持续集成

操作指南

产品文档

腾讯云

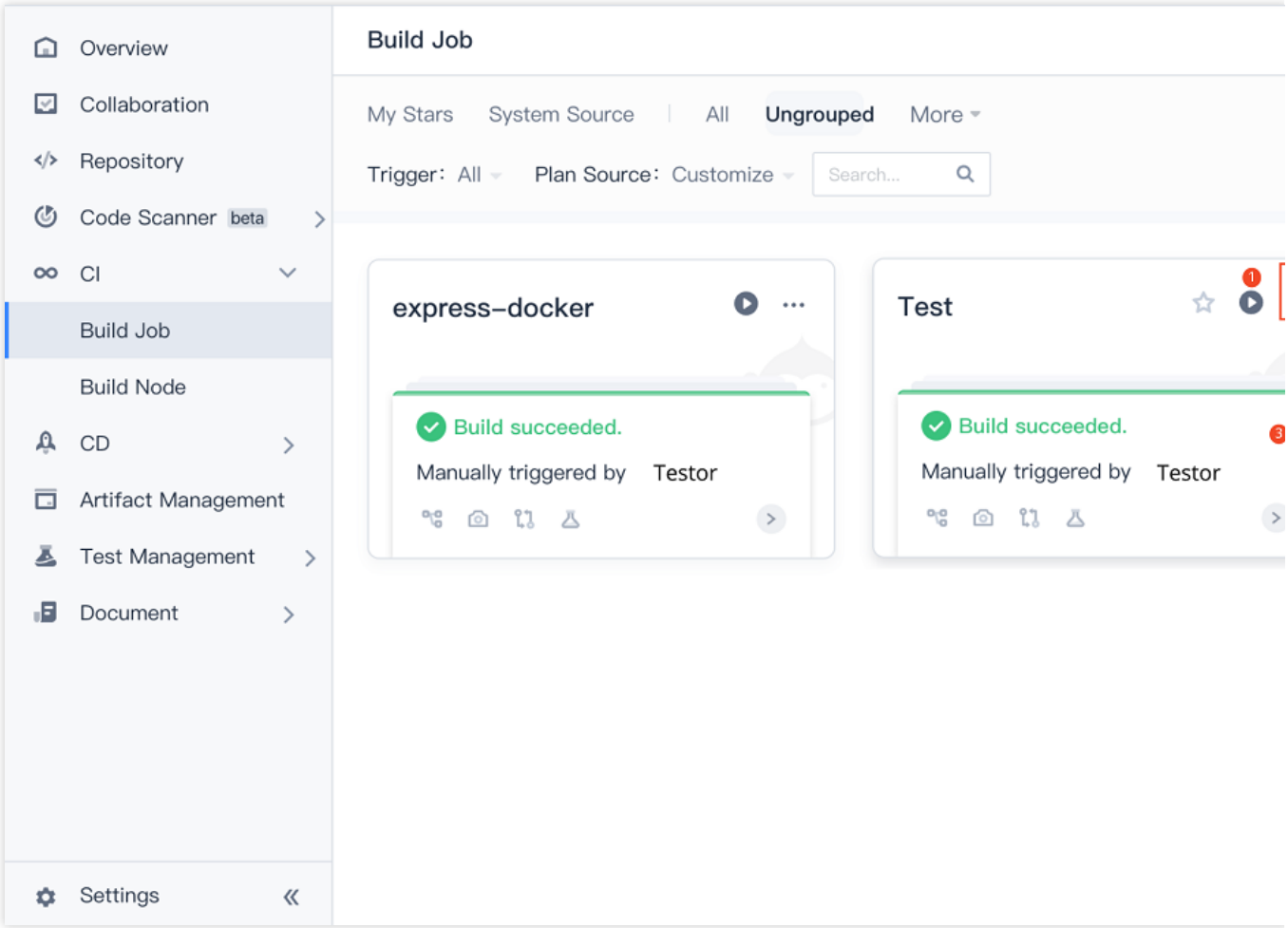
文档目录

操作指南
编写构建流程
文本编辑器

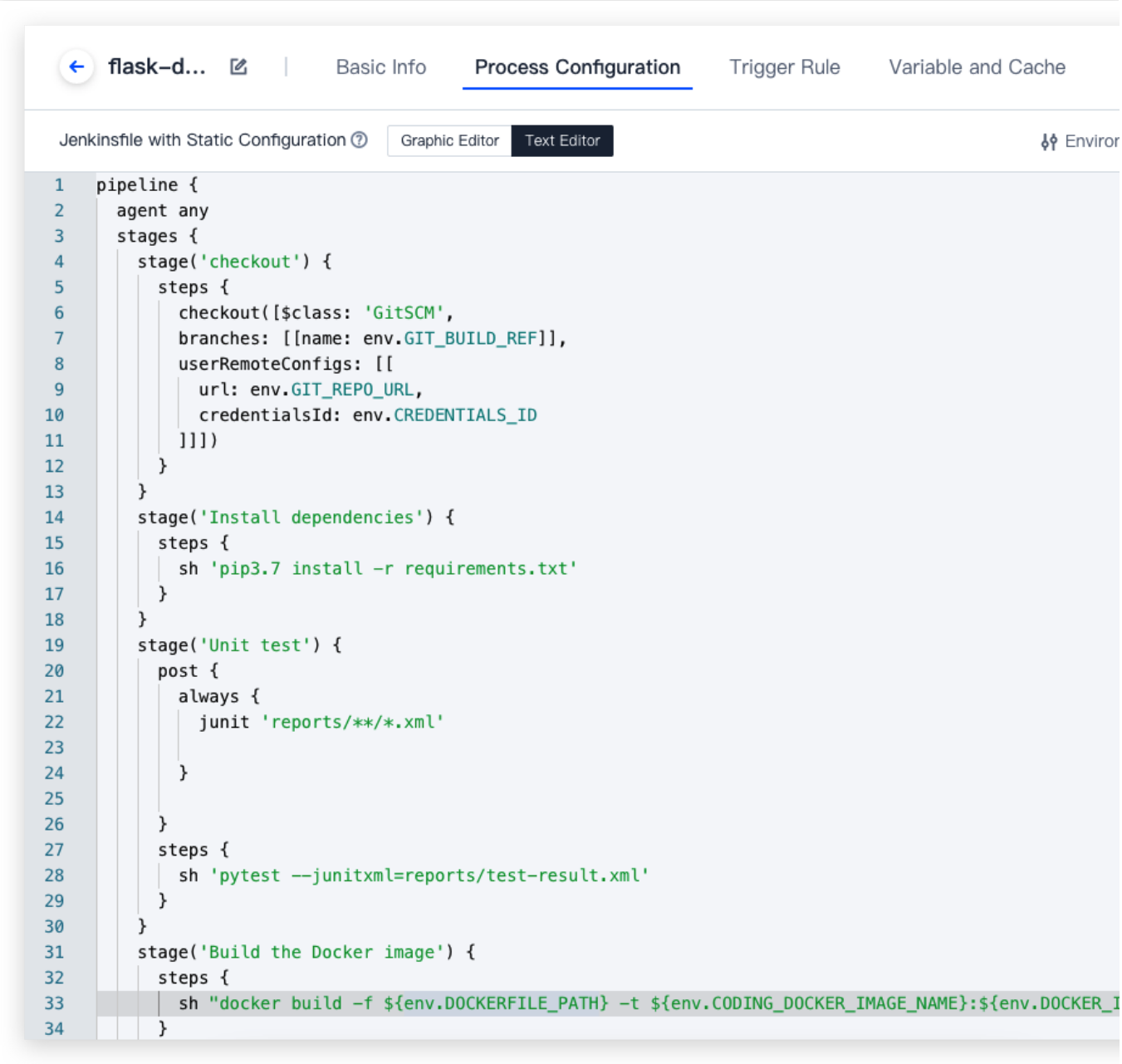
最近更新时间：2023-12-29 11:44:51

title: 文本编辑器 - CODING 帮助中心
pageTitle: 文本编辑器
pagePrevTitle: 快速开始
pagePrev: ci/start.html
pageNextTitle: 配置详情
pageNext: ci/process/detail.html

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。
进入任意项目后，点击左侧的「持续集成」按钮，点击构建计划的设置按钮。



点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。

参考阅读

Jenkinsfile 官方文档

配置详情

==== 2021/08/14 ====

流程配置详情

最近更新时间：2023-12-29 11:44:51

title: 流程配置详情 - CODING 帮助中心
pageTitle: 流程配置详情
pagePrevTitle: 文本编辑器
pagePrev: ci/process/text.html
pageNextTitle: 图形化编辑器
pageNext: ci/process/visual-editor.html
alias: process/detail.html

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。

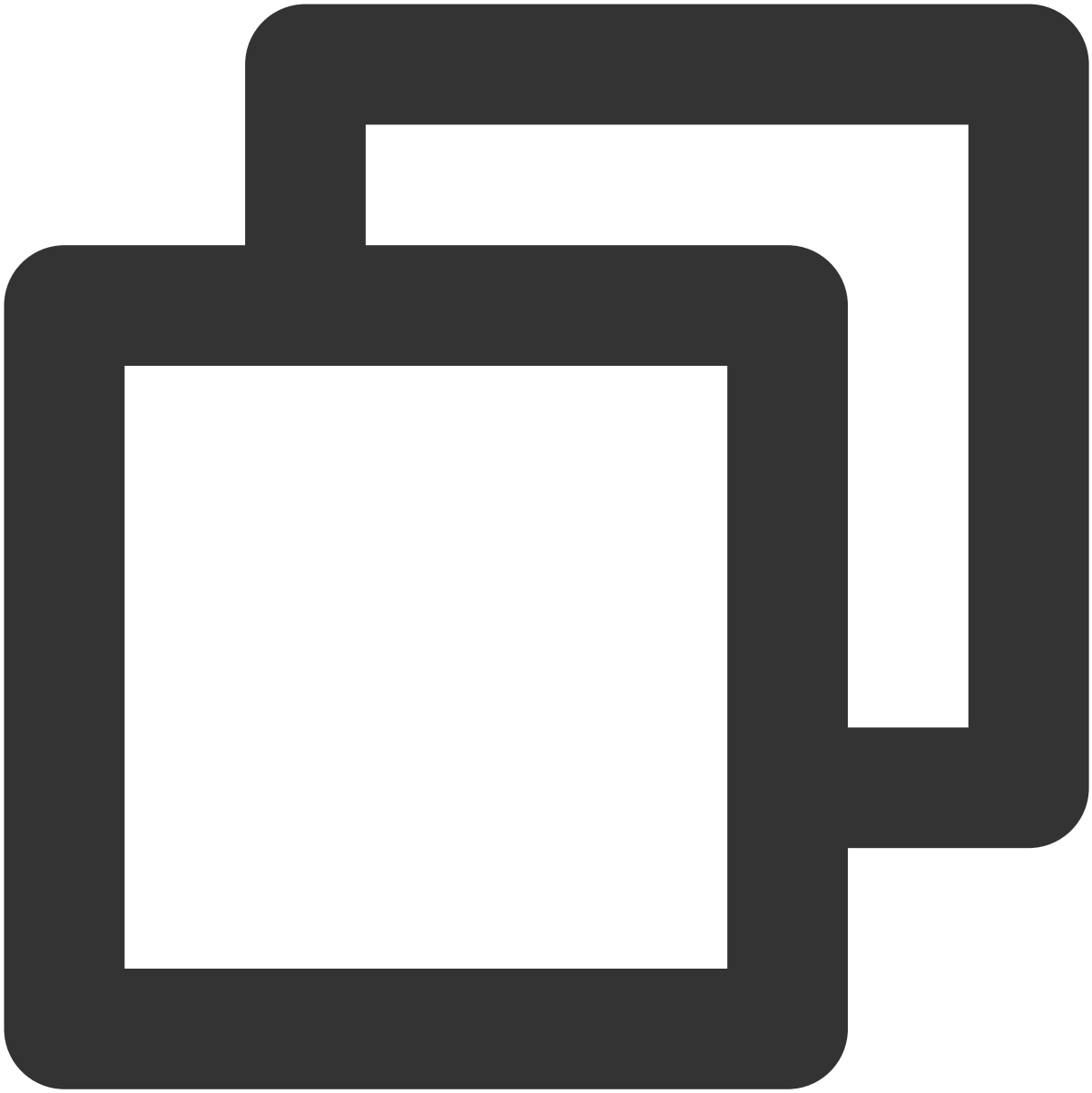
代码仓库

Git

用以检出当前项目的 Git 仓库源代码。本指令是 checkouk 指令的一个简易写法。
参数列表：
Git 地址 `url` : 类型 string
分支 `branch` : 类型 string
变更日志 `changelog` : 类型 string
身份认证 ID `credentialsId` : 类型 string
Poll `poll` : 类型 boolean

