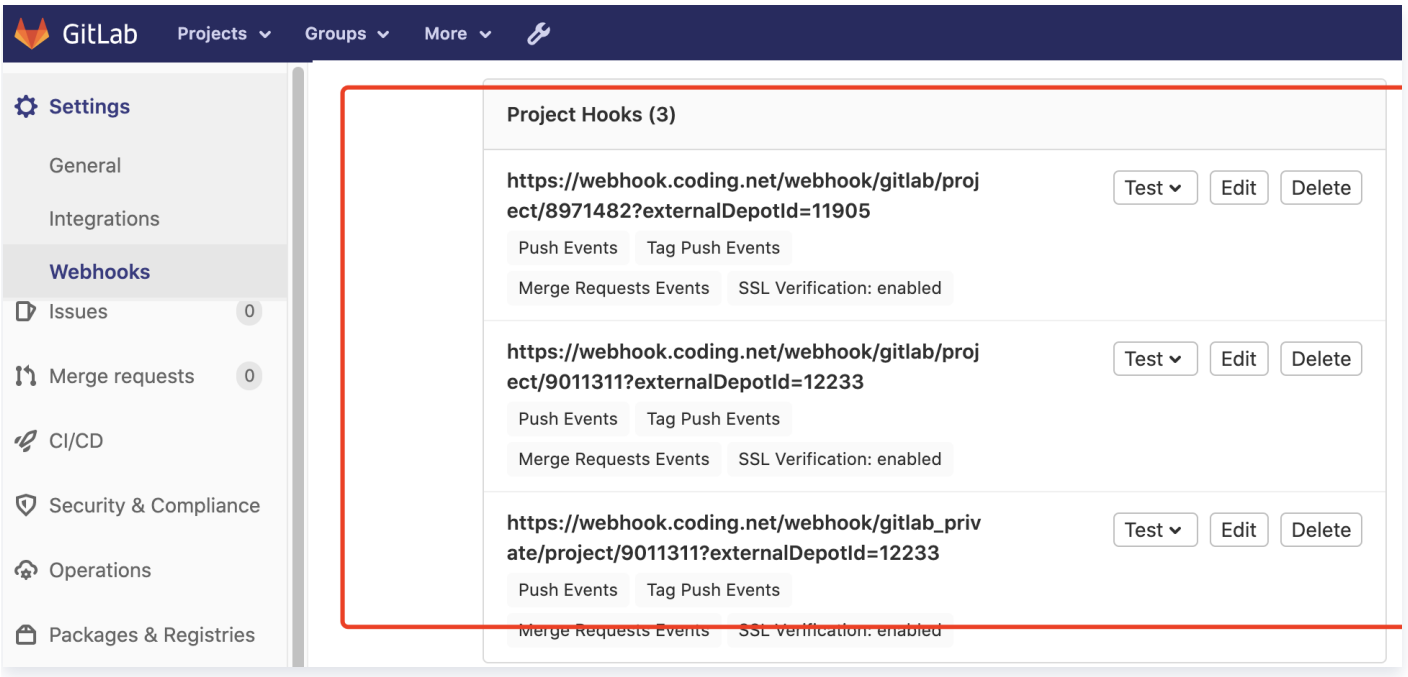
- **包含代码分支**：包含指定代码分支（支持正则表达式匹配）的合并请求才会触发构建。
- **排除代码分支**：除指定代码分支以外的合并请求才会触发构建。
- **包含文件路径**：合并请求中包含指定文件路径时，才会触发构建。
- **排除文件路径**：除指定文件路径外的合并请求才会触发构建。
- **包含创建人**：来自指定成员的合并请求才会触发构建。
- **排除创建人**：来自除指定成员以外的成员的合并请求才会触发构建。

自动取消相同构建

您可以在设置中勾选是否**自动取消相同版本号**以及**自动取消相同合并请求**所触发的构建（仅保留最新一个）。

GitLab 私有云

绑定私有 GitLab 时，会自动创建 GitLab Webhook，后续事件自动通知 CODING，然后匹配上述触发规则设置。



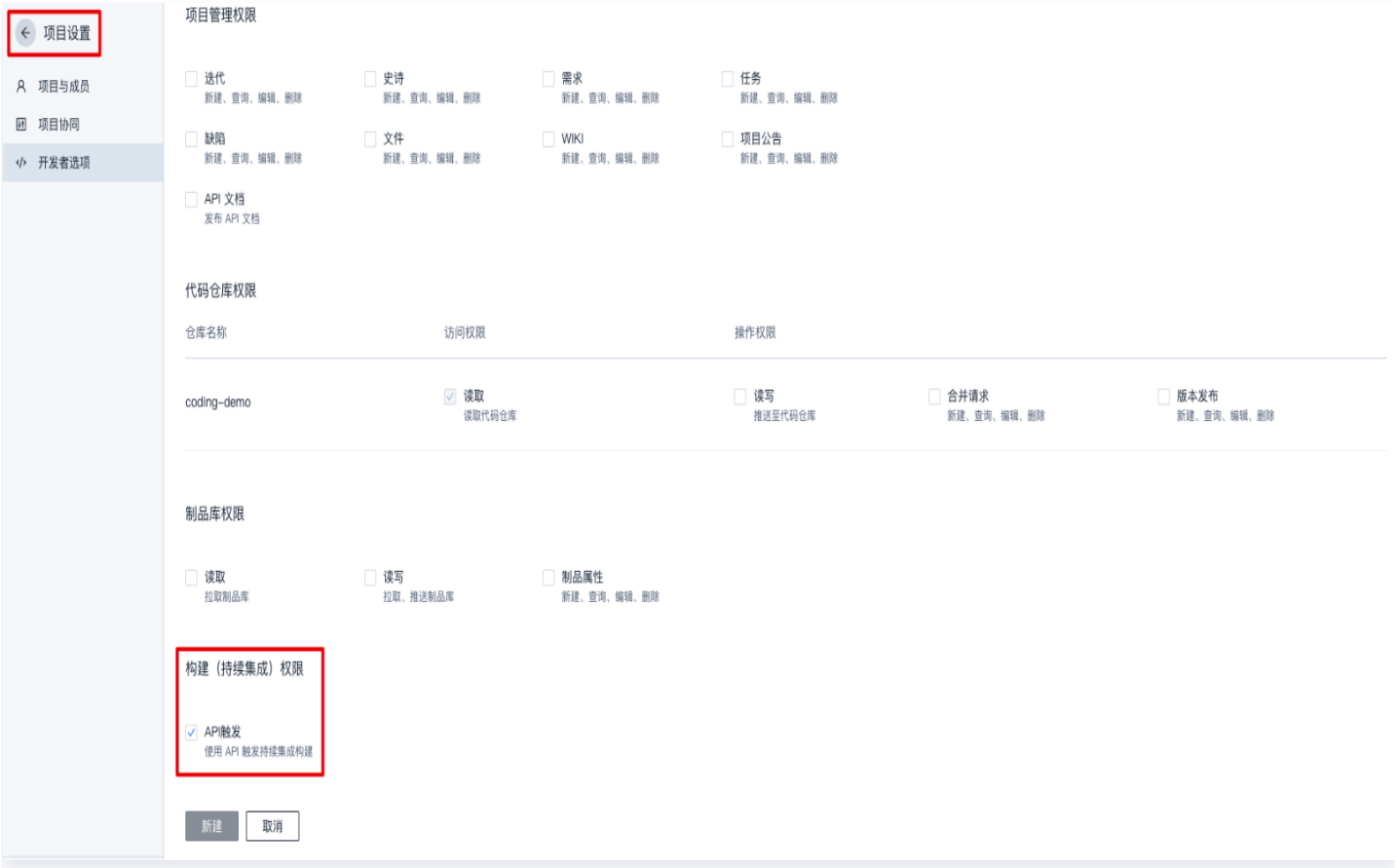
定时触发

通过给构建计划添加定时触发配置，您可以周期性或在某个具体的时间点，自动触发一个构建计划，产生一个具体的构建任务。
您可以为一个构建计划添加多个定时触发，没有前后优先级之分，多个定时触发有时间重合的，依然会触发多次构建。

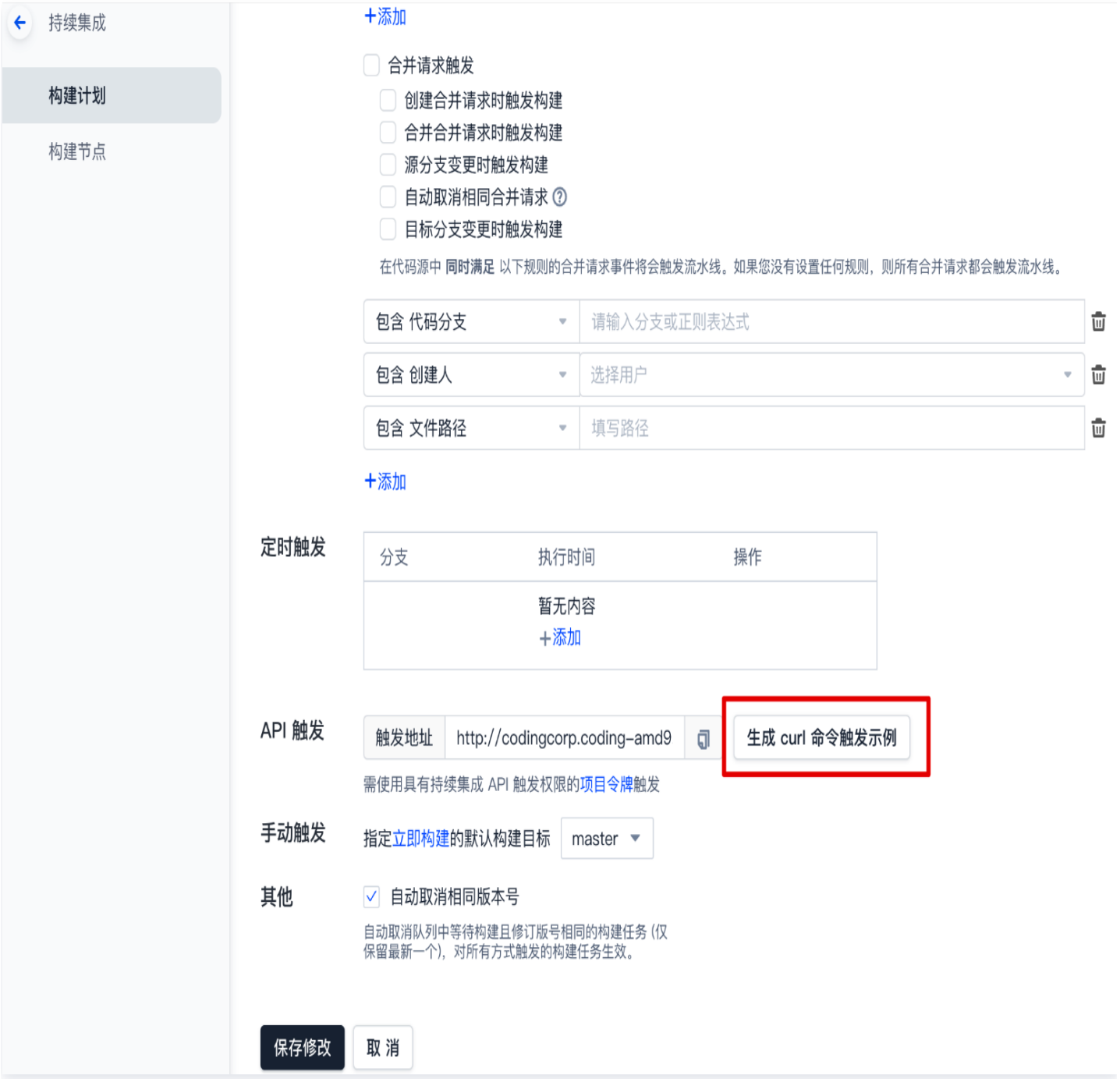
设置项	说明
选择分支	若选中分支代码与上次触发对比无变化，即使到达触发时间，也不会触发定时任务进行构建。
日期选择	您可以选择一周内的多个日期。
触发方式	• 周期触发：您可以选定00:00 – 24:00之间的任意时间为周期（精确到小时），按照选中的间隔触发任务。 • 单次触发：您可以在00:00 – 24:00之间选择任意时间为触发时间点（精确到分钟）。

API 触发

在使用此项功能之前，请确保您已经在**项目设置 > 开发者选项 > 项目令牌 > 新建令牌**中生成了具备持续集成 API 触发权限的令牌。



生成具备相应权限的令牌后便能够调用构建计划中的 API 触发接口。单击生成 curl 命令触发示例后即可生成相应的调用命令。



API 说明

项目令牌调用 CODING 持续集成 API 时所使用的认证方式为 `Basic Auth`，下面是关于 API 接口的详细信息和相关参数。

触发构建任务

```
POST https://< TEAM_GK >.coding.net/api/cci/job/< JOB_ID >/trigger
```

请求 body

```
{
  "ref": "master",
  "envs": [
    {
      "name": "my-params-1",
      "value": "hello",
      "sensitive": 1
    },
    {
      "name": "my-params-2",
      "value": "world",
      "sensitive": 0
    }
  ]
}
```

返回 body

```
{
  "code": 0
}
```

参数说明

参数名字	参数位置	是否必填	类型	默认值	说明
ref	body	否	string	master	构建目标的 ref（`commit sha / tag / branch`），若构建计划不使用代码仓库可以忽略
envs	body	否	env[]	-	构建计划的启动参数

envItem

参数名字	参数位置	是否必填	类型	默认值	说明
name	envItem.name	是	string	master	构建计划的启动参数的名称

value	envItem.value	否	string	-	构建计划的启动值
sensitive	envItem.sensitive	否	number	0	是否将启动参数设置为保密,设置保密后日志中不可见。 1 为保密，0 为明文

环境变量

最近更新时间：2024-08-02 15:23:01

本文为您介绍如何在构建计划设置中设置环境变量。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

持续集成过程中，我们总会将一些配置（如：账号密码/版本号等）信息以环境变量的形式注入到构建过程中。CODING 持续集成支持多种环境变量使用形式，您可以同时使用以下几种方式来为构建过程注入环境变量，其优先级为从上到下（排在前面的配置优先级最高）：

- Jenkinsfile 中的 withEnv。
- Jenkinsfile 中的 environment。
- 构建计划（Job）中的启动参数。
- 构建计划（Job）设置中的环境变量。
- 构建过程中系统内置的环境变量。

下文将详细介绍这几种方式的详细说明。

withEnv 与 environment

1. 您可以在 Jenkinsfile 中使用 environment 来定义环境变量（如下所示）：

```
pipeline {
    agent any
    environment {
        MY_PROJECT = 'project-1'
        MY_TEAM    = 'team-1'
    }
    stages {
        stage('Build') {
            steps {
```

```
                echo "MY_PROJECT is ${MY_PROJECT}"
                echo "MY_TEAM is ${MY_TEAM}"
                // 输出内容如下所示：
                // MY_PROJECT is project-1
                // MY_TEAM is team-1
            }
        }
    }
}
```

2. 在构建过程中，可能需要在不同的阶段使用同名的环境变量。可以使用 withEnv 来针对部分操作设置环境变量，避免全局的环境变量污染。 withEnv 中所执行的 step ，都将优先使用 withEnv 设置的环境变量。具体效果可以参考以下例子：

```
  pipeline {
    agent any
    environment {
        MY_PROJECT = 'project-1'
        MY_TEAM    = 'team-1'
    }
    stages {
        stage('Build') {
            steps {

                echo "MY_PROJECT is ${MY_PROJECT}"
                echo "MY_TEAM is ${MY_TEAM}"
                // 输出内容如下所示：
                // MY_PROJECT is project-1
                // MY_TEAM is team-1

                // withEnv 中设置的环境变量只对作用域下的 step 有效，优先级高
于 environment
                withEnv(['MY_PROJECT=project-2']) {

                    echo "MY_PROJECT is ${MY_PROJECT}"
                    echo "MY_TEAM is ${MY_TEAM}"
                    // 输出内容如下所示：
                    // MY_PROJECT is project-2
                    // MY_TEAM is team-1

                }
            }
```

```
            }
        }
    }
```

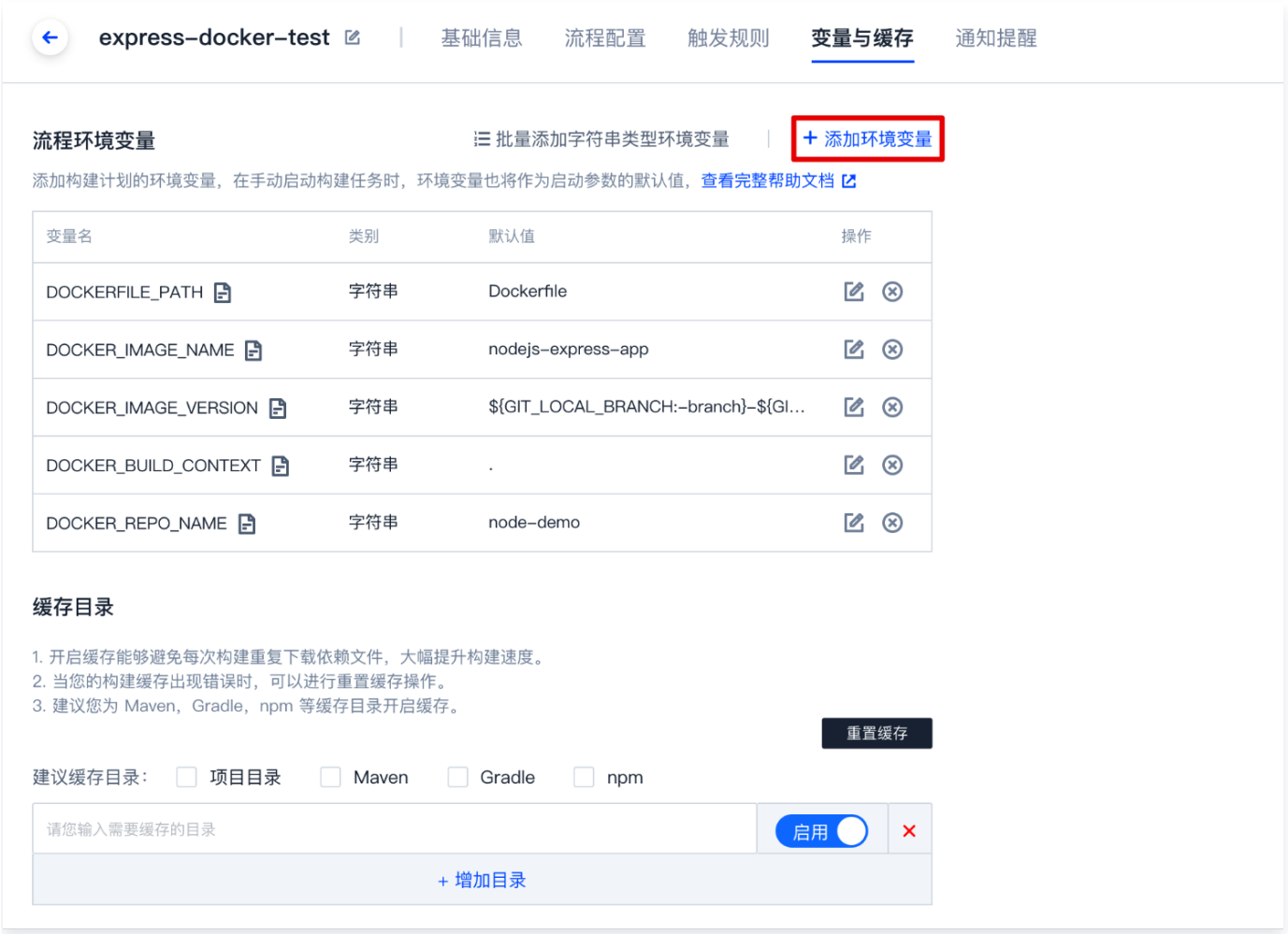
如果您想了解更多 Jenkinsfile 环境变量相关内容，请参见 使用环境变量 。

构建计划中的启动参数

优先级仅次于 Jenkinsfile 中配置的环境变量，您可以在启动构建计划时，选择或填写对应的环境变量值。

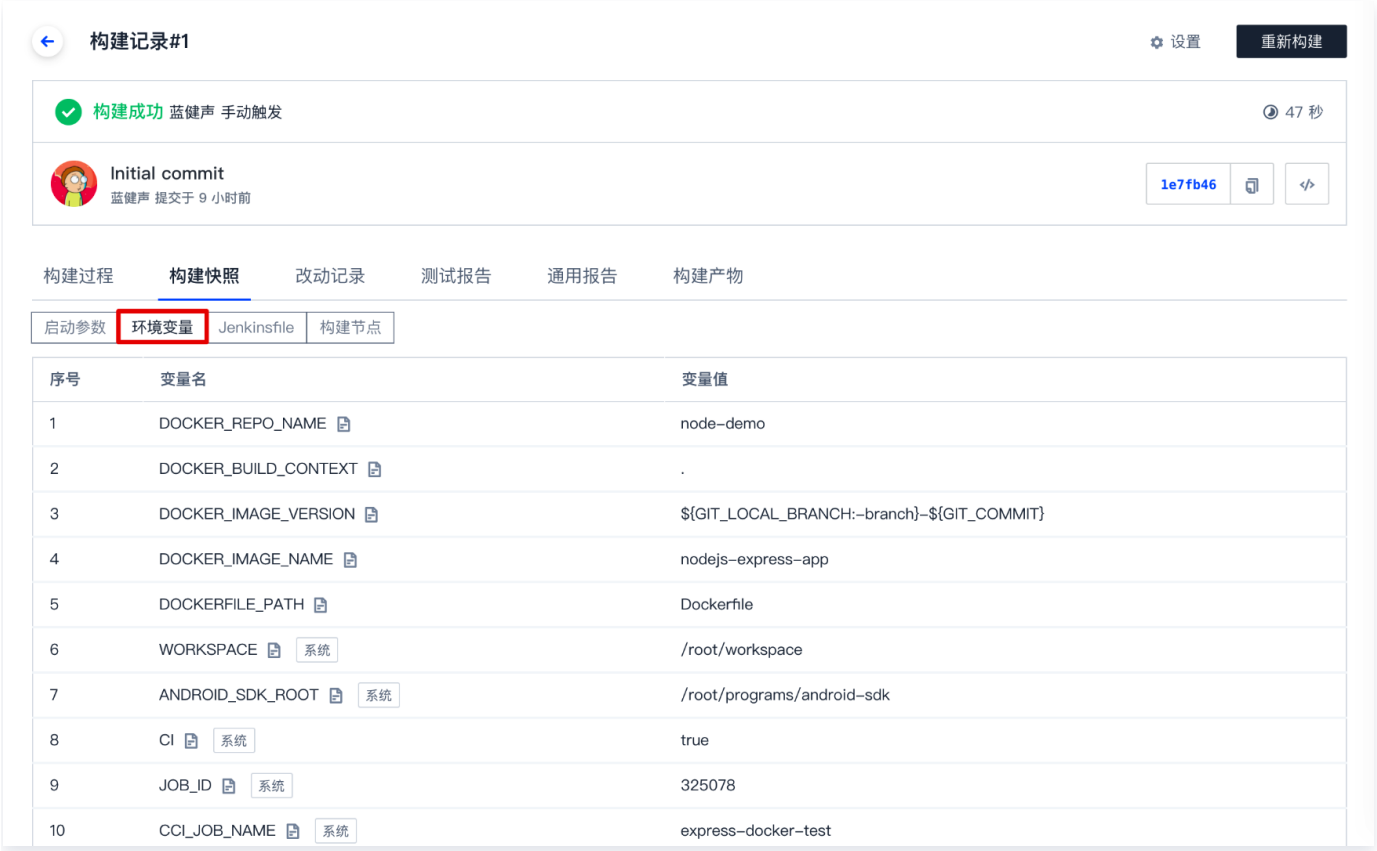
添加环境变量

除了在 Jenkinsfile 中硬编码环境变量，还可以在构建计划的配置中进行设置。目前 CODING 支持添加以下四种类别的环境变量：字符串、单选、多选、CODING 凭据。您还可以将构建计划中的环境变量配置视为启动参数的默认值。