从版本控制检出

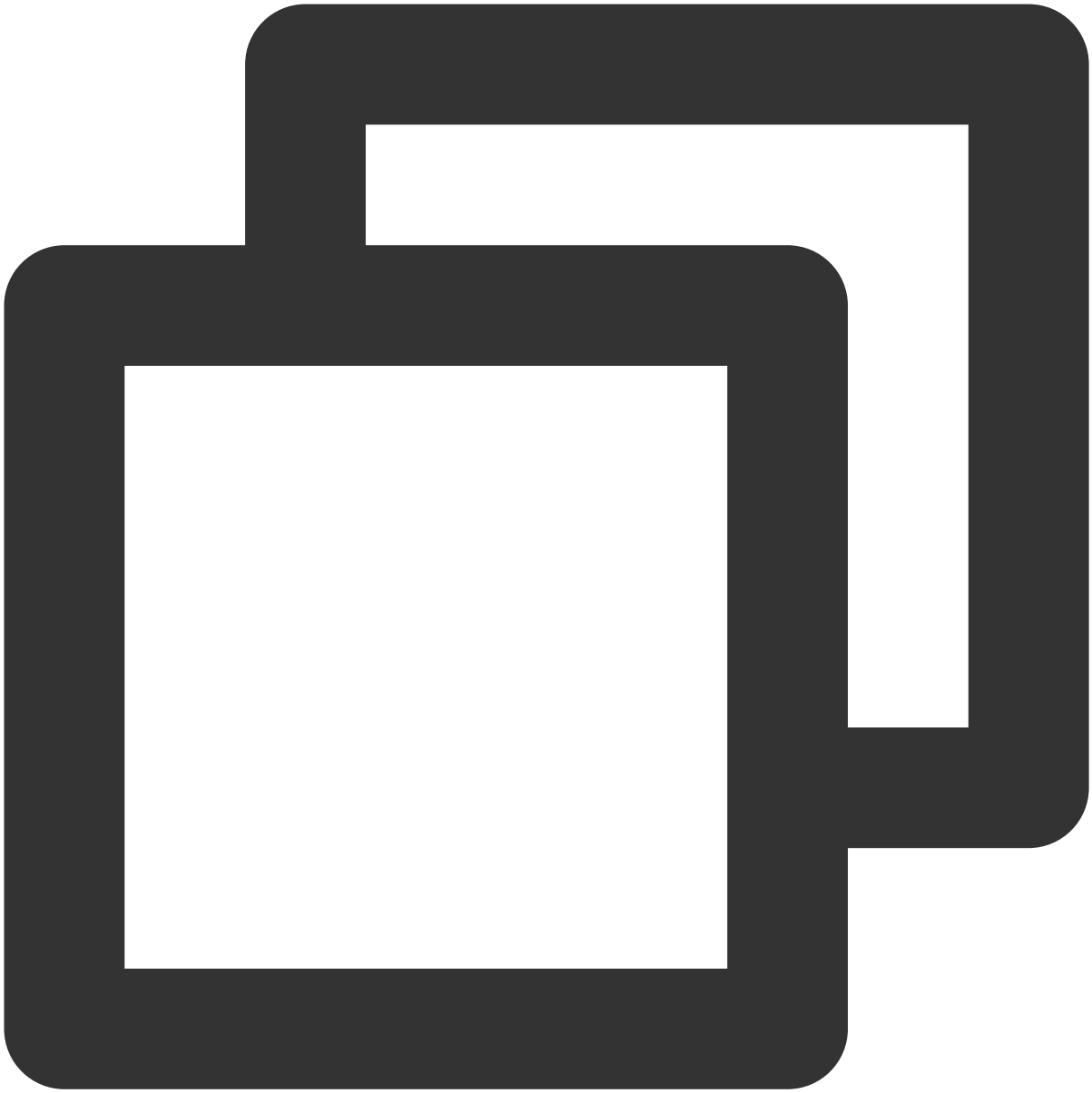
通用的检出 SCM(Git，SVN) 代码。
这个步骤返回一个 Map 格式的内容。例如如果你用 Git，你可以这样：



```
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：

`url` : 类型 string

`name` : 类型 string 远程仓库名称，例如 origin

`refspec` : 类型 string 关于 refspec 的详细说明，请查看这里: Git 内部原理 - 引用规格 。

`branches` : 类型 对象数组, 可选

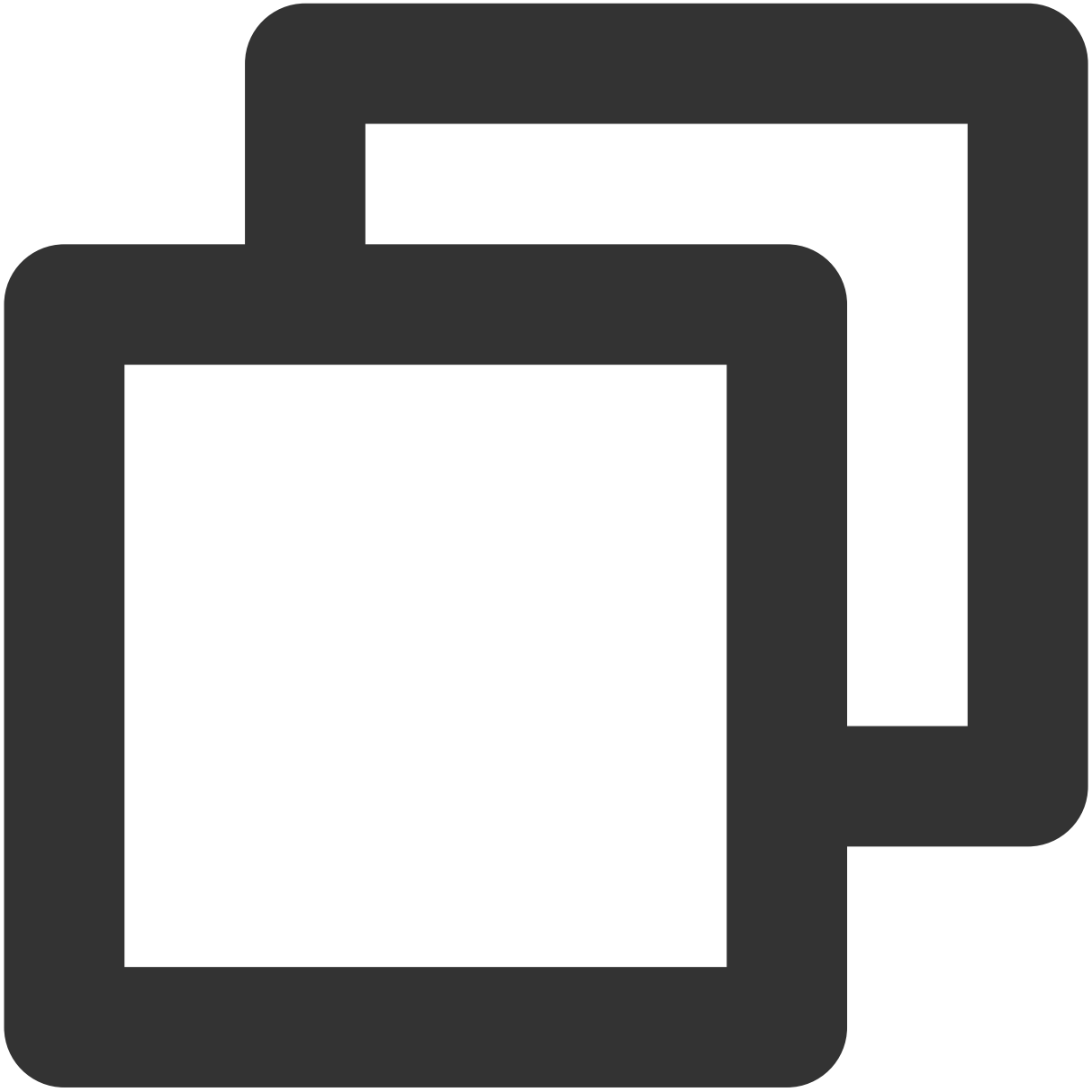
`changelog` : 类型 boolean, 可选

`credentialsId` : 类型 string, 可选

`doGenerateSubmoduleConfigurations` : 类型 boolean, 可选

`submoduleCfg` : 类型数组, 可选

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:

`locations` : 类型对象数组

`remote` : 类型 string

`credentialsId` : 类型 string

`local` : 类型 string 指明一个本地目录（相对于 workspace）作为检出代码的位置

`depthOption` : 类型 string 对应 --depth 的内容。默认值是无穷大, 点击这里了解更多信息 。

`ignoreExternalsOption` : 类型 boolean

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:

`artifacts` : 类型 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` : 类型 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` : 类型 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` : 类型 boolean, 可选

`excludes` : 类型 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 。

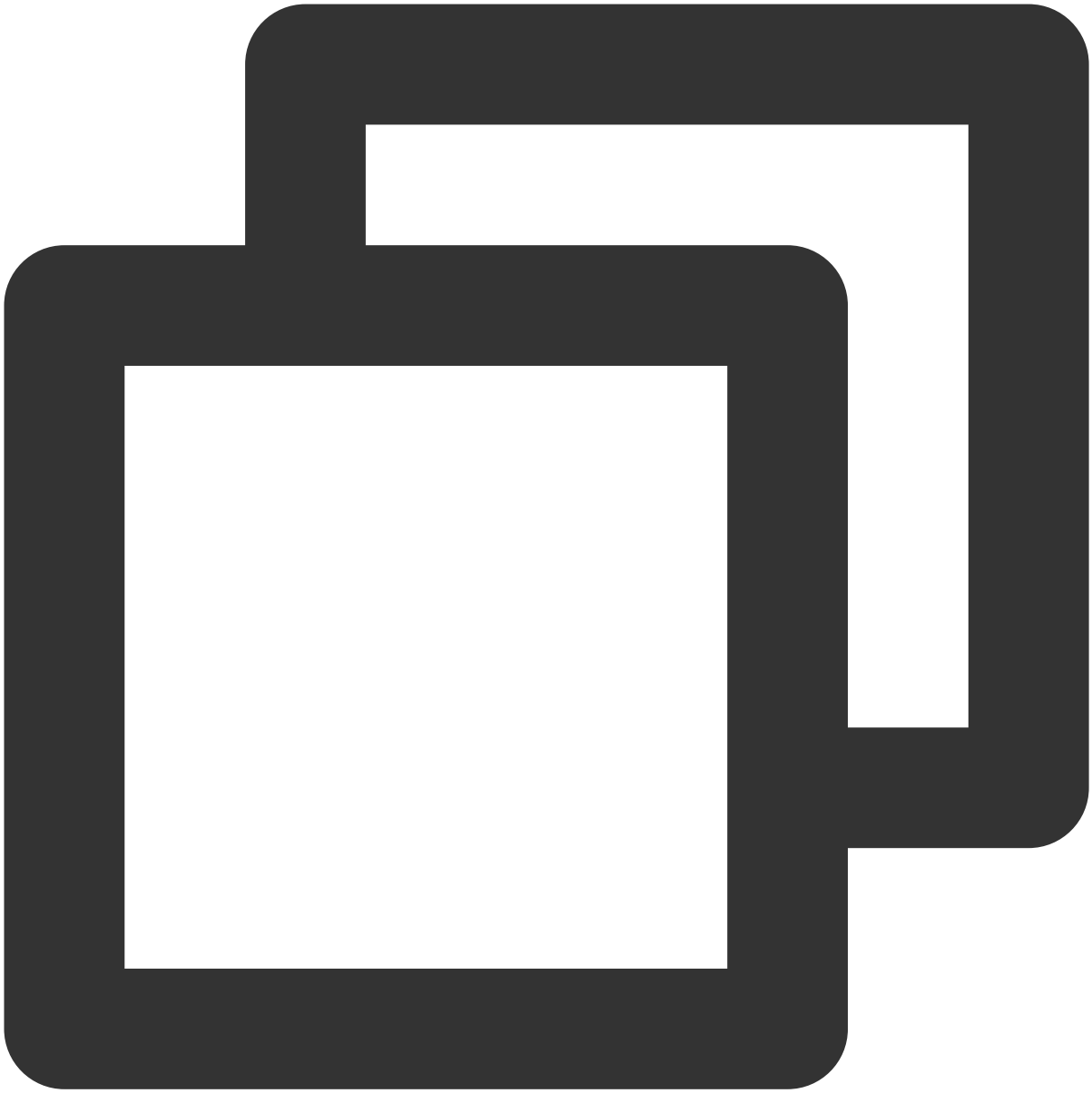
`fingerprint` : 类型 boolean, 可选 收集的时候同时计算文件的 hash 信息

`onlyIfSuccessful` : 类型 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），你可以指定收集哪些 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，你可以用逗号隔开写多个规则。
参数列表：
`testResults` : 类型 string