系统内置的环境变量

CODING 持续集成在构建过程中，会为每一个构建任务注入对应的环境变量，默认注入的环境变量列表您可以在构建快照中查看：



所有环境变量汇总如下，按照不同的触发规则（代码更新时触发、定时触发、合并请求时触发）进行分类介绍：

序号	变量名	变量含义	代码更新时触发	定时触发	合并请求时触发
1	CREDENTIALS_ID	部署私钥凭据 CredentialsId 用于拉取仓库	✓	✓	✓
2	DOCKER_REGISTRY_CREDENTIALS_ID	docker 私钥凭据 CredentialsId（等同于 CODING_ARTIFACTS_CREDENTIALS_ID）	✓	✓	✓
3	CODING_ARTIFACTS_CREDENTIALS_ID	制品库私钥凭据 CredentialsId 用于拉取项目内的制品库	✓	✓	✓
4	GIT_HTTP_URL	HTTPS 协议代码仓库地址	✓	✓	✓
5	GIT_BUILD_REF	构建对应的 Git 修订版本号	✓	✓	✓
6	GIT_DEPLOY_KEY	代码仓库的部署公钥	✓	✓	✓

7	GIT_COMMIT	当前版本的修订版本号	✓	✓	✓
7	GIT_COMMIT_SHORT	修订版本号的前 7 位	✓	✓	✓
8	GIT_PREVIOUS_COMMIT	前一个构建运行编号的修订版本号	✓	✓	✓
9	GIT_AUTHOR_EMAIL	本版本最新提交作者邮箱	✓	✓	✓
10	GIT_SSH_URL	协议代码仓库地址	✓	✓	✓
11	GIT_COMMITTER_NAME	本版本最新提交者名称	✓	✓	✓
12	GIT_AUTHOR_NAME	本版本最新提交作者名称	✓	✓	✓
13	REF	要构建的版本	✓	✓	✓
14	GIT_PREVIOUS_SUCCESSFUL_COMMIT	前一个构建运行成功的修订版本号	✓	✓	✓
15	GIT_COMMITTER_EMAIL	本版本最新提交者名称	✓	✓	✓
16	GIT_BRANCH	触发构建的分支	✓	✓	✓
17	GIT_URL	仓库 SSH 协议地址	✓	✓	✓
18	GIT_LOCAL_BRANCH/BRANCH_NAME	本地分支名称	✓	✓	✓
19	FETCH_REF_SPECS	git 要检出的 refs	✓	✓	✓
20	GIT_REPO_URL	仓库 SSH 地址	✓	✓	✓
21	JOB_ID	构建计划 ID	✓	✓	✓
22	JOB_NAME	构建计划名称	✓	✓	✓
23	CI_BUILD_NUMBER	构建编号	✓	✓	✓
24	PROJECT_ID	项目 ID	✓	✓	✓

25	PROJECT_NAME	项目名称	✓	✓	✓
26	PROJECT_WEB_URL	项目网页地址	✓	✓	✓
27	PROJECT_API_URL	项目后端 API 地址	✓	✓	✓
28	PROJECT_TOKEN	项目令牌密码用于读取项目	✓	✓	✓
29	PROJECT_TOKEN_GK	项目令牌用户名	✓	✓	✓
30	GIT_TAG	触发构建的 Git 标签 (仅在使用标签构建的时候才会有)	✓	–	–
31	DEPOT_NAME	当前使用的代码仓库名称	✓	–	–
32	CCI_CURRENT_PROJECT_COMMON_CREDENTIALS_ID (即将上线)	内置项目令牌的 CredentialsId	✓	–	–
33	CCI_CURRENT_TEAM（即将上线）	当前构建环境的企业名，如：myteam.coding.net 中的 myteam	✓	–	–
34	CCI_CURRENT_DOMAIN（即将上线）	当前构建环境的域名，如：myteam.coding.net 中的 coding.net	✓	–	–
35	MR_RESOURCE_ID	合并请求 ID	–	–	✓
36	MR_TARGET_BRANCH	合并请求目标分支名	–	–	✓
37	MR_TARGET_SHA	合并请求目标分支版本号	–	–	✓
38	MR_MERGED_SHA	模拟合并完的版本号	–	–	✓
39	MR_SOURCE_BRANCH	合并请求源分支名	–	–	✓
40	MR_STATUS	合并请求状态	–	–	✓
41	MR_SOURCE_SHA	合并请求源分支版本号	–	–	✓

构建快照

最近更新时间：2023-09-11 16:05:23

本文为您介绍如何使用构建快照功能。

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

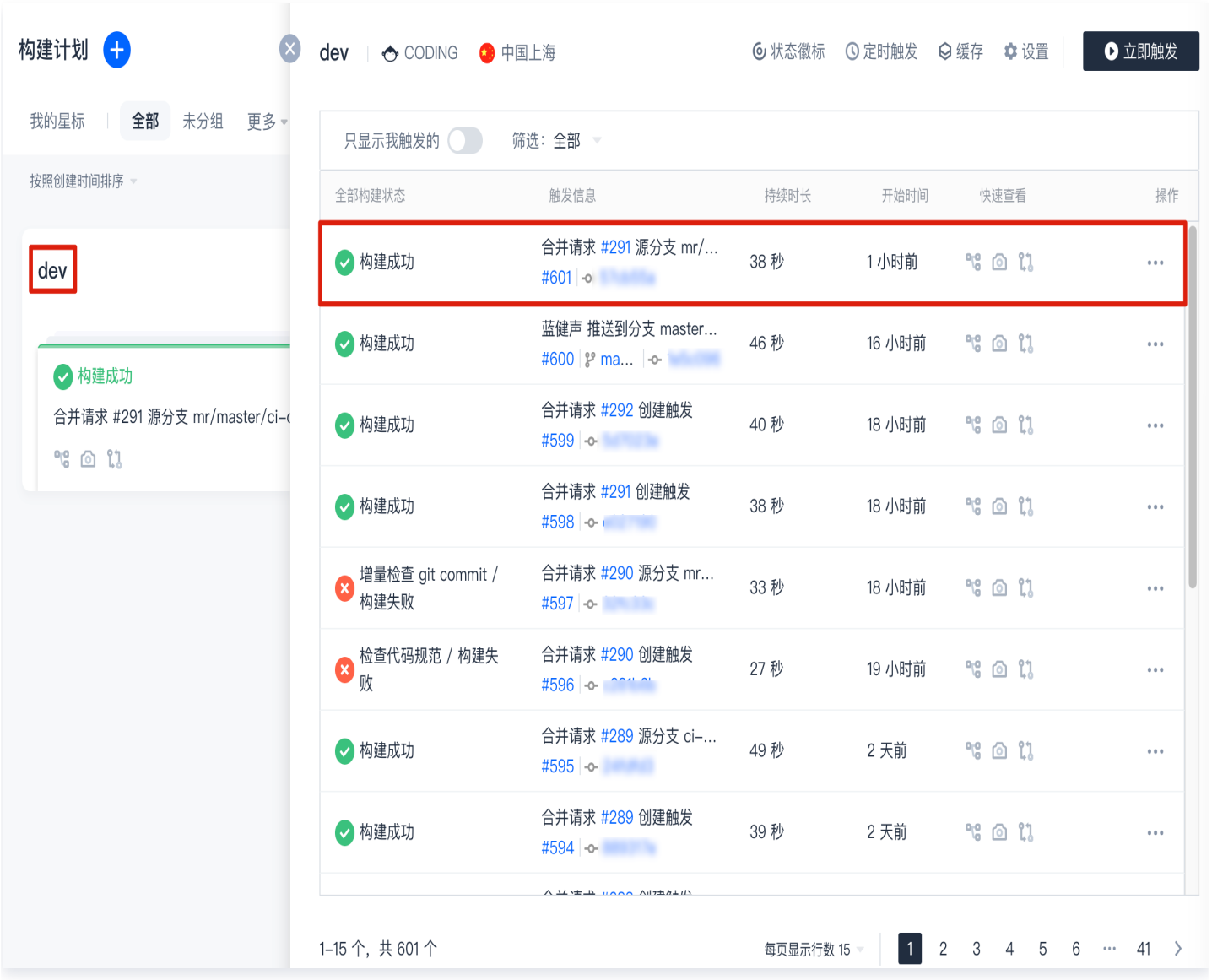
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

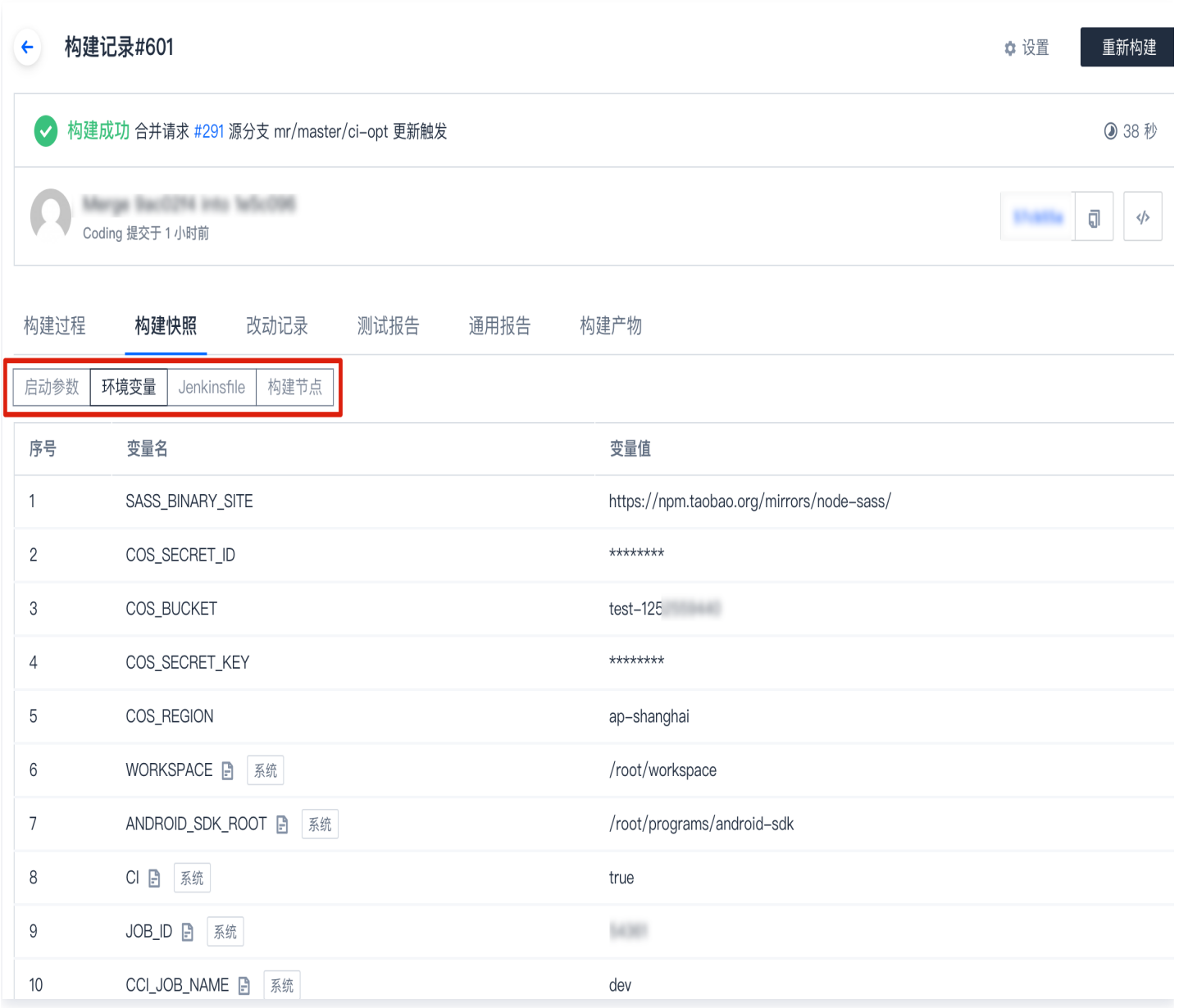
您在持续集成的每一个构建任务，都有可能使用到不同的配置文件或构建参数，为了方便您回顾构建任务的执行过程，CODING 持续集成提供了构建任务的构建快照查看功能。构建快照功能让您能清晰地了解到每次构建记录的配置参数。

查看构建配置

1. 在项目中单击**持续集成 > 构建计划**，单击单次构建计划的标题即可查看该计划的所有构建记录，单击进入任一一条构建记录：



2. 在构建记录中，单击构建快照就可以看到每次构建记录的配置快照：启动参数、环境变量、流程配置文件。



启动参数

1. 启动参数是您在启动构建任务时，输入的参数内容，会以环境变量的形式注入到构建任务的运行环境中。

2. 在构建完成后，您可以在构建快照中看到配置的启动参数。

序号	变量名	变量值
1	DOCKER_IMAGE_VERSION	${GIT_LOCAL_BRANCH:-branch}-${GIT_COMMIT}
2	DOCKER_IMAGE_NAME	logo-reg
3	DOCKERFILE_PATH	Dockerfile
4	DOCKER_REPO_NAME	build
5	DOCKER_BUILD_CONTEXT	.

环境变量

1. 这里仅包含启动任务时配置的环境变量，不包含所有在运行过程中产生或者动态设置的环境变量。

2. 在环境变量选项卡内，用户可以参看到任务启动时，系统和用户为构建任务设置的环境变量。

流程配置文件

通过流程配置选项卡，您可以看到此次构建记录使用的配置文件（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()
          }
        }
```

缓存目录

最近更新时间：2024-06-12 11:59:41

本文为您介绍如何使用缓存目录功能。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

本地项目在按安装依赖包时会把下载的文件缓存起来，以供下次安装使用。例如使用 `npm install` 命令后会在项目中生成 `./node_modules` ，缓存储存在 `~/.npm` 目录，后者体积更小，更通用。

- 默认构建节点

  CODING 会为每个构建计划自动分配计算资源，构建完毕即销毁，每次构建都会自动重新分配一台构建节点，因此需要指定缓存目录用于加速下次构建。
- 自定义构建节点

  若选择自行接入计算资源，并在构建计划中选择通过自定义构建节点执行任务，那么构建完毕不会销毁服务器，故无需指定缓存目录。

如果在持续集成中使用 Docker 时需要把缓存目录挂载至 Docker 中。

默认构建节点

1. CODING 为构建计划提供基础的任务计算资源，每次任务都会分配一台云服务器，构建环境为 Linux 系统、分配 root 用户权限，缓存目录如下：

包管理工具	缓存目录
Maven	/root/.m2/
Gradle	/root/.gradle/
npm	/root/.npm/
composer	/root/.cache/composer/
yarn	/usr/local/share/.cache/yarn/

2. 您可以在构建计划设置中的**变量与缓存**中勾选缓存目录，如果未找到目标目录还支持自行录入。

Docker 构建环境

若在构建计划中使用 Docker 环境，那么需先行前往**变量与缓存**中勾选缓存目录，再挂载至 Docker 中。

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('Java 缓存') {
      agent {
        docker {
          image 'adoptopenjdk:11-jdk-hotspot'
          args '-v /root/.gradle/:/root/.gradle/ -v
/root/.m2/:/root/.m2/'
          reuseNode true
        }
      }
      steps {
        sh './gradlew test'
      }
    }
    stage('npm 缓存') {
      steps {
        script {
          docker.image('node:14').inside('-v /root/.npm/:/root/.npm/') {
            sh 'npm install'
          }
        }
      }
    }
  }
}
```

自定义构建节点

在自定义构建节点中使用 Docker 环境时需按照服务器用户名找到对应的缓存目录，例如当 Ubuntu 服务器的默认用户名为 ubuntu 时，缓存目录为 `/home/ubuntu/.npm/`，那么对应的代码为：

```
docker.image('node:14').inside('-v /home/ubuntu/.npm/:/root/.npm/') {
  sh 'npm install'
}
```

缓存 Docker 基础镜像

1. 如果每次构建都需要拉取 Docker 基础镜像，例如 `Dockerfile` 基础镜像、CI agent 镜像，那么通常会耗费大量时间，此时就可以通过缓存进行加速。

   参考以下 `Jenkinsfile`，修改镜像名称即可复用：

```
pipeline {
  agent any
  environment{
    DOCKER_CACHE_EXISTS = fileExists '/root/.cache/docker/php-8.0-
cli.tar'
  }
  stages {
    stage('加载缓存') {
      when { expression { fileExists(DOCKER_CACHE_PATH).equals(true) }
}
      steps {
        sh 'docker load -i /root/.cache/docker/php-8.0-cli.tar ||
true'
      }
    }
    stage('使用镜像（请修改此段）') {
      agent {
        docker {
          image 'php:8.0-cli'
          args '-v /root/.cache/:/root/.cache/'
          reuseNode 'true'
        }
      }
      steps {
        sh "php -v"
      }
    }
    stage('生成缓存（仅运行一次）') {
      when { expression { DOCKER_CACHE_EXISTS == 'false' } }
      steps {
        sh 'mkdir -p /root/.cache/docker/'
        sh 'docker save -o /root/.cache/docker/php-8.0-cli.tar
php:8.0-cli'
      }
    }
  }
}
```

2. 在缓存目录处增加 `/root/.cache/` 路径，此时第二次的构建耗时明显更短，说明缓存已生效：

> ⚠ **注意：**
> 缓存镜像会逐渐过时，建议定时清除，与官方更新保持一致。

保存 Dockerfile

在持续集成中使用 Dockerfile 作为构建环境，需要运行 `docker build` 命令用以初始化，较为不便。可以将已构建的 Docker 镜像保存到仓库，方便二次拉取复用。

Jenkinsfile

```
// 创建 CODING Docker 制品库，获取用户名、密码和仓库地址
sh "docker login -u $DOCKER_USER -p $DOCKER_PASSWORD my-team-
docker.pkg.coding.net"

// 使用 Dockerfile 的 md5 做 tag
md5 = sh(script: "md5sum Dockerfile | awk '{print \$1}'", returnStdout:
true).trim()
imageFullName = "my-team-docker.pkg.coding.net/my-project/my-repo/my-
app:dev-${md5}"

// 检查镜像是否已存在远端仓库
dockerNotExists = sh(script: "docker manifest inspect $imageFullName >
/dev/null", returnStatus: true)
def testImage = null
if (dockerNotExists) {
    testImage = docker.build("$imageFullName", "--build-arg
APP_ENV=testing ./")
    sh "docker push $imageFullName"
} else {
    testImage = docker.image(imageFullName)
```

```
}

// 使用镜像进行自动化测试
testImage.inside("-e 'APP_ENV=testing'") {
    stage('test') {
        echo 'testing...'
        sh 'ls'
        echo 'test done.'
    }
}
```

代码解释：在 shell 中执行下列命令，通过返回值可以判断镜像是否已经存在。

```
$ docker manifest inspect ecoding/foo:bar
no such manifest
$ echo $?
1
```

缓存也是加速构建的最好方式之一，但是不同工具和语言的方案各不相同，接下来将会枚举几种常见的场景和加速方案。

默认节点使用缓存

对于持续集成默认节点来说，如果直接在宿主机上进行构建，Maven 和 Gradle 会将构建的依赖下载到 /root/.m2 和 /root/.gradle 中， Npm 也是同理的依赖会下载到 /root/.npm 中。

Docker 自定义构建环境使用缓存

您可以参考 Dockerfile 实践教程 。该文档是 Docker 的官方文档，文档中提出了一些措施帮助我们编写优秀的 Dockerfile。

对 FROM 的镜像进行抽象

如果您有多个服务，每个 Dockerfile 里面都需要 apt-get 安装一些工具，此时您应该抽象出您的 FROM 镜像安装好必要的工具，将其作为 FROM 镜像给 Dockerfile 使用。对于团队来说，基础镜像的管理是非常有必要的，不光可以减少构建耗时，还可以统一的修复安全漏洞，增加内置的工具，可以很大程度上减少开发和运维同学的维护成本。

自定义构建环境使用缓存

持续集成除了使用官方提供的构建环境， 还可以使用 Docker 自定义构建环境 。自定义构建环境就相当于 docker run 出了一个 container，构建的指令在这个环境中执行。
举例： 假如需要用 Java18 和 Maven 构建程序：

```
pipeline {
  agent {
    docker {
      reuseNode 'true'
      registryUrl 'https://coding-public-docker.pkg.coding.net'
      image 'public/docker/openjdk:18-2022'
      args '-v /root/.m2:/root/.m2 -v /usr/bin/docker:/usr/bin/docker -v
/var/run/docker.sock:/var/run/docker.sock'
    }
  }

  stages {
    stage('检出') {
      steps {
        checkout([
          $class: 'GitSCM',
          branches: [[name: env.GIT_BUILD_REF]],
          extensions: [[$class: 'CloneOption', depth: 1, noTags: false,
shallow: true]],
          userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId:
env.CREDENTIALS_ID]]
        ])
      }
    }
    stage('编译') {
      steps {

        sh "mvn clean install"
      }
    }

    stage('push ') {
      steps {
        sh "echo password | docker login xxx-docker.pkg.coding.net -u
xxx  --password-stdin  "
        sh "docker build -t myjava:latest ."
        sh "docker push myjava:latest "
      }
    }
  }
}
```

请注意到

```
args '-v /usr/bin/docker:/usr/bin/docker -v
/var/run/docker.sock:/var/run/docker.sock -v /root/.m2:/root/.m2 '
```

- `-v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock`，将 docker 和 docker.sock 挂载到自定义环境的容器中，可以在后续的 stage 和 step 中使用 docker。
- `-v /root/.m2:/root/.m2`，将宿主机的 /root/.m2 和自定义构建环境的容器的 /root/.m2 进行一个映射，在自定义构建环境容器中执行 mvn clean install 下载的 maven 一来会下载到容器的 /root/.m2。
- 勾选上变量与缓存中 Maven，这样下次构建就会利用上这个缓存。

构建节点
构建节点类型

最近更新时间：2025-01-20 14:00:52

本文为您介绍构建节点的类型。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

当您在使用 CODING 持续集成进行构建时，本质上是调用计算资源作为**构建节点**完成构建任务。你可以选择使用官方默认提供的云计算资源或自行接入自定义构建节点两种方式运行构建任务。