`allowEmptyResults` : 类型 boolean，可选，允许测试报告文件不存在或为空。
`keepLongStdio` : 类型 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。你也可以使用 throw new Excepiton(),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

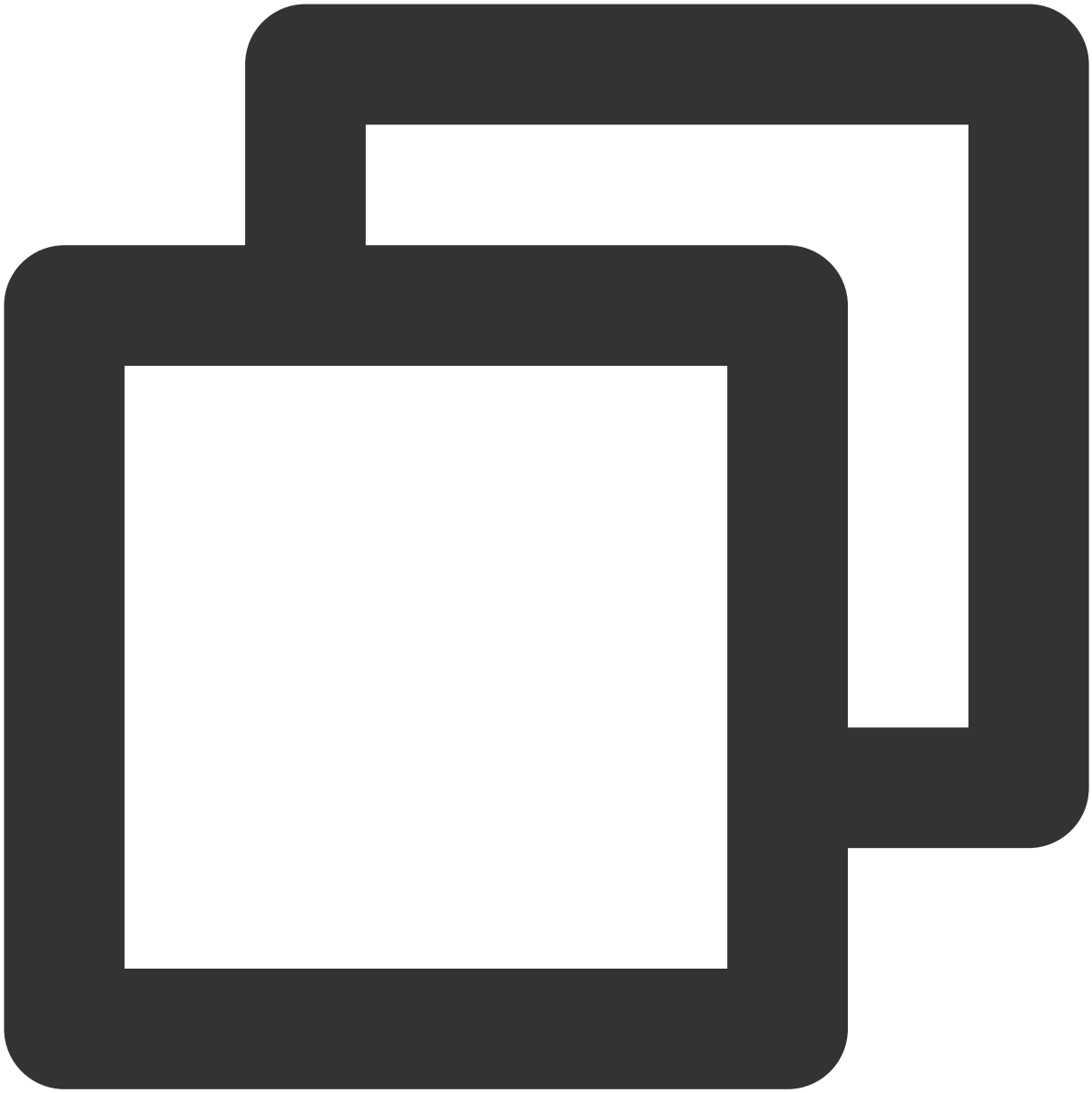
执行任意的 Pipeline 脚本

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。你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。你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。
==== 2020/08/13 ====

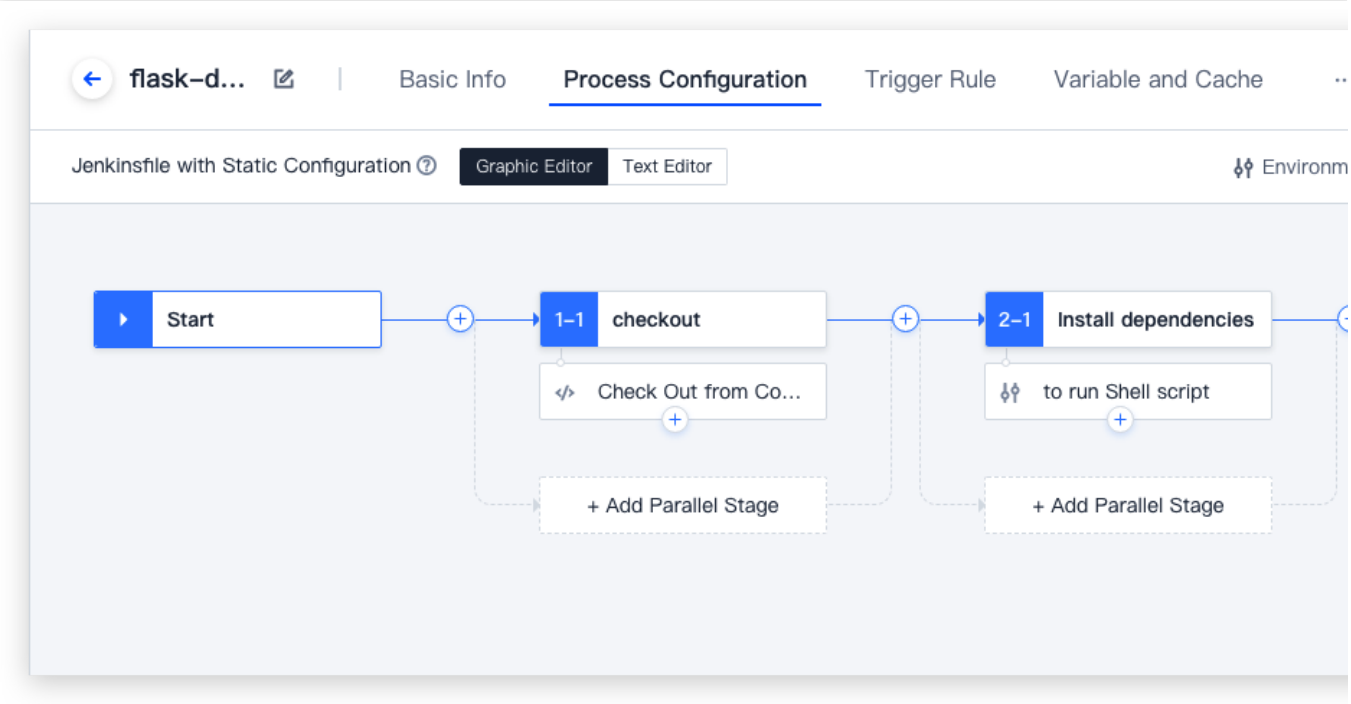
图形化编辑器

最近更新时间：2023-12-29 11:44:50

title: 图形化编辑器 - CODING 帮助中心
pageTitle: 图形化编辑器
pagePrevTitle: 流程配置详情
pagePrev: ci/process/detail.html
pageNextTitle: 触发规则
pageNext: ci/configuration/trigger.html
alias:
devops/ci/visualeditor.html
ci/visualeditor.html
ci/visual-editor.html

功能介绍

通过命令行文编编辑 Jenkinfile（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。

说明：

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。比如“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

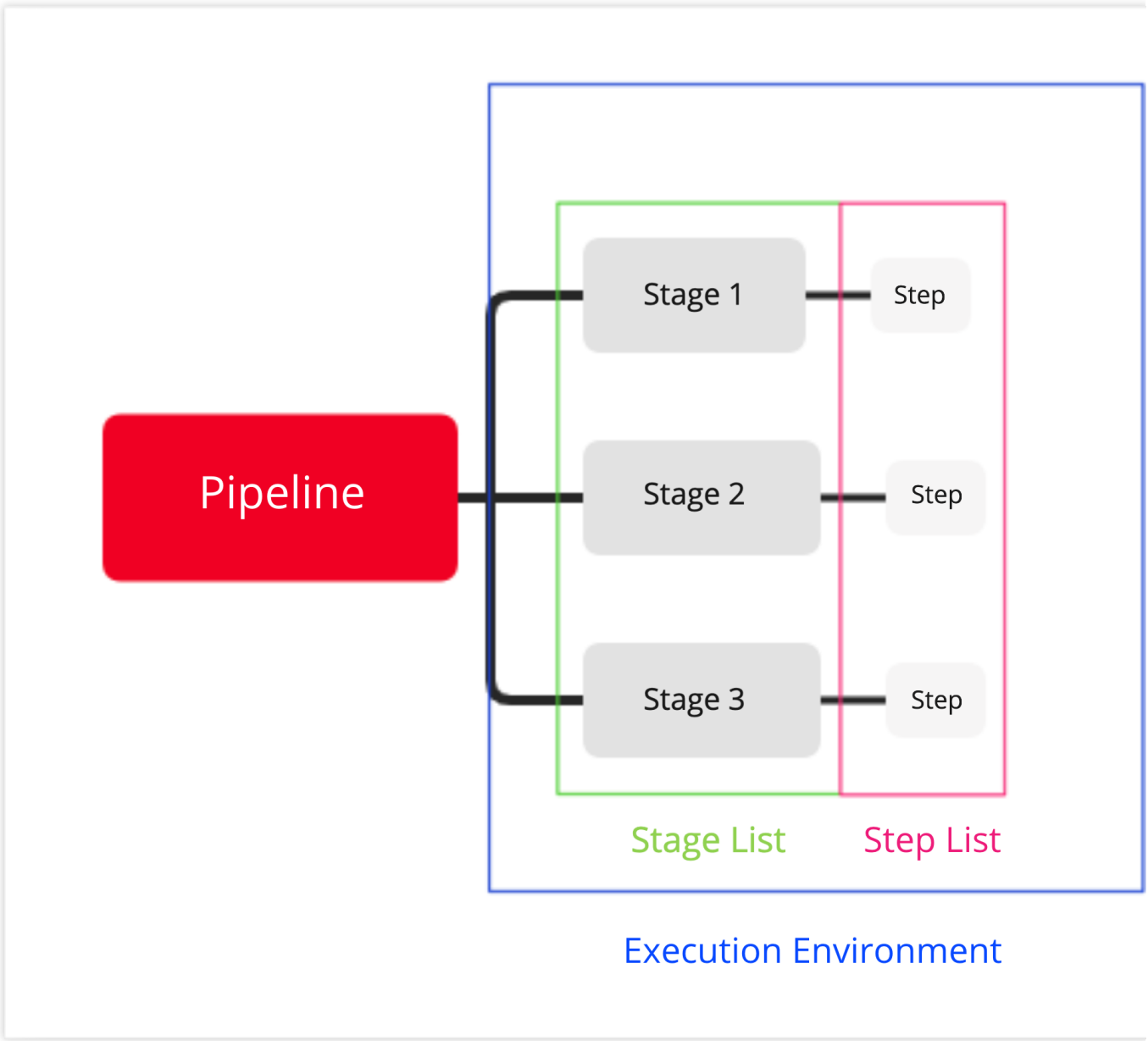
步骤列表

`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。比如有一个 `步骤（step）` 需要系统打印一条“构建中...”的消息，即执行命令 `echo '构建中...'` 。

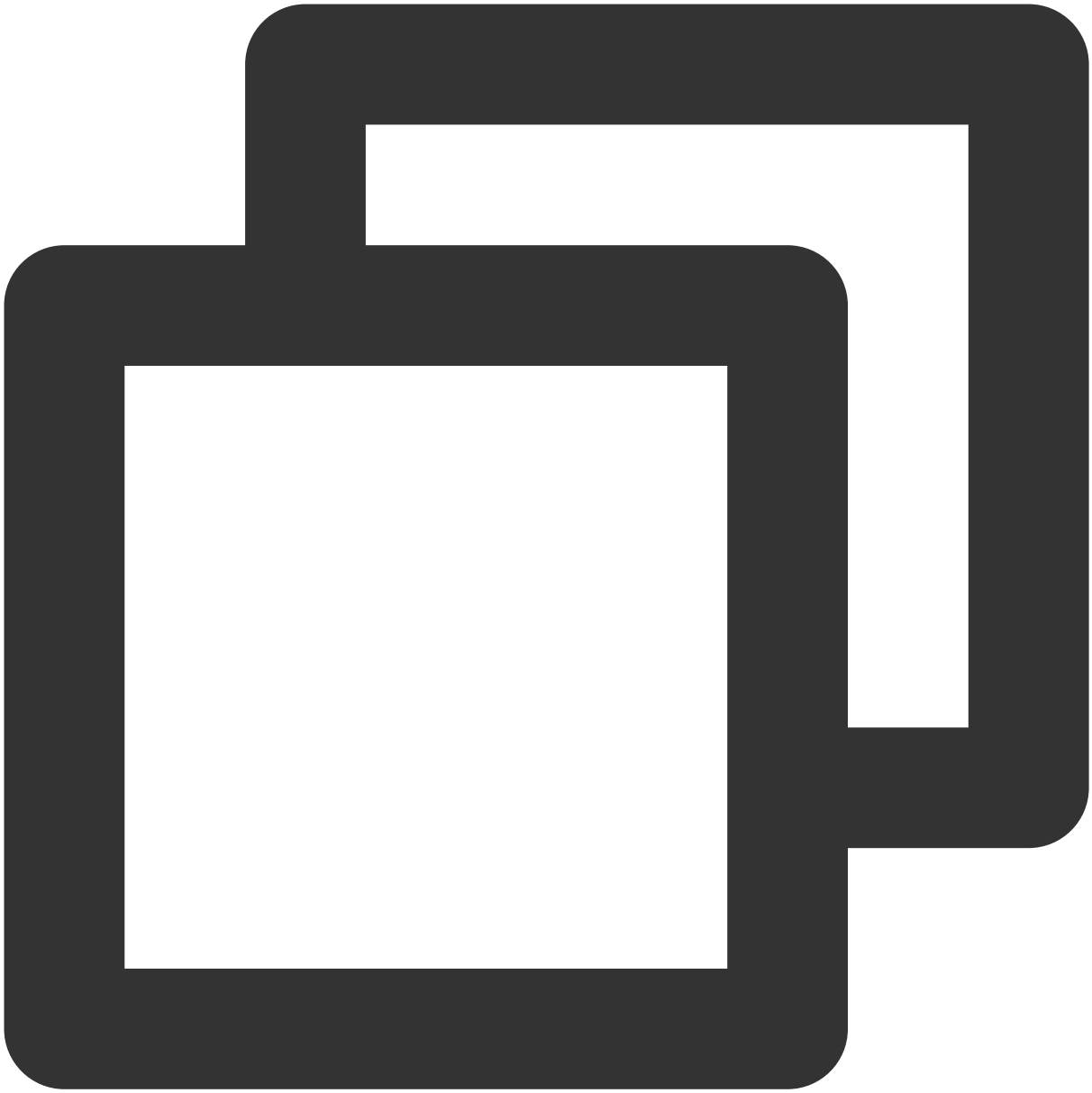
是否必须	是
参数列表	无
允许的位置	在每一个 `阶段（stage）` 块内

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` 。


Pipeline
Stage 1
Stage 2
Stage 3
Step
Step
Step
Stage List
Step List
Execution Environment

示例文件

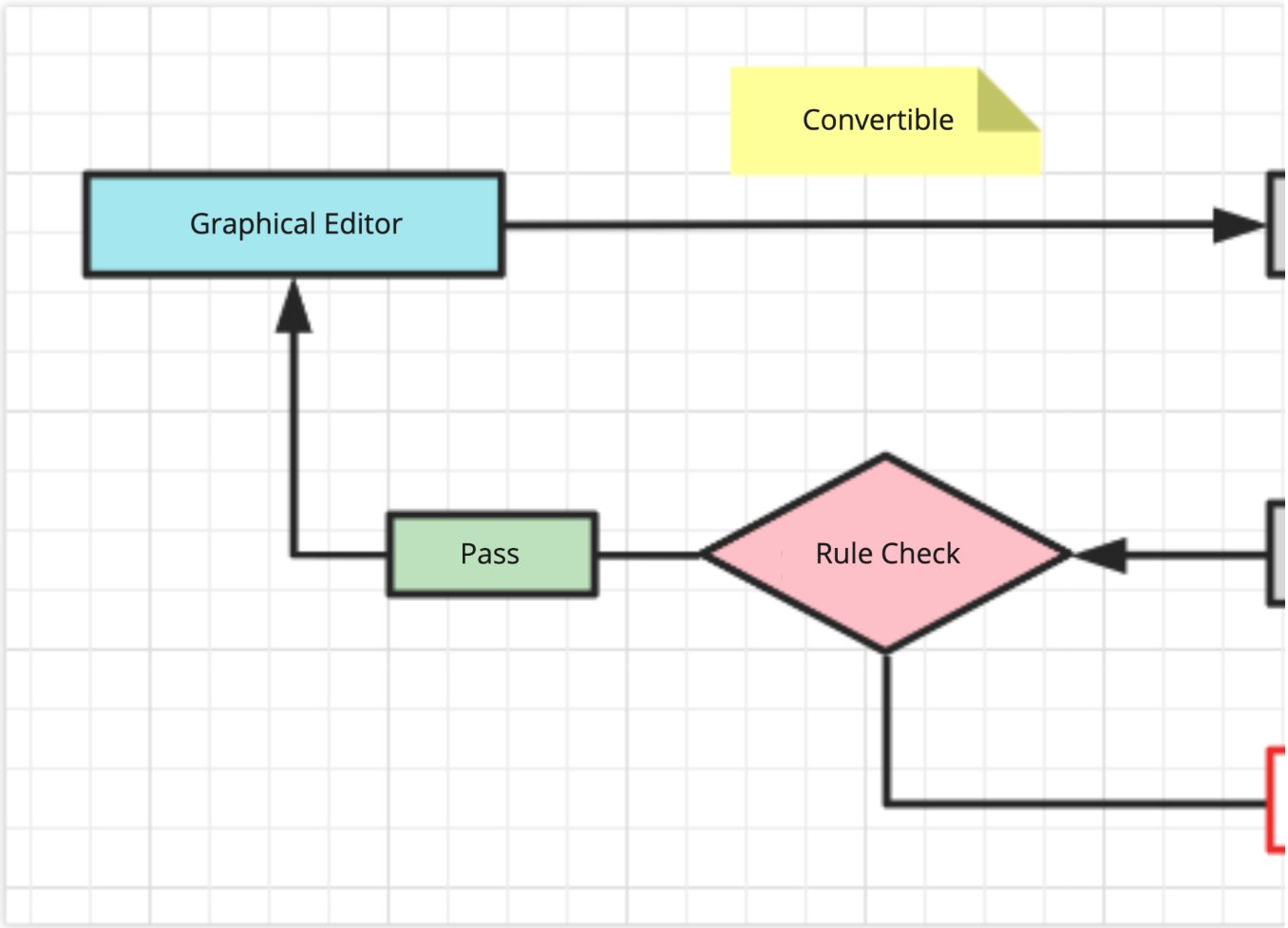