默认构建节点

默认节点内置了构建环境，其中预装了开发语言 SDK、命令行工具等服务，请参见 默认节点环境 。

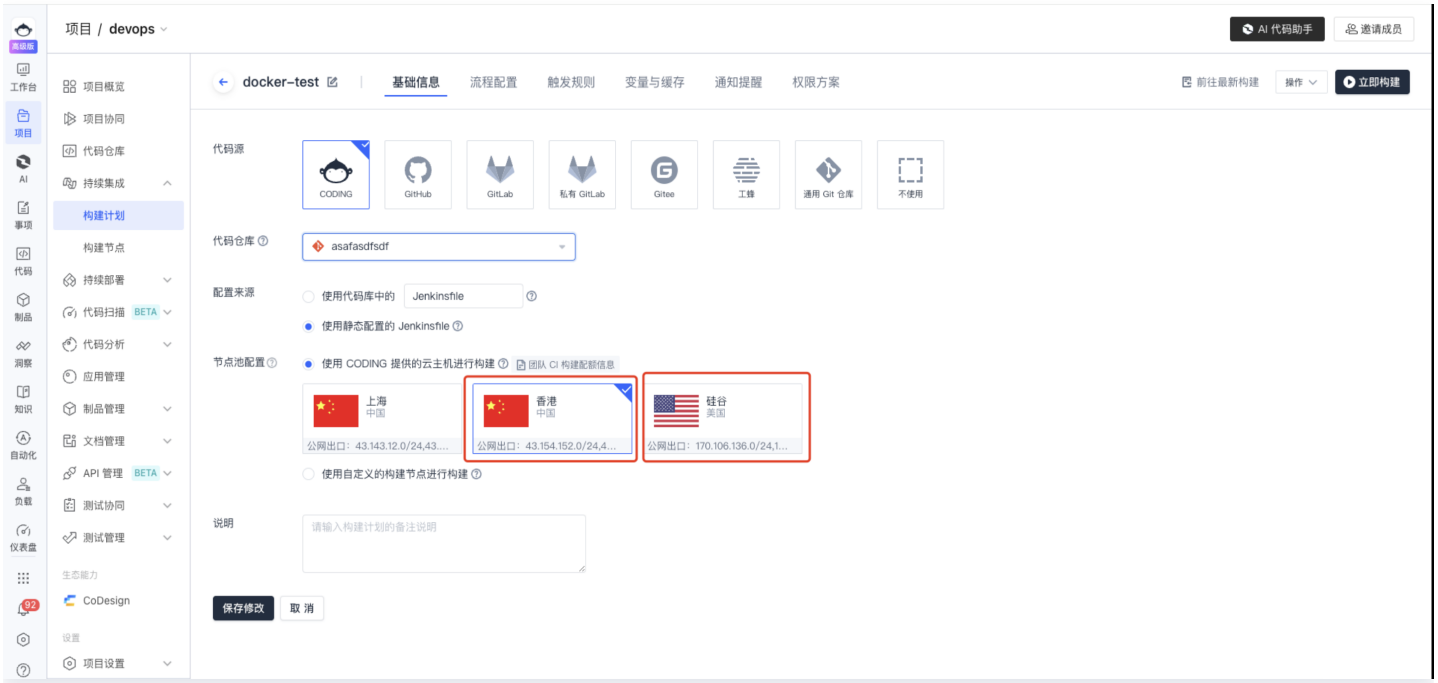
出口 IP 地址

默认构建节点所使用的出口 IP 地址如下：

中国上海	中国香港	美国硅谷
• 43.143.12.0/24 • 43.142.234.0/24 • 43.142.23.0/24 • 124.220.180.0/24 • 81.68.101.0/24 • 111.231.92.0/24 • 101.33.207.0/24 • 43.136.72.0/24	• 124.156.164.25/32 • 119.28.15.65/32	• 170.106.136.17/32 • 170.106.83.77/32

切换香港或硅谷为构建节点

进入**构建计划 > 基础信息**，可以手动切换节点池配置。



自定义构建节点

在实际的开发项目中，所涉及的开发环境可能是多种多样的，当官方节点的构建环境无法承载项目的持续集成要求时，例如需要使用 macOS Xcode 构建 iOS 应用时，就可以通过接入自定义类型节点（物理机/虚拟机/容器等）运行特定任务。

更多关于自定义构建节点的内容请参见 自定义节点 。

默认构建环境

最近更新时间：2023-09-20 11:11:01

本文为您介绍默认节点的构建环境。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

构建任务由构建节点执行，而构建环境指的便是构建节点中内置的系统底层环境，预装了开发语言 SDK、命令行工具等服务。
构建环境有以下类别：

- 默认环境
- 自定义构建环境

若开发项目对运行环境有特定要求，如 swift 项目需要在 macOS 环境下运行，可以参见 自定义节点 自行接入构建节点。

默认环境

在构建计划的开始环节中选择构建环境。

对应的 `Jenkinsfile` 为 `agent any`：

```
pipeline {
  agent any
  stages  {
    stage("检出") {...}
    stage("检查代码规范") {...}
  }
}
```

CODING 云主机为 Ubuntu 系统，预装了以下 SDK 和命令行工具：

SDK	命令行工具
• android-sdk: 26.1.1 • build-essential • dotnet-core: 2.2 • elixir: 1.8.1 • erlang: Erlang/OTP 21 • go: 1.14.4 • java: 1.8.0_191 • nodejs: 10 • php: 8.0、7.4、7.3 • python3/pip3: 3.9、3.8、3.7 • python: 2.7.12	• bundler: 1.17.2 • cmake: 3.5.1 • composer：1.10.8 • coscmd：1.8.5.36 • docker-compose: 1.26.0 • docker：20.10.6 • git-lfs: 2.7.2 • git：2.28.0 • gradle: 7.0.2 • helm: 2.13.1 • jq: 1.5-1-a5b5cbe

- ruby: 2.6.0

- kubectl: 1.18.4
- maven: 3.6.3
- mercurial: 3.7.3
- pigz: 2.3.1
- rancher: 2.2.0
- rvm: 1.29.7
- sshpass: 1.05
- svn: 1.9.3
- tccli: 3.0.67.1
- vsftpd: 3.0.3
- yarn: 1.15.2
- axcl: 2.5

说明：
预装的软件版本有限且定期升级，而各个项目所需要的版本可能不同。

自定义构建环境

最近更新时间：2024-08-09 14:56:01

本文为您介绍如何自定义构建环境。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

公共节点自定义版本

在持续集成中可自行下载安装软件的各种版本，如果官网下载较慢，我们亦提供了 镜像服务 以供下载。如需增加制品，欢迎提交至 开源项目 。

Go

```
stage('Go') {
  steps {
    // 建议设置「缓存目录」 /root/.cache/downloads
    sh 'rm -rf /root/programs/go'
    dir ('/root/.cache/downloads') {
      sh 'wget -nc "https://coding-public-
generic.pkg.coding.net/public/downloads/go-linux-amd64.tar.gz?
version=1.17.3" -O go-linux-amd64-1.17.3.tar.gz | true'
      sh 'tar -zxvf go-linux-amd64-1.17.3.tar.gz -C /root/programs'
    }
    sh 'go version'
  }
}
```

Helm

```
stage('Helm') {
  steps {
    dir ('/root/.cache/downloads') {
```

```
      sh 'wget -nc "https://coding-public-
generic.pkg.coding.net/public/downloads/helm-linux-amd64.tar.gz?
version=v3.7.1" -O helm-linux-amd64-v3.7.1.tar.gz | true'
      sh "tar -zxvf helm-linux-amd64-v3.7.1.tar.gz -C \$HELM_BIN linux-
amd64/helm --strip-components 1"
    }
    sh 'helm version'
  }
}
```

kubectl

```
stage('kubectl') {
  steps {
    dir ('/root/.cache/downloads') {
      sh 'wget -nc "https://coding-public-
generic.pkg.coding.net/public/downloads/kubectl-linux-amd64?
version=v1.22.4" -O kubectl-linux-amd64-v1.22.4 | true'
      sh 'cp kubectl-linux-amd64-v1.22.4 /usr/local/bin/kubectl'
    }
    sh 'chmod +x /usr/local/bin/kubectl'
    sh 'kubectl version --client'
  }
}
```

Node.js

```
stage('Node.js') {
  steps {
    sh 'rm -rf /usr/lib/node_modules/npm/'
    dir ('/root/.cache/downloads') {
      sh 'wget -nc "https://coding-public-
generic.pkg.coding.net/public/downloads/node-linux-x64.tar.xz?
version=v16.13.0" -O node-v16.13.0-linux-x64.tar.xz | true'
      sh 'tar -xf node-v16.13.0-linux-x64.tar.xz -C /usr --strip-
components 1'
      // sh 'wget -nc "https://coding-public-
generic.pkg.coding.net/public/downloads/node-linux-x64.tar.xz?
version=v14.18.2" -O node-v14.18.2-linux-x64.tar.xz | true'
      // sh 'tar -xf node-v14.18.2-linux-x64.tar.xz -C /usr --strip-
components 1'
      // 更多版本：v12.22.7、v17.2.0
```

```
    }
    sh 'node -v'
  }
}
```

PHP

```
pipeline {
  agent {
    docker {
      reuseNode 'true'
      registryUrl 'https://coding-public-docker.pkg.coding.net'
      image 'public/docker/php:8.0'
      // image 'public/docker/php:7.4' 以及 7.3、7.2、7.1、5.6
      args '-v /var/run/docker.sock:/var/run/docker.sock -v
/usr/bin/docker:/usr/bin/docker'
    }
  }
  stages {
    stage('安装依赖') {
      steps {
        // Possible values for ext-name:
        // bcmath bz2 calendar ctype curl dba dom enchant exif fileinfo
filter ftp gd gettext gmp
        // hash iconv imap interbase intl json ldap mbstring mysqli oci8
odbc opcache pcntl pdo
        // pdo_dblib pdo_firebird pdo_mysql pdo_oci pdo_odbc pdo_pgsql
pdo_sqlite pgsql phar posix pspell
        // readline recode reflection session shmop simplexml snmp soap
sockets sodium spl standard
        // sysvmsg sysvsem sysvshm tidy tokenizer wddx xml xmlreader
xmlrpc xmlwriter xsl zend_test zip
        sh 'apt-get update && apt-get install -y libbz2-dev'
        sh 'docker-php-ext-install bz2'
        sh 'php -i | grep bz2'
      }
    }
  }
}
```

Docker 环境

CODING 持续集成为您提供了默认构建环境，若默认环境中预装的 SDK 版本和命令行工具无法满足您的要求，还可以通过在持续集成中使用 Docker 构建环境来解决。您可以通过以下方式使用 Docker 构建环境：

- CODING 官方提供的镜像。
- 使用已托管至项目级制品库的 Docker 镜像。
- 适用于项目层级的标准构建环境，保障项目内镜像安全，方便管理，通过项目令牌您也可以拉取其他项目的镜像。
- 使用指定 Registry 地址（默认为 Docker Hub）的 Docker 镜像。
- 使用 Dockerfile 脚本构建环境。

CODING Docker 镜像

在**持续集成计划设置 > 流程配置 > 基础配置 > 图形化编辑器**中，选择**使用 CODING 官方 Docker 镜像**，例如 Node.js 14：

对应的 `Jenkinsfile` 参考：

```
pipeline {
  agent {
    docker {
      reuseNode 'true'
      registryUrl 'https://coding-public-docker.pkg.coding.net'
      image 'public/docker/nodejs:14'
    }
  }
  stages {
```

```
    stage('Test') {
      steps {
        sh 'node --version'
      }
    }
  }
}
```