```
pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
        sh 'ci-init'
        checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
                                    userRemoteConfigs: [[url: env.GIT_REPO_URL]]])
      }
    }
    stage('构建') {
```

```
        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。

==== 2020/09/07 ====

配置构建计划
触发规则

最近更新时间：2023-12-29 11:44:51

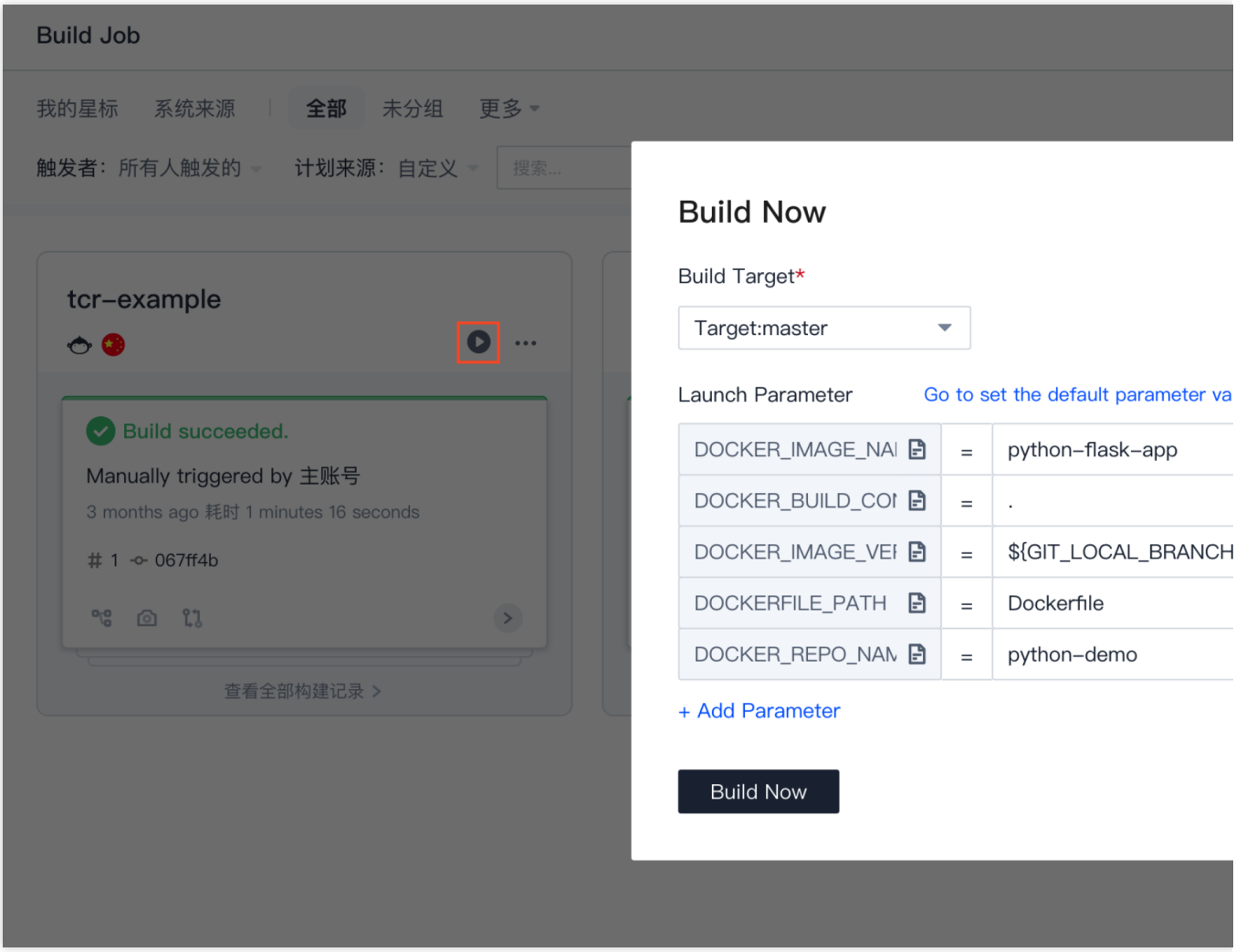
title: 触发规则 - CODING 帮助中心
pageTitle: 触发规则
pagePrevTitle: 图形化编辑器
pagePrev: ci/process/visual-editor.html
pageNextTitle: 环境变量
pageNext: ci/configuration/env.html
alias:
devops/ci/trigger.html
ci/trigger.html

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
代码变更触发
定时触发
API 触发
上述的多种触发方式可以同时使用。

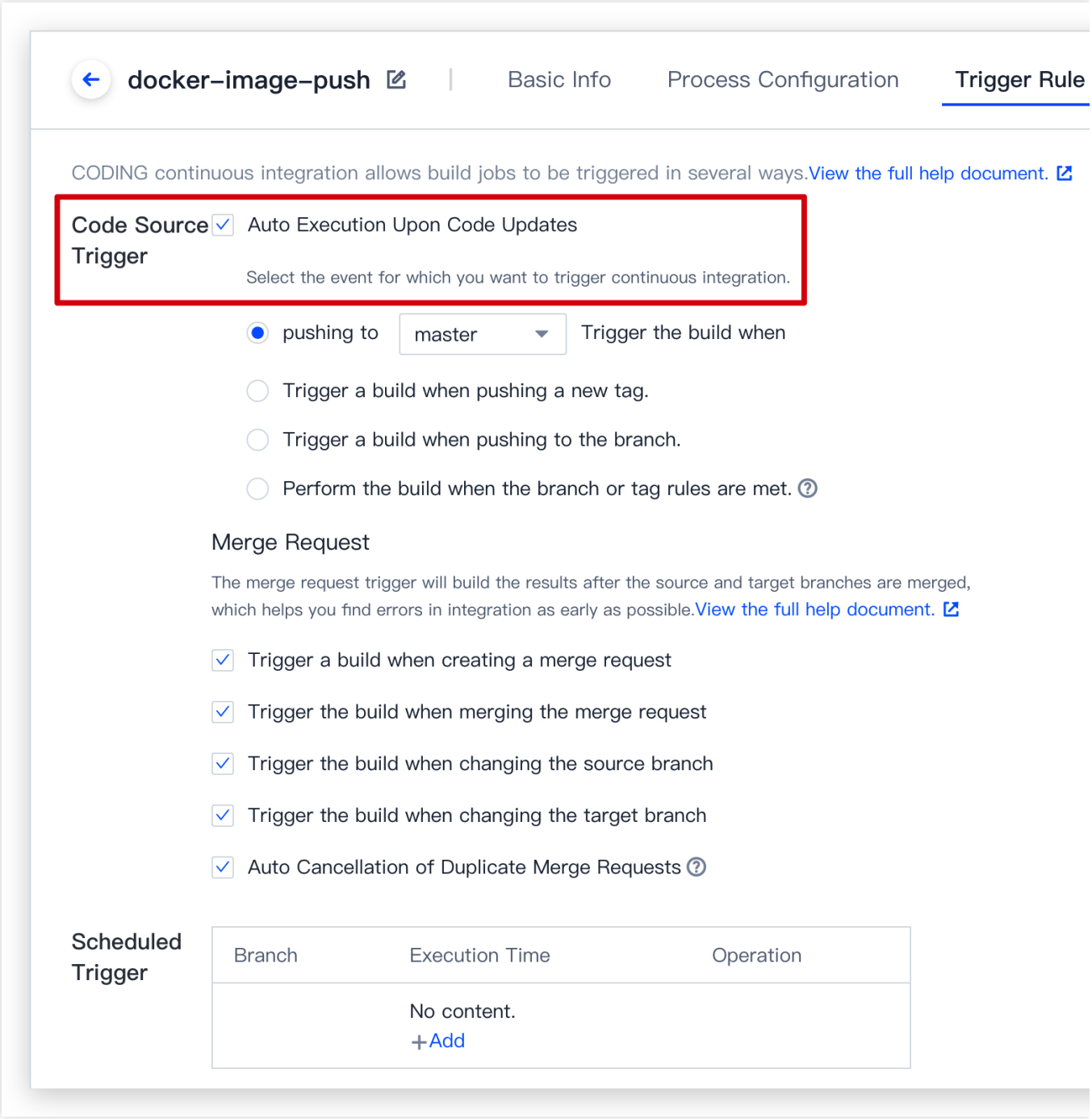
手动触发

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，点击【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。