项目制品库 Docker 镜像

以下内容以配置了进程管理工具 pm2 的 Node.js 12 环境为例，分步骤演示如何将**自建镜像推送至制品仓库**以及**使用自建镜像**作为构建环境。

步骤一：构建 Docker 镜像

1. 新建目录，创建 Dockerfile 如下：

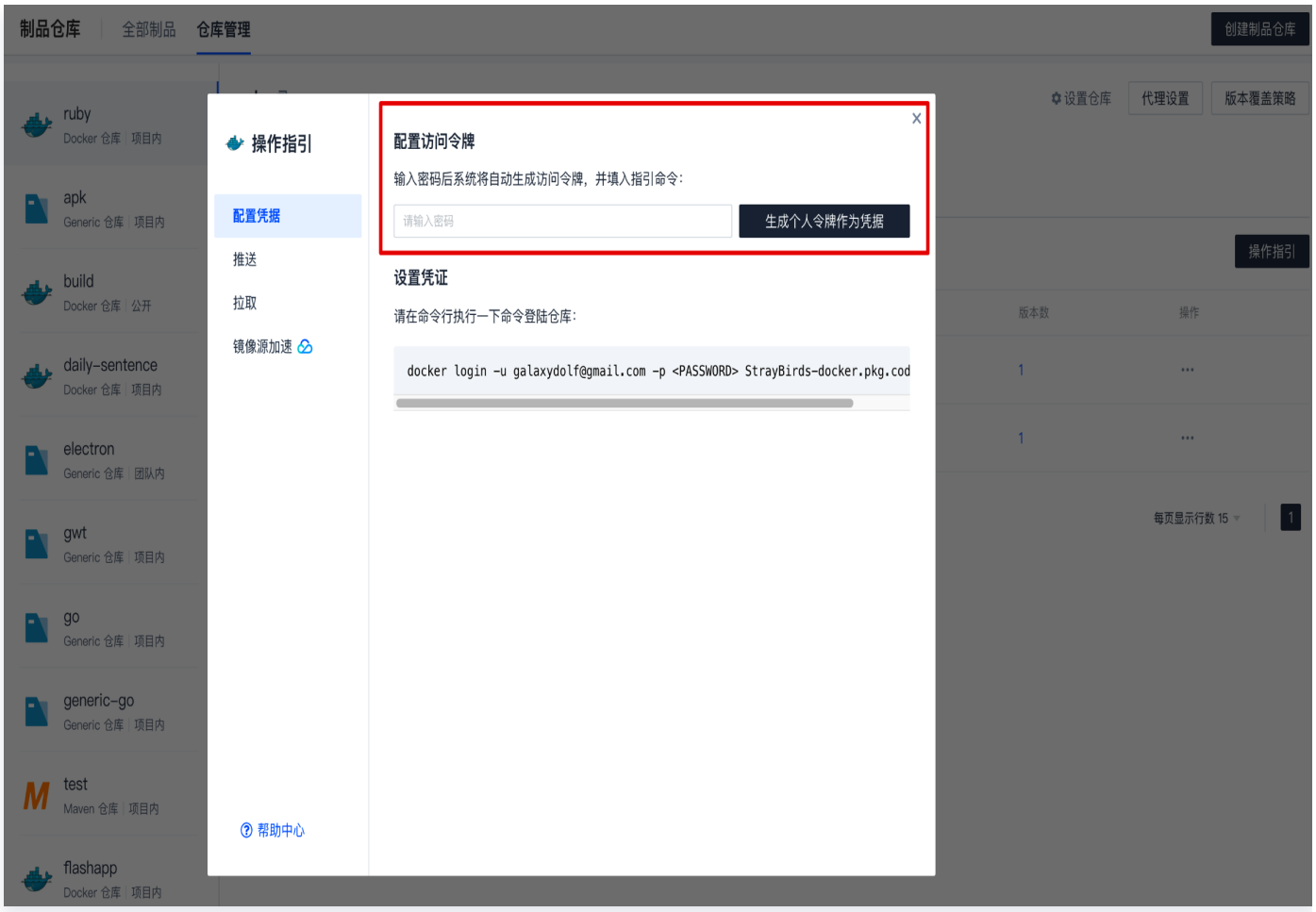
```
# 指定 node.js 版本为 node 12，默认从 Docker Hub 上拉取
FROM node:12
# 安装 pm2
RUN npm install pm2 -g

COPY . .
# 设置容器启动时的命令
CMD [ "pm2-runtime", "start" ]
```

2. 运行指令 `docker build -t pm2-test .`；`-t` 指定镜像名称。

```
Step 1/4 : FROM node:12
...
Step 2/4 : RUN npm install pm2 -g
...
Step 3/4 : COPY . .
...
Step 4/4 : CMD [ "pm2-runtime", "start" ]
 ---> Running in 46cc5081cb4f
Removing intermediate container 46cc5081cb4f
 ---> 5f8335fa91d4
Successfully built 5f8335fa91d4
Successfully tagged pm2-test:latest
```

步骤二：推送镜像到 CODING 制品库

1. 进入 CODING 制品库，选择已有制品库或新建制品库，输入密码后单击**生成个人令牌作为凭据**。



2. 复制后在终端输入命令进行登录。
3. 按照操作指引提示，为本地镜像打标签。

```
docker tag pm2-test ******/test-dd/test/pm2-test
```

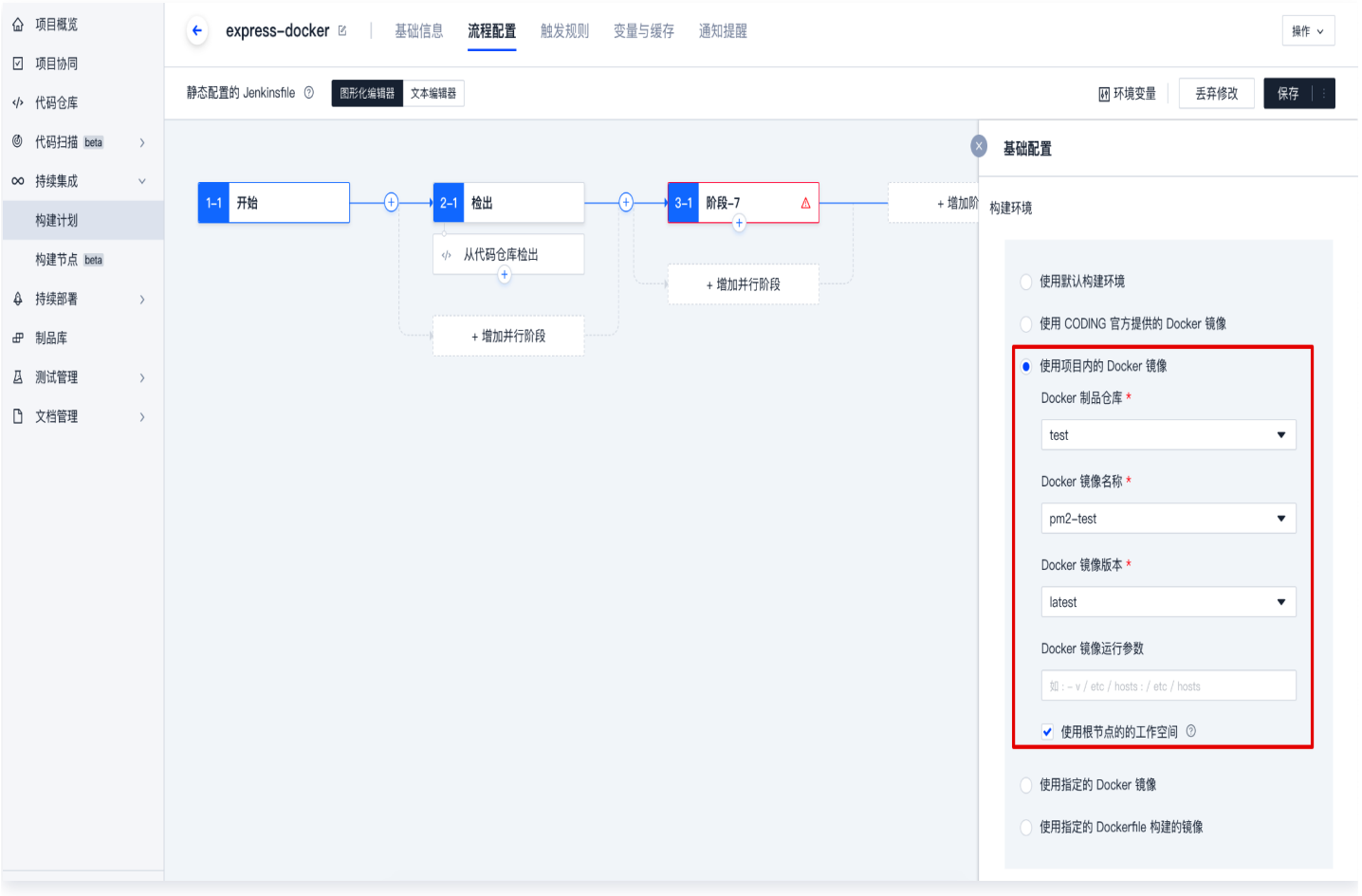
4. 推送您的 docker 镜像到 CODING 制品库。

```
docker push ******/test-dd/test/pm2-test
The push refers to repository [******************/test-dd/test/pm2-
test]
809e73e276b8: Pushed
9159d4abedcd: Pushed
...
latest: digest: sha256:ccecda5071e60593d1be44ea27d4ec5b35f6a5f6872fb9
size: 2634
```

5. 推送成功后可以在镜像列表找到您的镜像。

步骤三：在持续集成中使用镜像作为构建环境

进入**持续集成设置 > 流程配置**，选择使用项目内的 Docker 镜像，选择对应制品库和镜像。



指定地址的 Docker 镜像

Docker 镜像为必填项，需要填入您的镜像名称。**Registry 地址**需填写的格式为不带路径的 URL 地址，例如：

正确：https://codes-farm-docker.pkg.coding.net

错误：https://codes-farm-docker.pkg.coding.net/laravel-demo/laravel-docker/

若拉取私有镜像，需录入凭据后填写 Registry 认证凭据 ID。

对应的 Jenkinsfile：

```
pipeline {
  agent {
    docker {
      image 'node:14-alpine'
      reuseNode 'true'
    }
  }
  stages {
    stage('Test') {
      steps {
        sh 'node --version'
```

```
      }
    }
  }
}
```

Dockerfile 构建环境

若项目已经使用 Docker，建议将 `Dockerfile` 提交到代码库，用它作为持续集成构建环境。`Dockerfile` 示例代码：

```
FROM php:8.0-apache

RUN apt-get update \
  && apt-get install -y unzip
```

`Jenkinsfile`：

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        checkout([
          $class: 'GitSCM',
          branches: [[name: env.GIT_BUILD_REF]],
          userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId:
env.CREDENTIALS_ID]]
        ])
      }
    }
    stage('Use Docker') {
      agent {
        dockerfile {
          filename 'Dockerfile' // 可选，自定义 Dockerfile 文件名
          dir 'build' // 可选，Dockerfile 所在目录
          additionalBuildArgs  '--build-arg version=1.0.2' // 可选，
docker build 自定义参数
        }
      }
      stages {
        stage('Test') {
          steps {
```

```
            sh 'php -v'
            sh 'unzip -v'
          }
        }
      }
    }
  }
}
```

若构建次数频繁，而不想将时间浪费在 `docker build` 过程上，那么可以通过使用 Jenkins Dockerfile 保存镜像用于下次构建，从而节省大量时间，请参见 保存 Dockerfile 镜像 。

在阶段中使用 Docker

Jenkinsfile 参考：

```
pipeline {
  agent none
  stages {
    stage('Back-end') {
      agent {
        docker {
          image 'maven:3-alpine'
          reuseNode 'true'
        }
      }
      steps {
        sh 'mvn --version'
      }
    }
    stage('Front-end') {
      agent {
        docker {
          image 'node:14-alpine'
          reuseNode 'true'
        }
      }
      steps {
        sh 'node --version'
      }
    }
  }
}
```

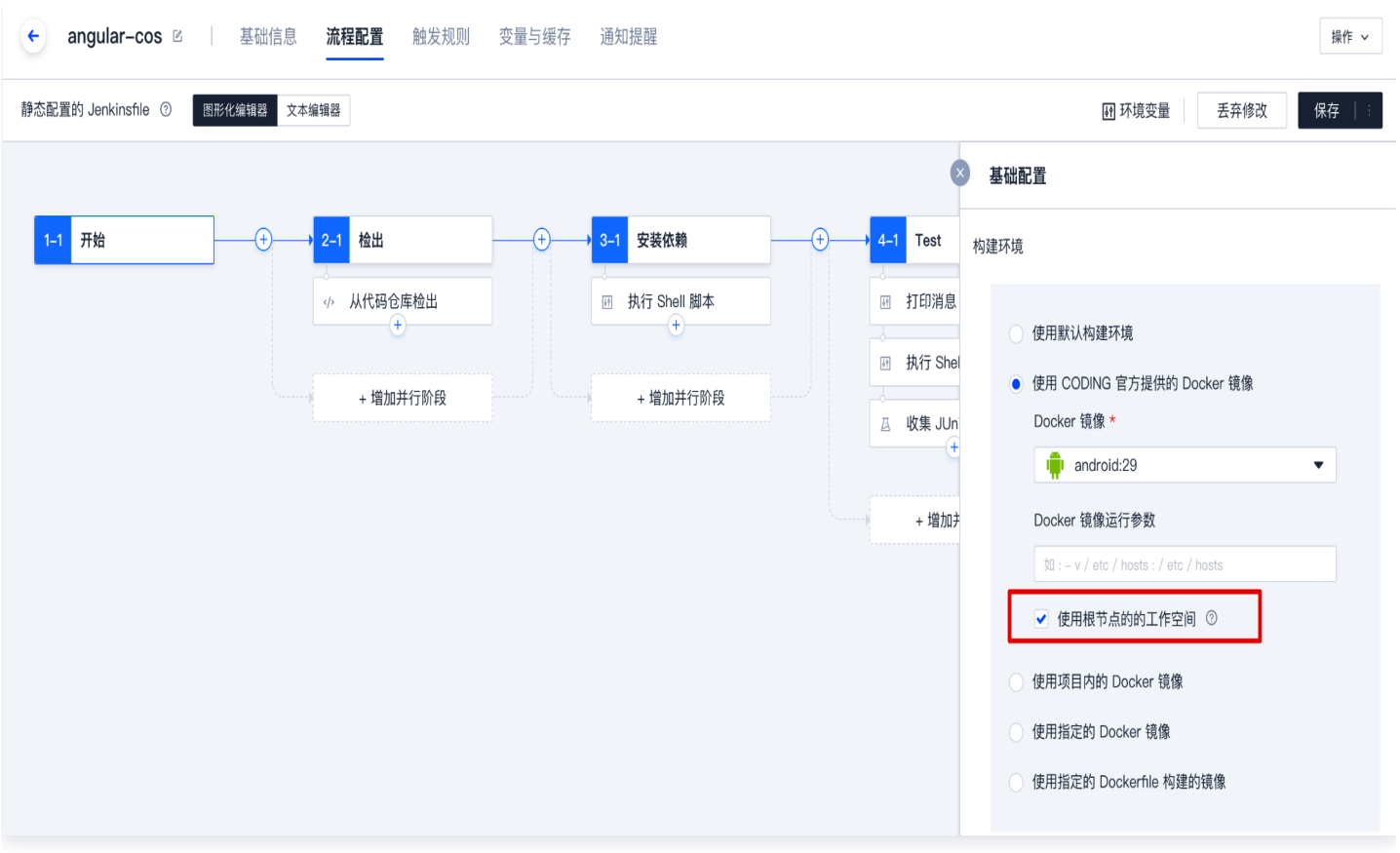
多个 Docker 后台

自动化测试往往需要临时的基础设施（例如 MySQL、Redis、Elasticsearch）。那么创建一个桥接网络，在其中启动多个 Docker 后台，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/')
    }
  }
}
```

根节点工作空间

将自定义 Docker 用作构建环境时，可以选择是否使用根节点的工作空间。勾选该选项后，当前阶段的 Docker 容器会和流水线在同一台构建节点中运行，可以获取流水线工作空间下根目录保存的所有文件。



对应的 `Jenkinsfile` 参数为 `reuseNode`，类型：Boolean，默认为 false：

```
pipeline {
  agent {
    docker {
      registryUrl 'https://coding-public-docker.pkg.coding.net'
      image 'public/docker/android:29'
    }
  }
  stages {
    // 代码被检出到 pipeline agent 的工作空间根目录下
    stage('检出代码') {
      steps {
        checkout([
          $class: 'GitSCM',
          branches: [[name: env.GIT_BUILD_REF]],
```

```
          userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId:
env.CREDENTIALS_ID]]
        ])
      }
    }
    stage('单元测试') {
      agent {
        dockerfile {
          // 默认在当前节点工作空间根目录下找名为 「Dockerfile」的文件构建环境
          filename 'Dockerfile'
          // 如果 reuseNode 为 false，则无法找到之前检出到 pipeline agent 的工
作空间根目录下的 Dockerfile
          reuseNode true
        }
      }
      steps {
        sh 'npm run test:ci'
        junit '*.xml'

      }
    }
  }
}
```

执行 Docker 命令

在 Jenkins Docker 环境中执行 `docker` 命令时，需要挂载外部虚拟机的 docker socket，否则会报错：`docker: command not found`。

```
pipeline {
  agent {
    docker {
      image 'ecoding/php:8.0'
      reuseNode 'true'
      // 挂载外部虚拟机的 docker socket
      args '-v /var/run/docker.sock:/var/run/docker.sock -v
/usr/bin/docker:/usr/bin/docker'
    }
  }
  stages {
    stage('自定义阶段') {
      steps {
        sh 'php -v'
```

```
      }
    }
    stage('构建 Docker 镜像') {
      steps {
        sh 'docker -v'
        script {
          docker.withRegistry("https://${env.CCI_CURRENT_TEAM}-
docker.pkg.coding.net", "${env.CODING_ARTIFACTS_CREDENTIALS_ID}") {
            //docker.build("foo:bar").push()
          }
        }
      }
    }
  }
}
```

若希望在自定义构建节点中运行 Docker 命令，在节点中先行安装 Docker 服务即可开始使用。

自定义节点

最近更新时间：2025-05-27 17:24:32

> **说明：**
> 由于安全合规原因，自2025年01月01日起我们已不再支持新增自定义构建节点的接入。现有自定义构建节点仍然可以继续正常运行。详情可 查看公告 。

本文为您介绍如何使用自定义节点。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

在实际的开发项目中，所涉及的开发环境可能是多种多样的。当默认节点的构建环境无法承载项目的运行要求时，例如需使用 macOS Xcode 构建 iOS 应用，那么就可以通过接入自定义类型节点（物理机/虚拟机/容器等）运行特定任务。
接入构建节点时需指定接入的构建节点池，构建节点不能游离节点池而单独存在。构建节点池 分为团队与项目两种类型。

> **注意：**
> 使用自定义节点相当于将您的构建机纳入 CODING 的管控范围。为了保障您的机器和内网的安全，请务必妥善保管您的用户名和密码，并建议开启登录二次认证。在必要时，您还需要制定构建机内网的安全策略，以确保您的机器和内网不受到未经授权的访问和攻击。

接入节点

接入自定义节点的过程本质上是在节点中运行 `Worker` 服务，单击了解 Worker 常用命令 。
目前支持 macOS、Windows、Linux 环境接入至构建计划节点池。

推荐配置

- CPU 8 核或以上。
- 内存 16 GB 或以上。

环境依赖

- **Python 3.6, Python 3.7, Python 3.8, Python 3.9**

  单击访问 项目地址 进行下载。
- **Git ⩾ 2.8**

  单击访问 项目地址 进行下载。
- **Java 8, Java 11**

  单击访问 项目地址 进行下载。
- **Jenkins**

Windows：进入 `C:/` 目录，创建 `codingci/tools` 目录。在其中下载 jenkins.war 、jenkins_home.zip 文件，并在 `tools/` 目录解压 jenkins_home.zip 文件。
Linux：进入 `/root/` 目录，创建 `codingci/tools` 目录。在其中下载 jenkins.war 、jenkins_home.zip 文件，并在 `tools/` 目录解压 jenkins_home.zip 文件。
macOS：进入 `~/` 目录，创建 `codingci/tools` 目录。在其中下载 jenkins.war 、jenkins_home.zip 文件，并在 `tools/` 目录解压 jenkins_home.zip 文件。

> **说明：**

> 预装以上环境是接入自定义节点时的先决条件。接入后若要确保构建计划正常运行，仍需视任务或插件的实际需求预装不同的环境，您可以参见 默认节点环境 安装常见的 SDK 与命令行工具。