代码变更触发

配置了代码变更触发的构建计划，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

推送到 <分支> 时触发构建

只有指定的分支更新代码时，才触发构建

推送新标签时触发构建

只有创建了新的 git tag，才会触发构建

推送到分支时触发构建

任意分支更新都会触发构建

符合分支或标签规则时构建

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

目前合并请求会在以下几种情况下执行构建：

发起合并请求时触发

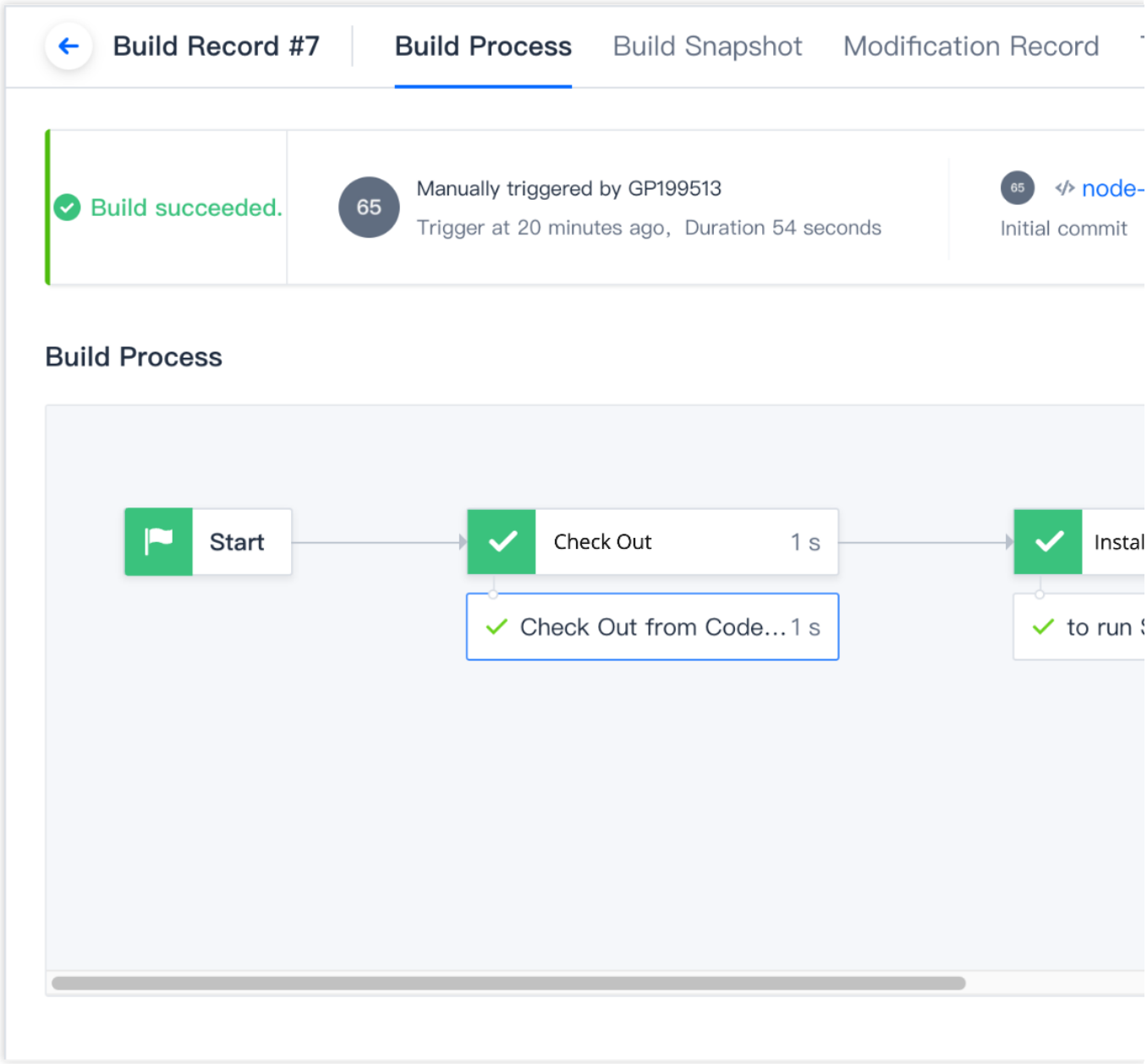
合并请求的源分支发生变更

合并请求的目标分支发生变更

归并合并请求时触发

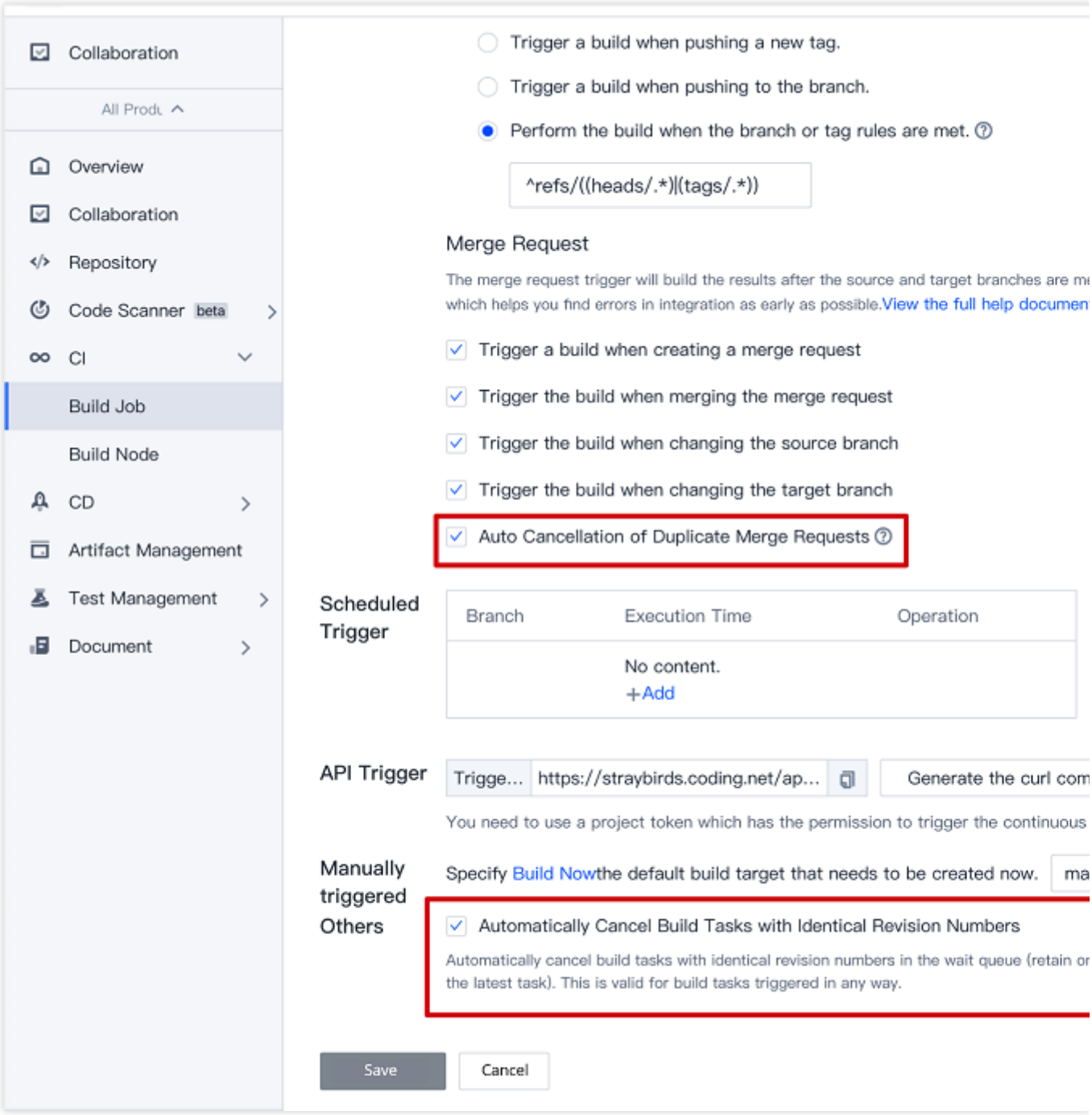
说明：

合并请求执行与**代码更新时执行**的不同之处在于，合并请求构建会构建**源分支与目标分支合并后的结果**，可以尽早发现集成中的错误。



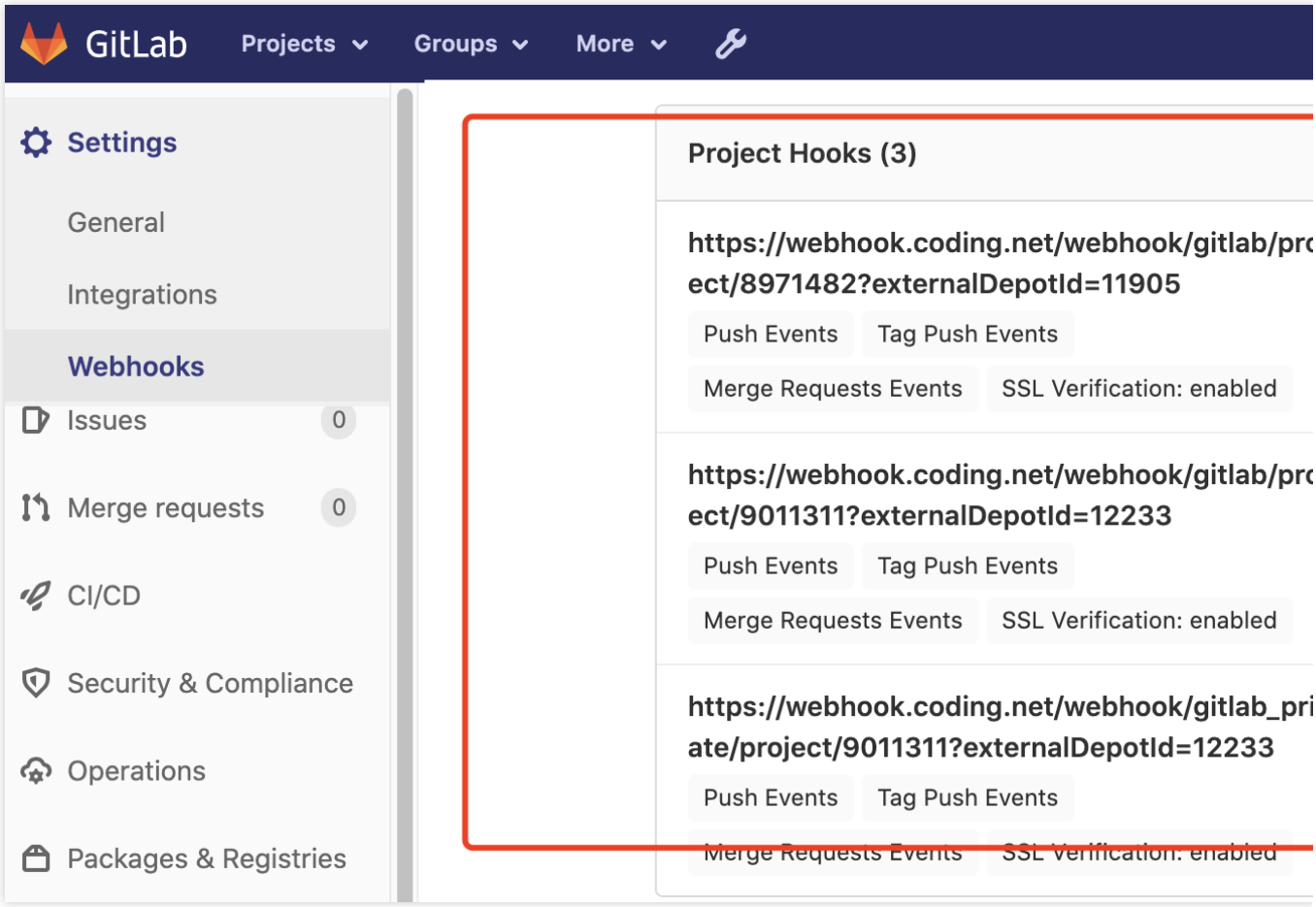
自动取消相同构建

您可以在设置中勾选是否取消「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

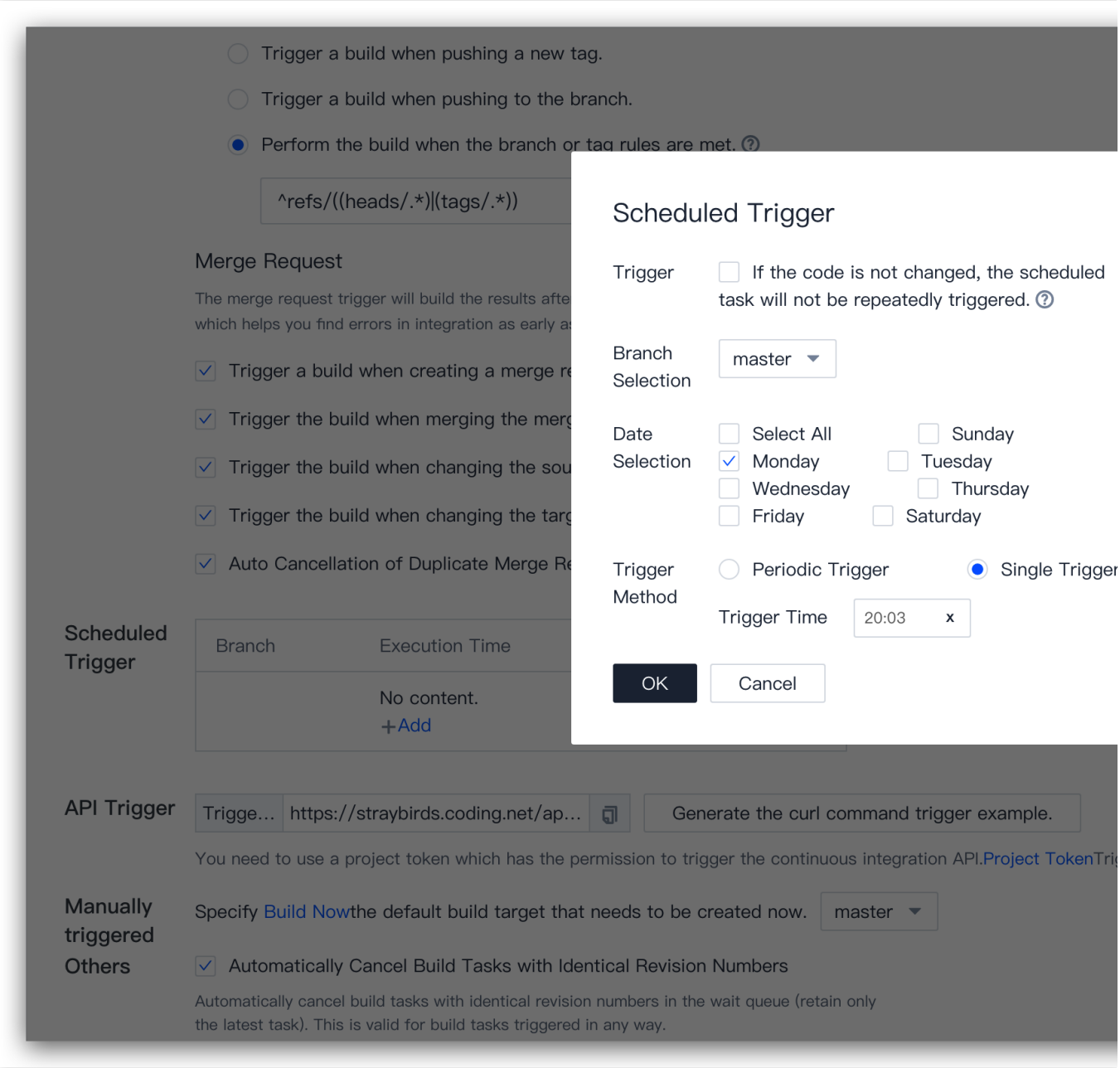
绑定 GitLab 私有云时，会自动创建 GitLab Webhook，后续事件自动通知 CODING，然后匹配上述触发规则设置。



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。



触发条件：代码无变化时不重复触发定时任务

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，也不会触发构建。

日期选择

您可以选择选择一周内的多个日期。

周期触发

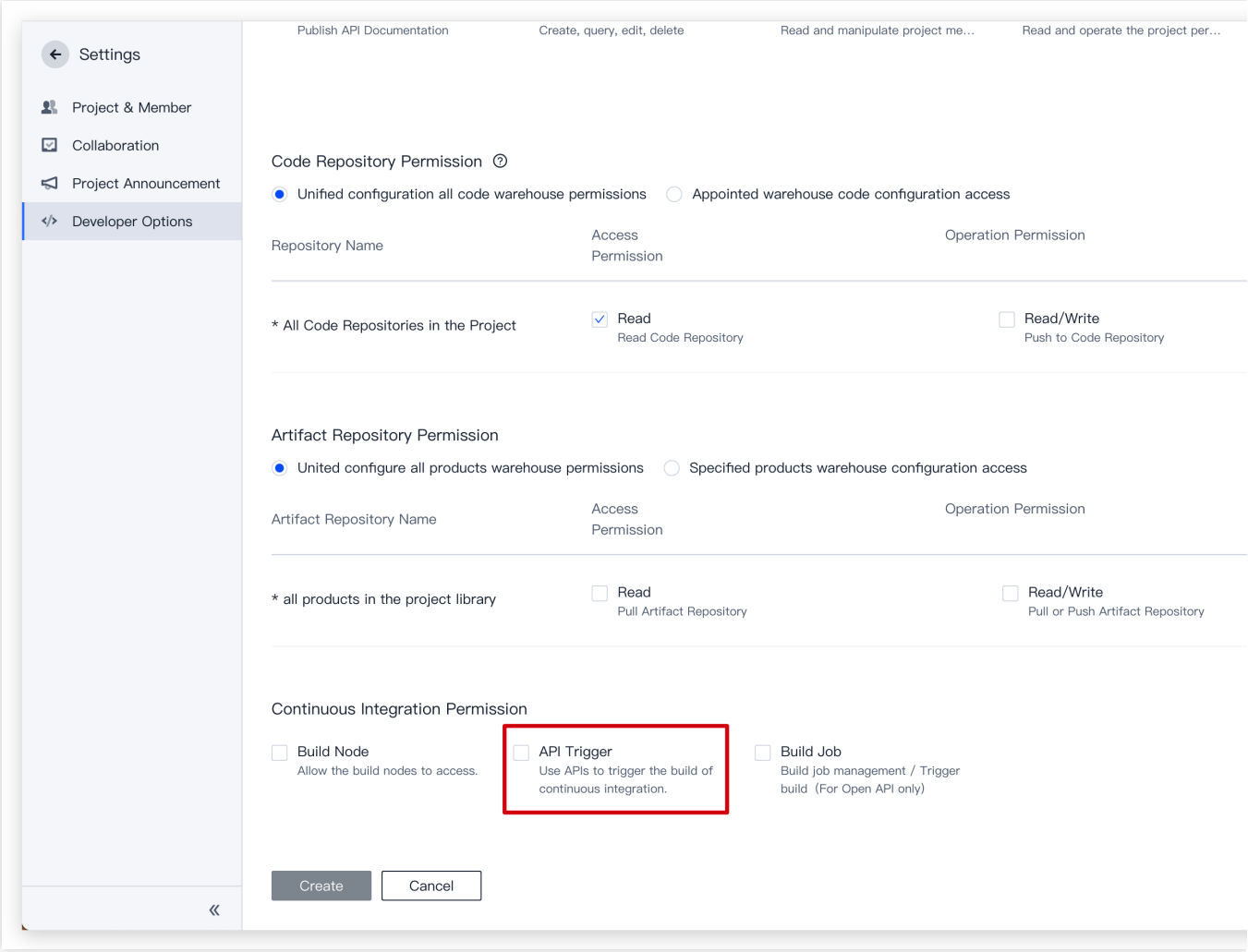
您可以选定 00:00 - 24:00 之间的任意时间为周期（精确到小时），按照选中的间隔触发任务。

单次触发

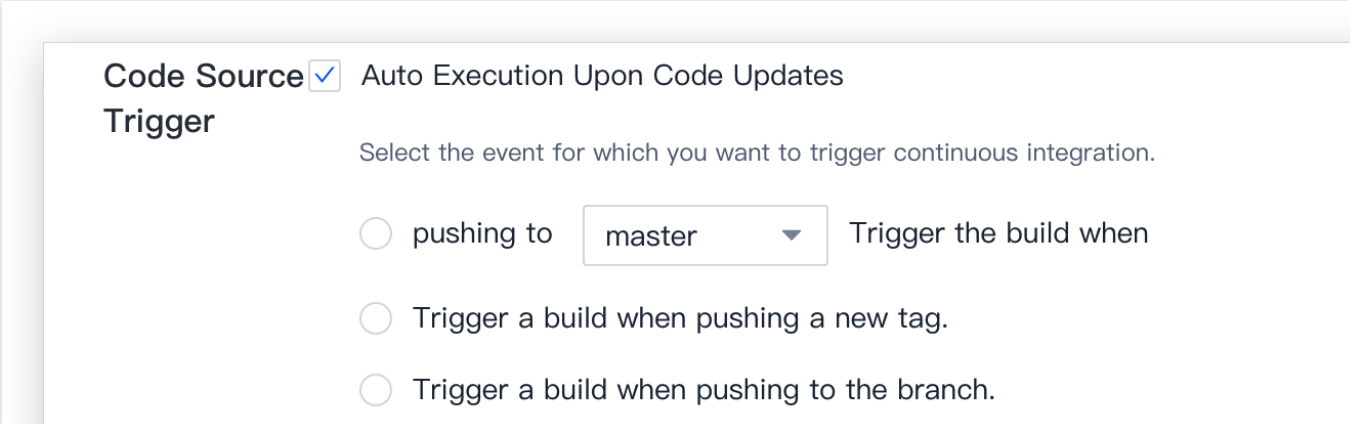
您可以在 00:00 - 24:00 之间选择任意时间为触发时间点（精确到分钟）。

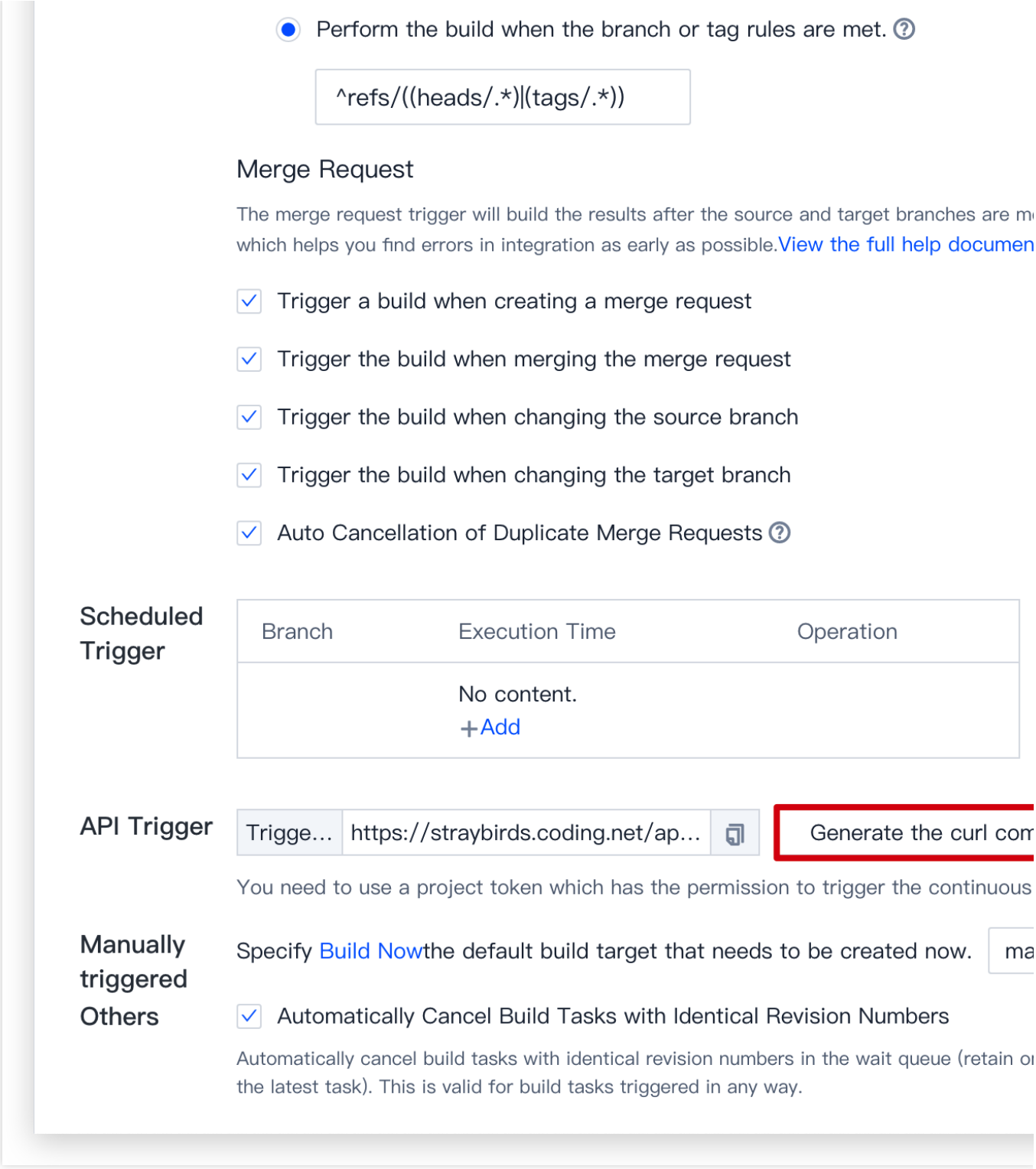
API 触发

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。点击生成 curl 命令触发示例后即可生成相应的调用命令。

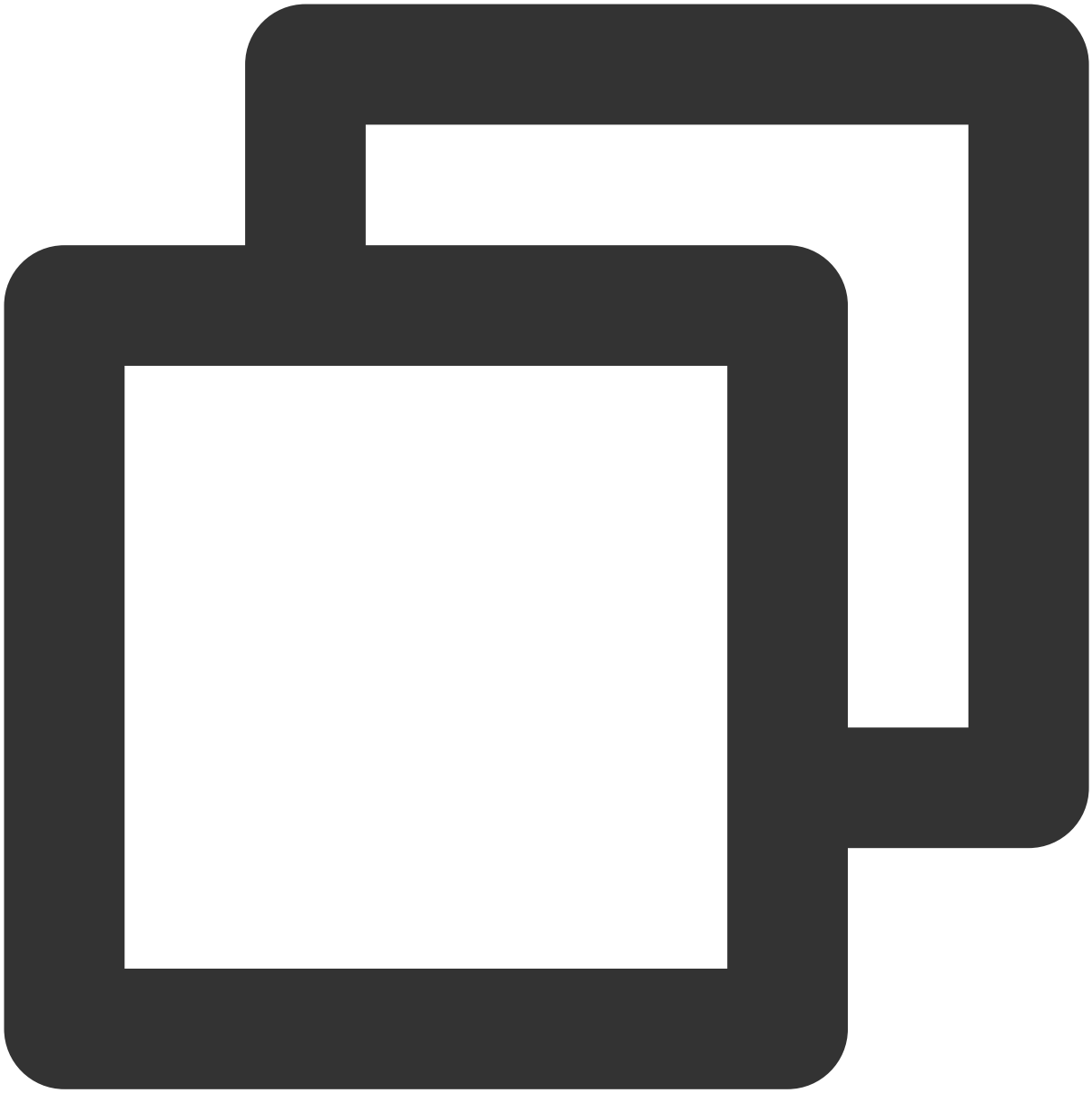




API 说明

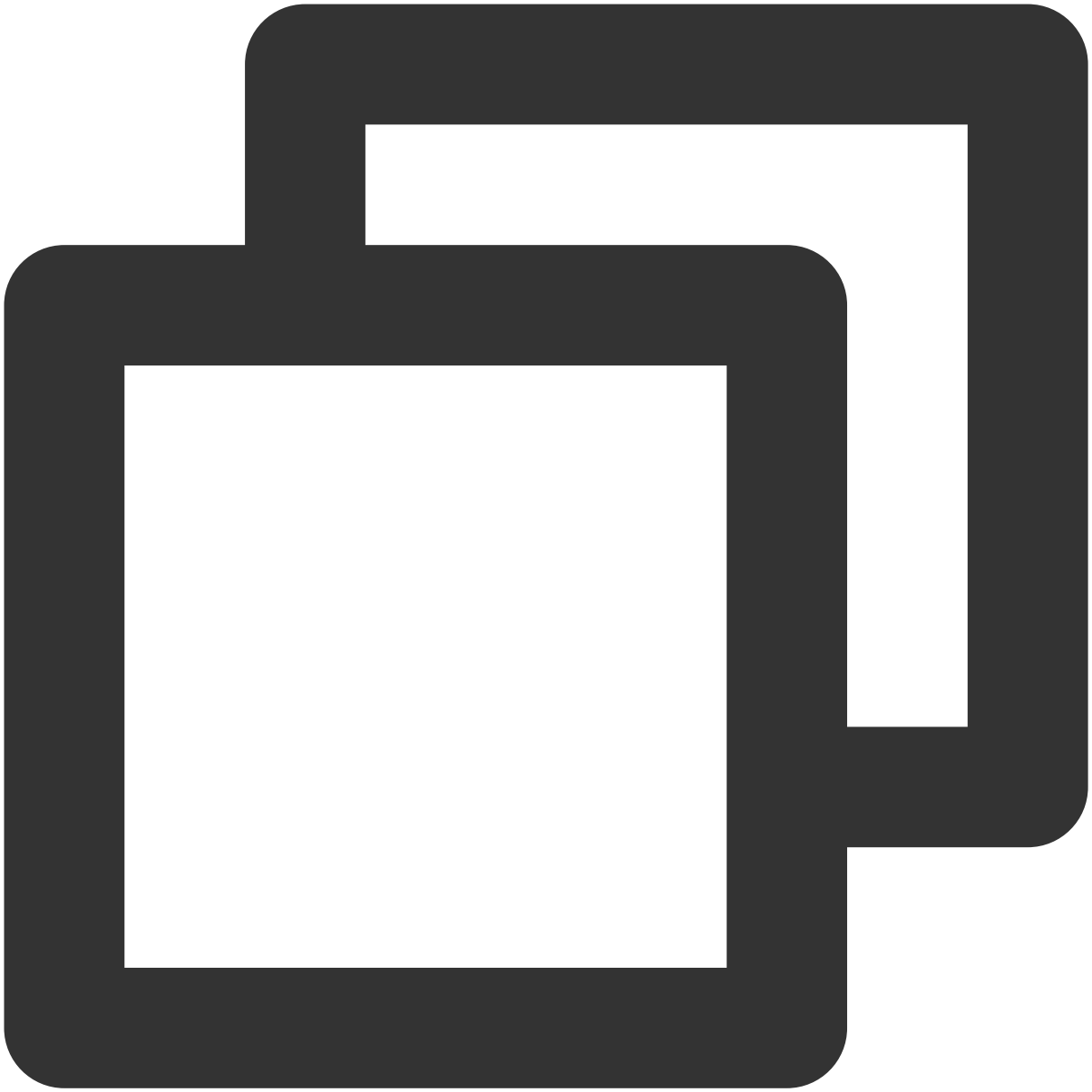
项目令牌调用 CODING 持续集成 API 时所使用的认证方式为 `Basic Auth`，下面是关于 API 接口的详细信息和相关参数。