macOS

命令接入

1. 进入构建节点，选择**接入新节点 > macOS**，接入方式选择 Bash，在接入配置中选择对应节点池，单击**生成接入配置并复制**。

2. 在终端中输入命令后，等待服务下载完成。安装完成后可以使用以下命令进行验证：

```
qci_worker version
```

命令接入的默认安装目录为 `/root/codingci` 。

手动接入

使用手动接入方式前请确保节点 已满足上文中的 环境依赖 要求。

1. 接入方式选择手动接入，按照提示在终端中输入命令安装客户端，即安装 `Worker` 服务。

2. 选择拟接入的节点池，单击**一键生成并复制**，生成初始化命令。
3. 在终端中执行已自动生成的客户端启动命令，让构建节点保持在线状态。

Windows

命令接入

1. 选择**持续集成 > 构建节点**中，单击右上角的**接入新节点**，选中 Windows，选择 Powershell 接入方式。

2. 安装完成后可以使用以下命令进行验证：

```
qci_worker version
```

命令接入的默认安装目录为 `/root/codingci` 。

手动接入

使用手动接入方式前请确保节点 已满足上文中的 环境依赖 要求。

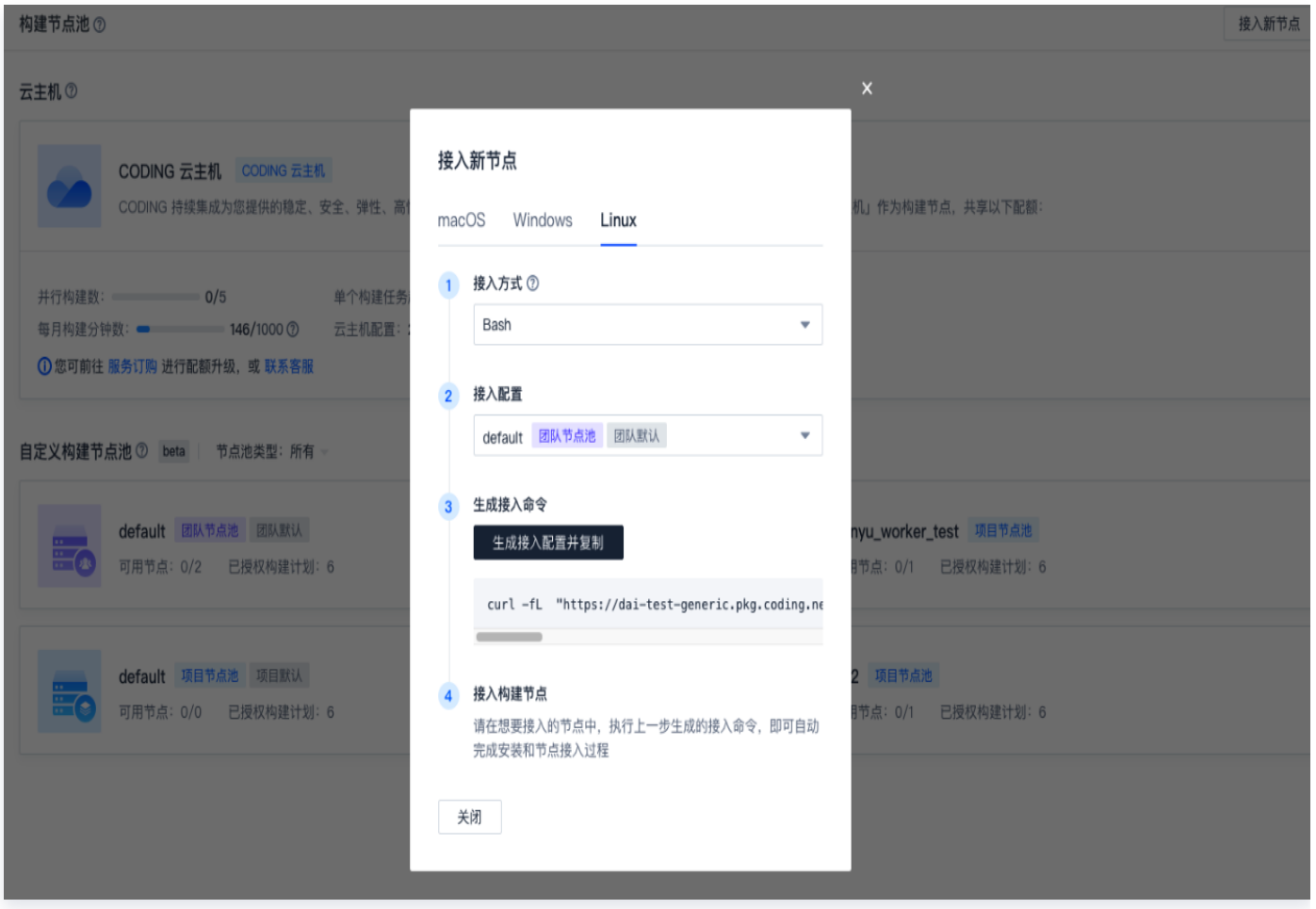
1. 接入方式选择手动接入，按照提示在终端中输入命令安装客户端，即安装 `Worker` 服务。

2. 选择拟接入的节点池，单击**一键生成并复制**，生成初始化命令。
3. 在终端中执行已自动生成的客户端启动命令，让构建节点保持在线状态。

Linux

命令接入

1. 选择**持续集成 > 构建节点**中，单击右上角的**接入新节点**，选中 Linux，选择 Bash 接入方式。设置完成后在 Linux 环境中运行已生成的接入配置命令。



2. 安装完成后可以使用以下命令进行验证：

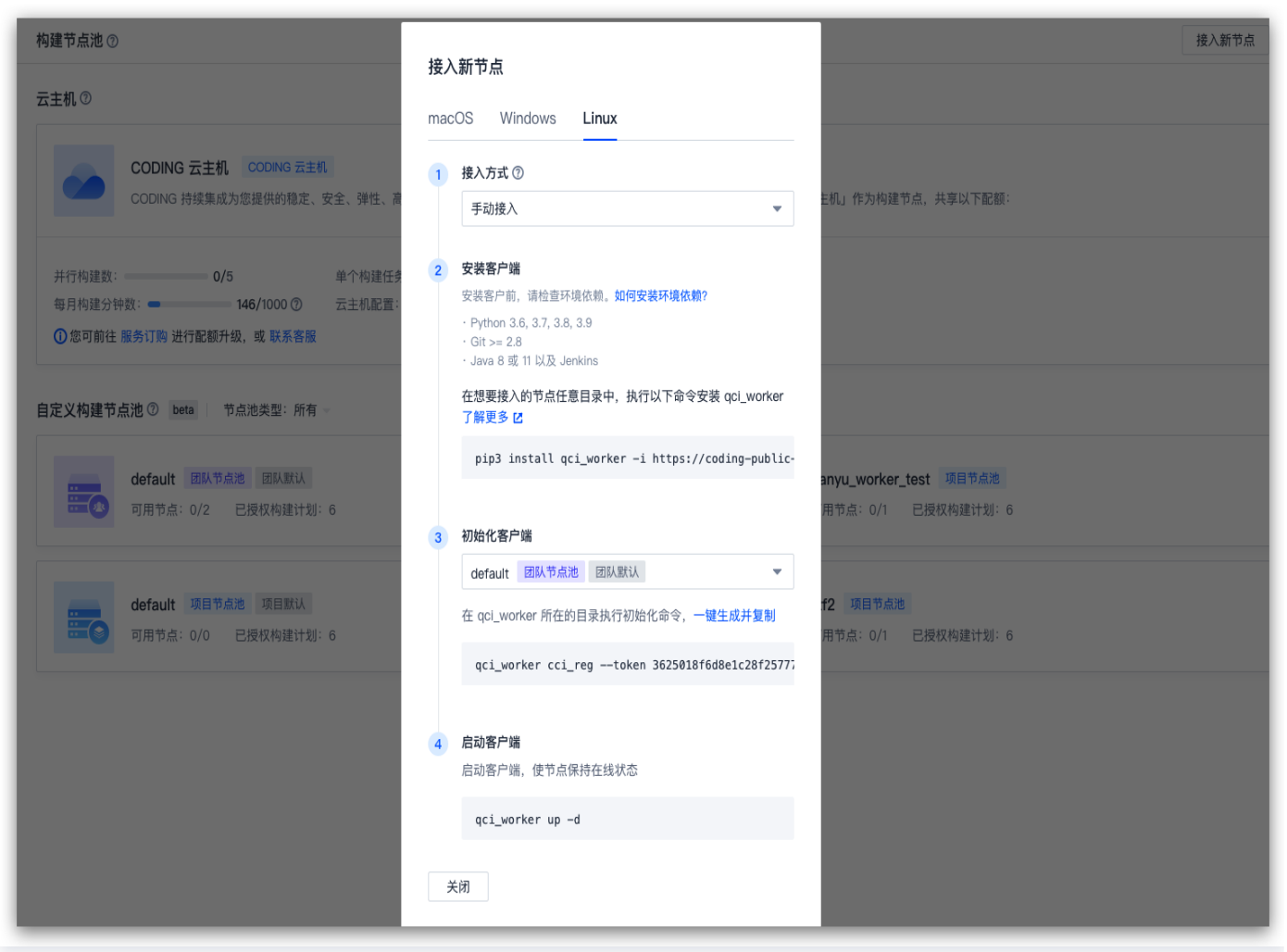
```
qci_worker version
```

命令接入的默认安装目录为 `/root/codingci` 。

手动接入

使用手动接入方式前请确保节点 已满足上文中的 环境依赖 要求。

1. 接入方式选择手动接入，按照提示在终端中输入命令安装客户端，即安装 `Worker` 服务。



2. 选择拟接入的节点池，单击**一键生成并复制**，生成初始化命令。
3. 在终端中执行已自动生成的客户端启动命令，让构建节点保持在线状态。

启动守护进程

安装完成后，在构建节点上需要运行守护进程，用以监听并获取由 CODING 后台下发的 CI 任务。以下为运行 / 删除命令行：

```
# 在后台运行
qci_worker up -d

# 在前台运行
qci_worker up

# 暂时停止运行
qci_worker stop
```

Worker 常用命令

最近更新时间：2023-09-20 11:11:01

本文为您介绍 qci-worker 服务的常用命令。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

接入自定义构建节点时将在环境中安装 `Worker` 服务，并由此服务调度 CI 构建任务的下发与计算资源的分配。因此熟悉 `Worker` 服务的常用命令能够更好地配合 CI 构建任务，不同操作系统下的安装方法请参见 自定义节点 。

常用配置项命令

注册

```
qci_worker reg_cci  --token  token   --server server  --home home
--token 项目令牌，必填
--server  指定的接入服务，非必填
--home    指定工作目录
```

示例：

```
qci_worker cci_reg  --token db6fd4d6a2fc7d753a2985d55c44a2262f3e543f  --
server ws://codingcorp.nh113vufq.dev.coding.io --home ~/.codingqci
```

启动服务

```
qci_worker up -d
```

重启服务

```
qci_worker stop
```

```
qci_worker up -d
```

手动删除节点

```
qci_worker stop    #停止 qci_worker
qci_worker remove  # 后台删除节点
```

修改配置

若需要让指定的 Jenkins 配置项生效，需要先停止 Jenkins 服务进程，然后重启 `qci_worker` 服务。

```
qci_worker config  JENKINS_HOST=127.0.0.1    # 指定 Jenkins 启动 host
qci_worker config  JENKINS_PORT=15740        # 指定 Jenkins 启动 port

qci_worker config JENKINS_ENCODING='UTF-8'
qci_worker config JENKINS_OPT="-Xms4096m -Xmx4096m" # 指定 Jenkins 使用的
JVM 内存申请大小

qci_worker config PACKAGE_AUTO_UPDATE=0 # 关闭自动更新
```

构建节点池

最近更新时间：2024-08-02 15:23:01

本文为您介绍构建节点池。

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. 进入目标项目后，选择左侧导航栏的**持续集成**。

功能介绍

构建节点池是构建节点的集合，在使用自定义的构建节点时，需要将构建节点接入构建节点池，并通过将构建计划节点池配置来指定构建节点池。

权限控制

用户组需具备**团队构建节点**权限才能进行创建/删除构建节点池等操作。选择左侧导航栏**团队设置中心 > 全局设置 > 团队权限方案**为用户组勾选开启相应的权限。

单个项目内支持设置多个构建节点池，每个构建节点池支持接入多个构建节点。在构建节点池详情中的节点列表可以查看节点状态并对其进行管理。

节点状态

- **闲置**：构建节点此时空闲。
- **占用**：构建节点已被分配到构建任务中使用。
- **准备中**：构建节点正在准备构建环境。
- **开启**：只有处于开启状态的节点才能被分配使用，如果关闭节点不会影响正在运行的构建任务。
- **删除**：节点将会脱离 CODING 持续集成服务，但只会删除工作空间和相关的配置信息，之前产生的全局缓存文件仍会保留。

构建节点池详情内可以查看节点的构建记录。

1. 构建节点池默认授权给所有构建计划，您也可以选择只授权给指定的构建计划（支持多选）。

2. 在构建计划的基础信息设置中可以修改相应的节点池配置。构建计划默认使用 CODING 提供的云主机，您也可以选择其它项目内配置的节点池进行构建。


项目概览
项目协同
代码仓库
代码分析 beta
持续集成
构建计划
构建节点 beta
持续部署
制品库
测试管理
文档管理
示例任务
基础信息
流程配置
触发规则
变量与缓存
通知提醒
代码源
CODING
GitHub.com
GitLab.com
私有 GitLab
码云
不使用
代码仓库
coding-demo
配置来源
使用代码库中的
Jenkinsfile
使用静态配置的 Jenkinsfile
节点池配置
使用 CODING 提供的云主机进行构建
团队 CI 构建配额信息
使用自定义的构建节点进行构建
default
项目节点池
项目默认
可用节点：0/1
default
团队节点池
团队默认
可用节点：0/0
保存修改
取消