触发构建任务



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

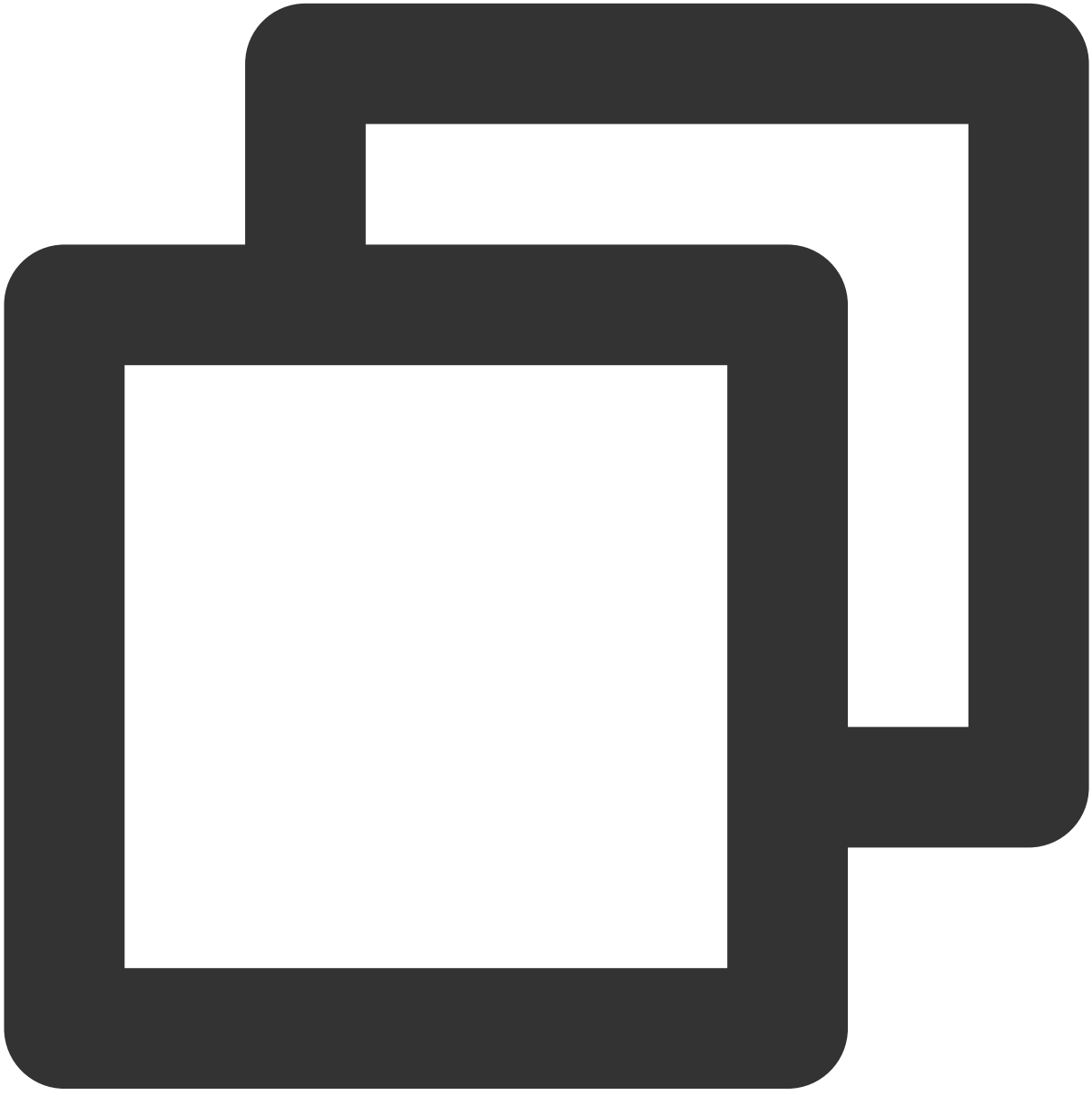
请求 body



```
{
  "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

==== 2021/08/24 ====

环境变量

最近更新时间：2023-12-29 11:44:51

title: 环境变量 - CODING 帮助中心
pageTitle: 环境变量
pagePrevTitle: 触发规则
pagePrev: ci/configuration/trigger.html
pageNextTitle: 构建快照
pageNext: ci/configuration/snapshot.html
alias:
devops/ci/env.html
ci/env.html

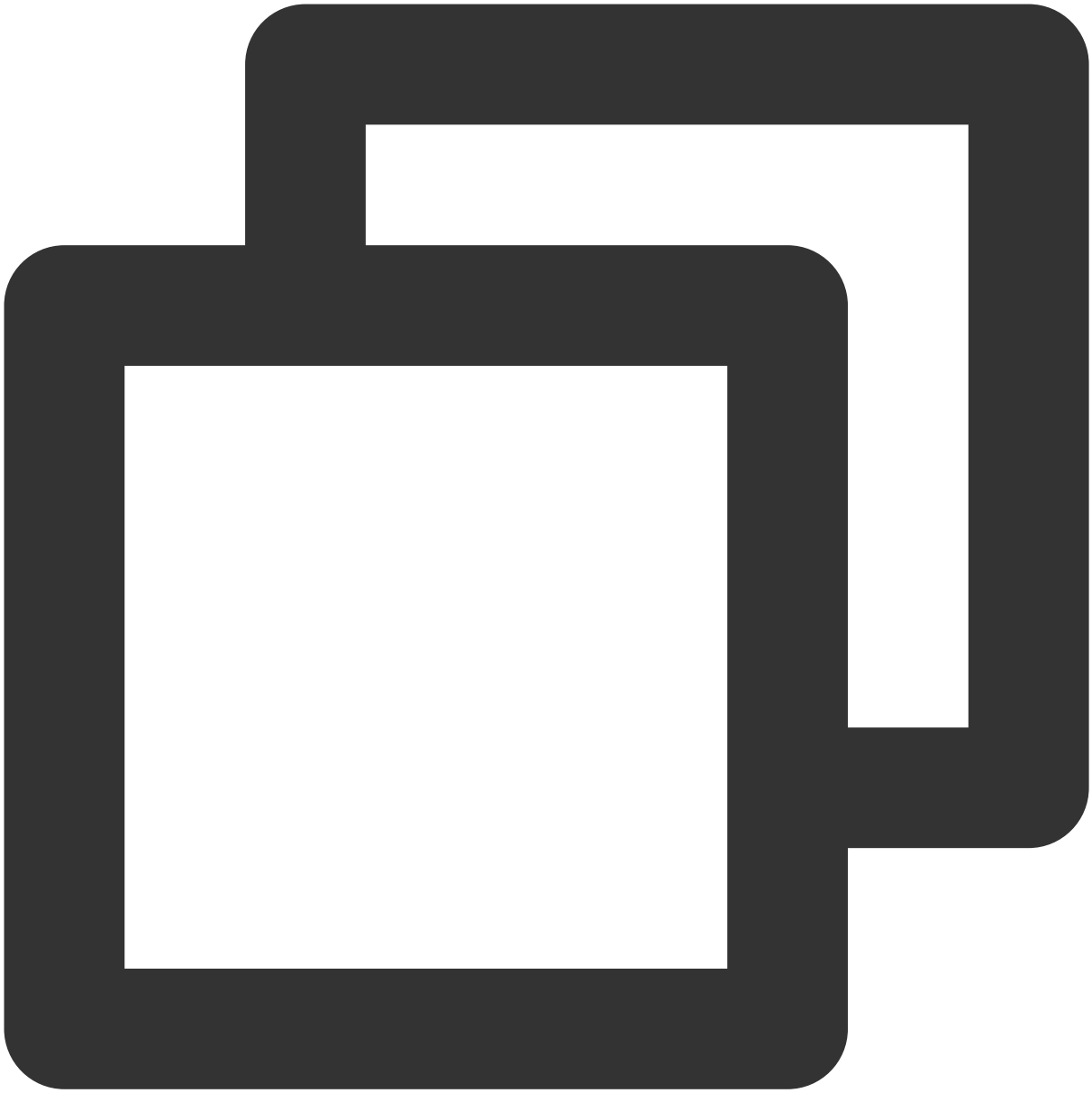
功能介绍

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

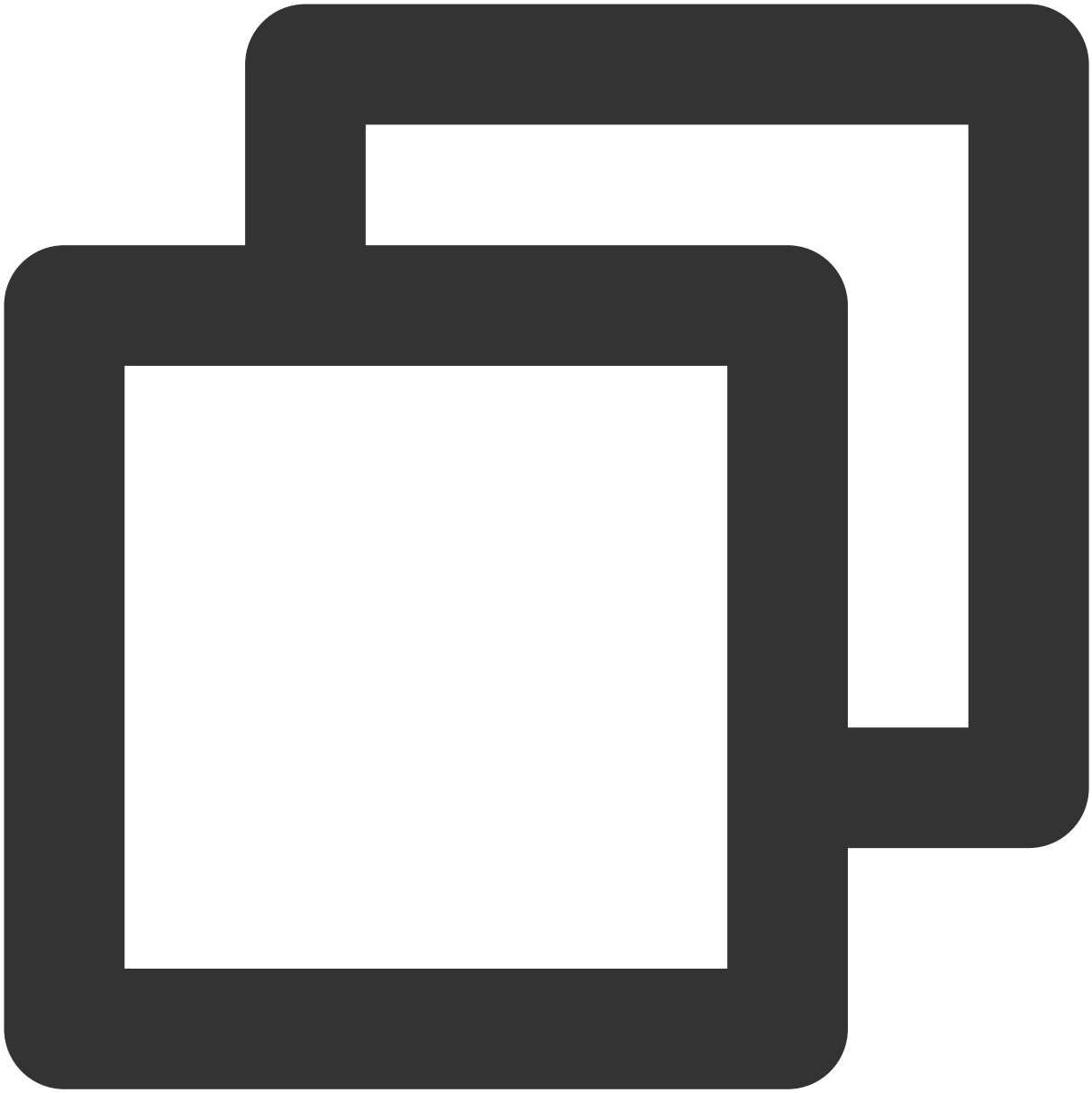
您可以在 Jenkinsfile 中使用 environment 来定义环境变量（如下所示）：



```
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
            }
        }
    }
}
```

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}"
```

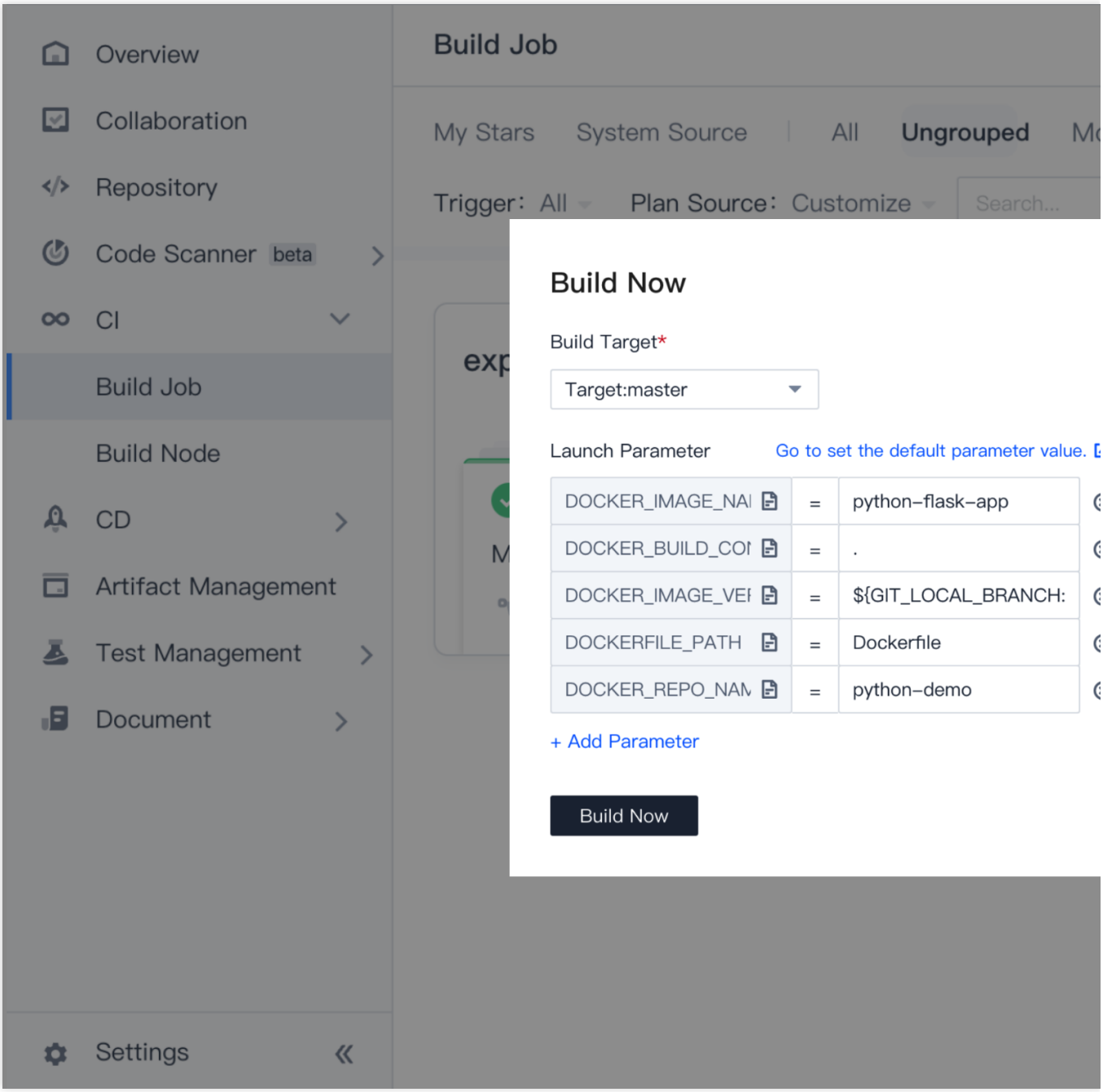
```
                echo "MY_TEAM is ${MY_TEAM}"
                // 输出内容如下所示：
                // MY_PROJECT is project-1
                // MY_TEAM is team-1

                // 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

                }
            }
        }
    }
}
```

说明：

如果您想了解更多 Jenkinsfile 环境变量相关内容，可以拓展阅读：《Jenkins 官方文档——使用环境变量》

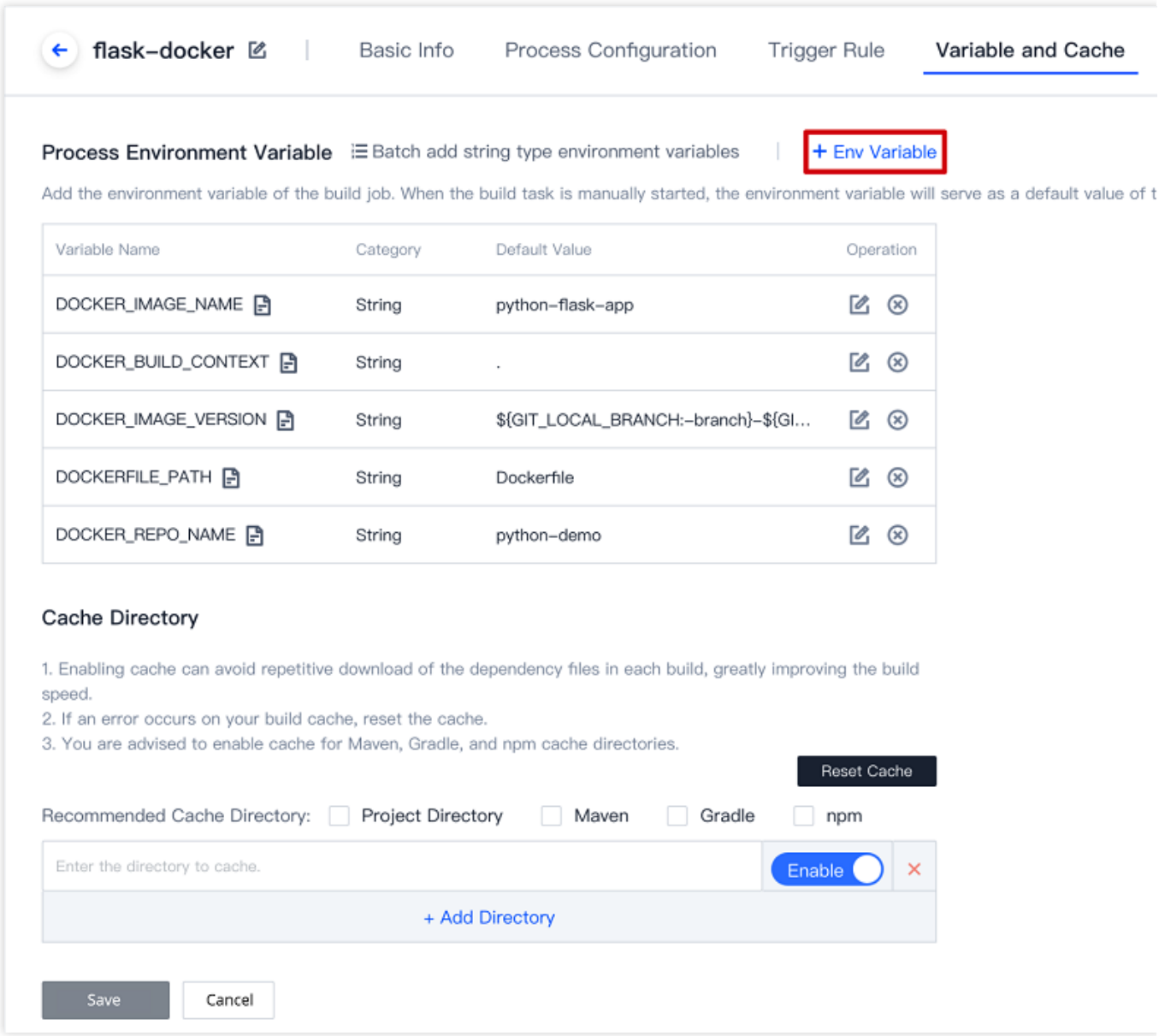
构建计划中的启动参数

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。



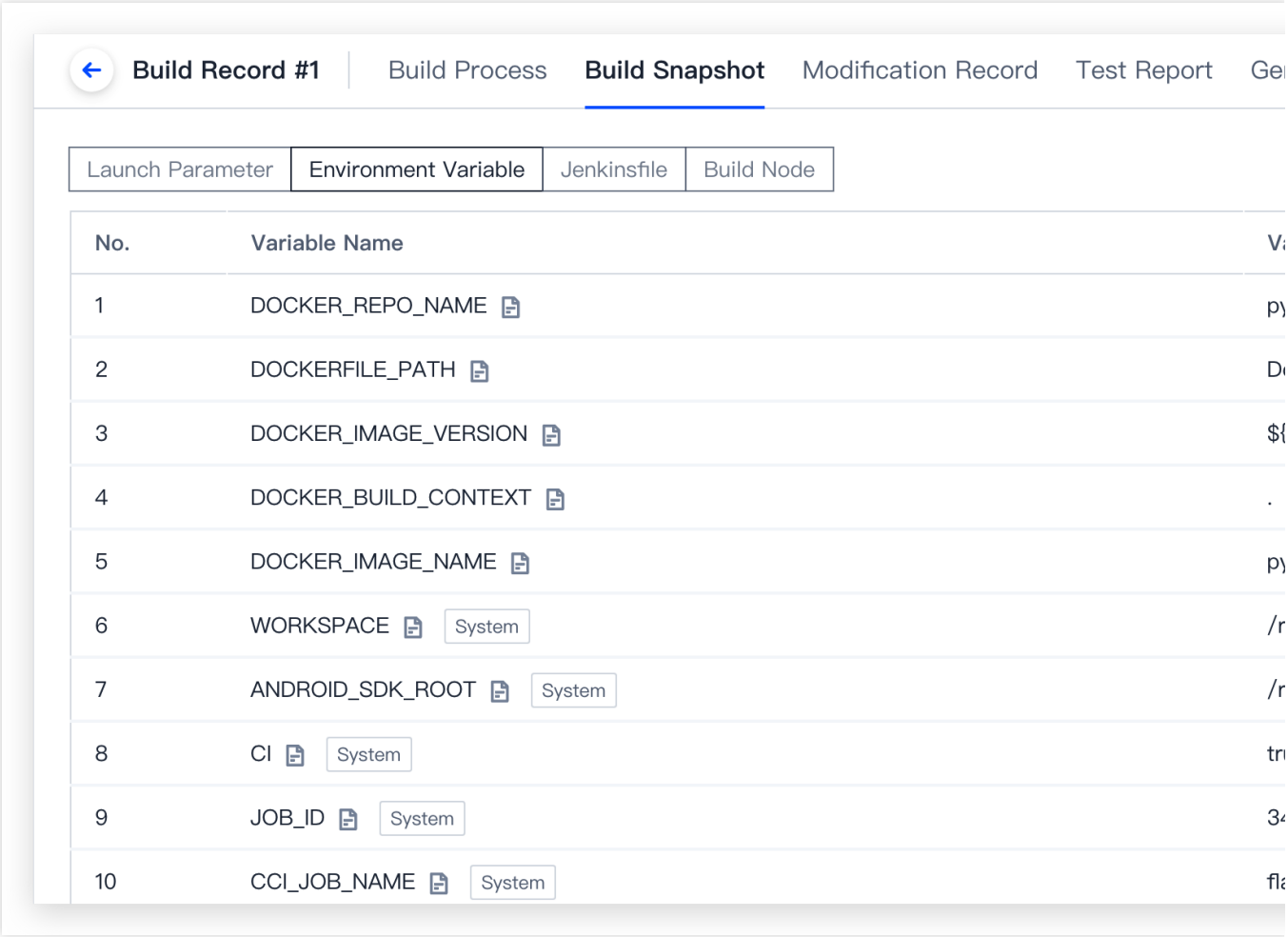
添加环境变量

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
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_I
3	CODING_ARTIFACTS_CREDENTIALS_ID	制品库私钥凭据 CredentialsId 用于拉取项内的制品库
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. 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

==== 2020/10/14 ====

构建快照

最近更新时间：2023-12-29 11:44:51

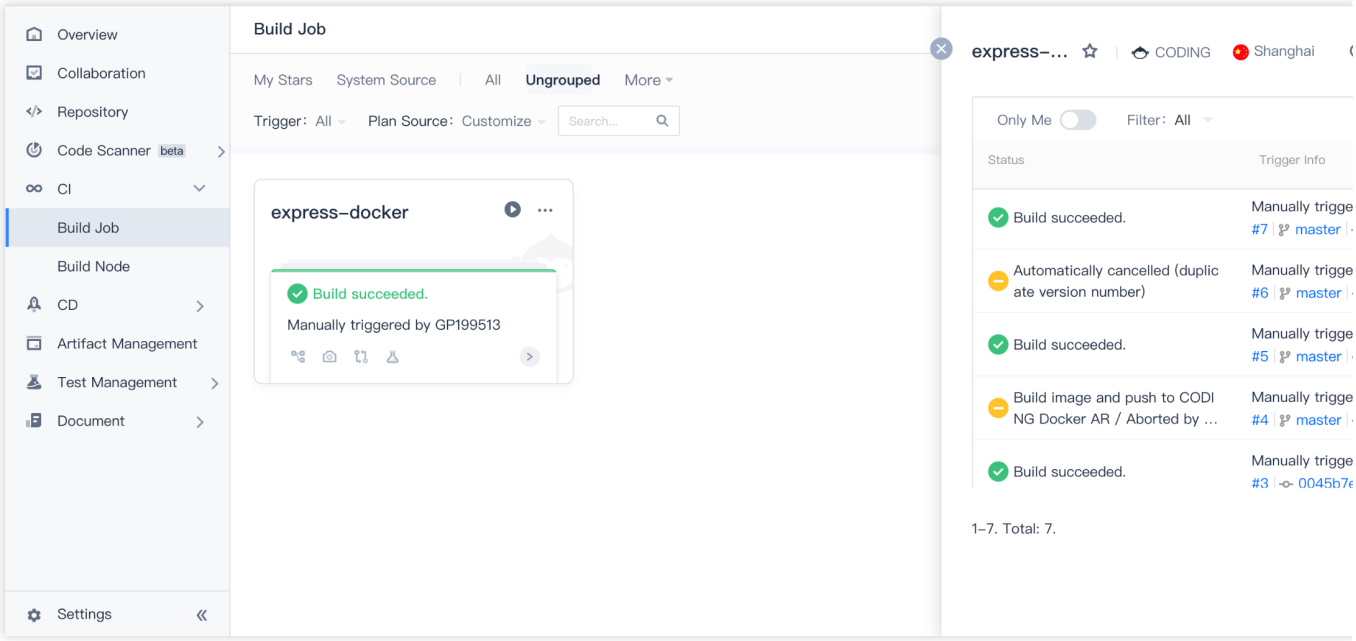
title: 构建快照 - CODING 帮助中心
pageTitle: 构建快照
pagePrevTitle: 环境变量
pagePrev: ci/configuration/env.html
pageNextTitle: 缓存目录
pageNext: ci/configuration/cache.html
alias:
devops/ci/snapshot.html
ci/snapshot.html

功能介绍

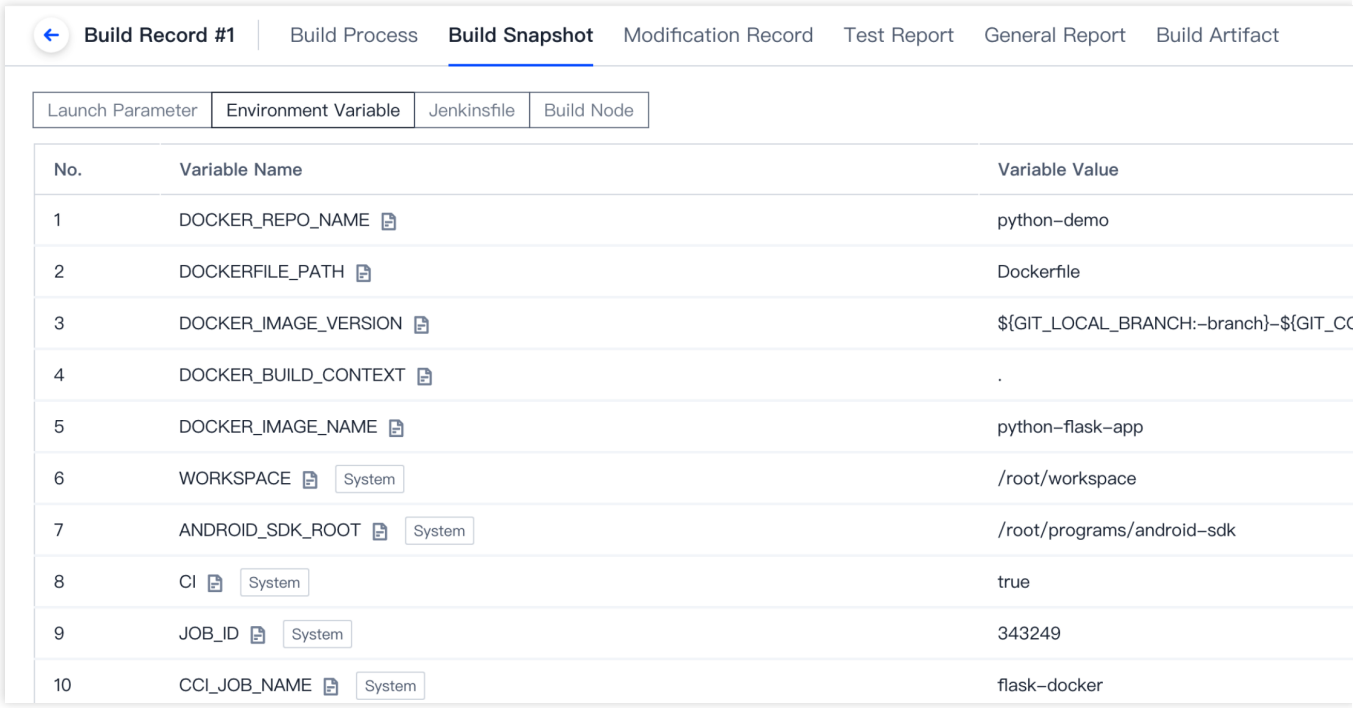
您在持续集成的每一个构建任务，都有可能使用到不同的配置文件或构建参数，为了方便您回顾构建任务的执行过程，CODING 持续集成提供了构建任务的构建快照查看功能。构建快照功能让您能清晰地了解到每次构建记录的配置参数。

查看构建配置

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

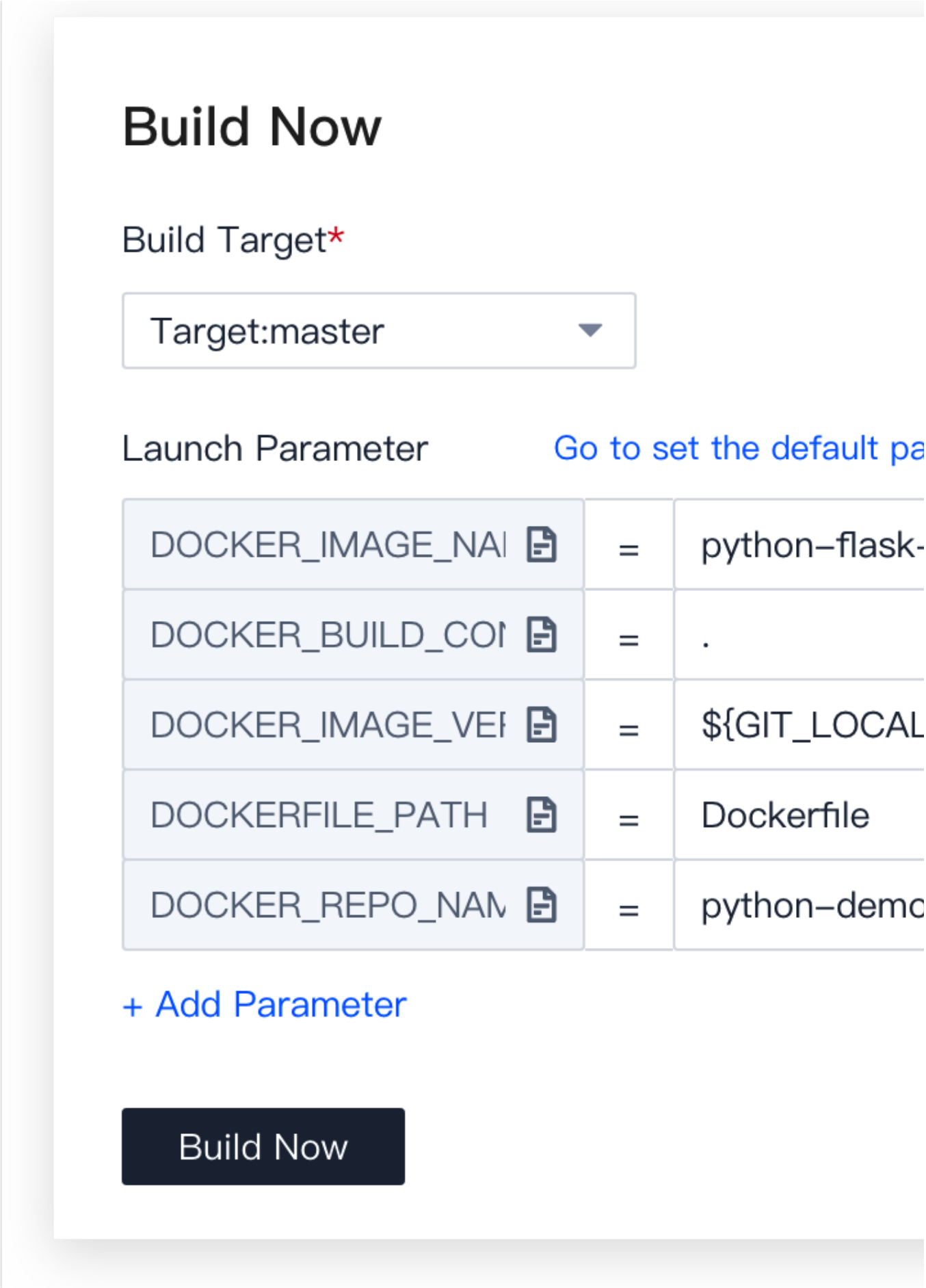


2. 在构建记录中，点击【构建快照】就可以看到每次构建记录的配置快照：启动参数、环境变量、流程配置文件。

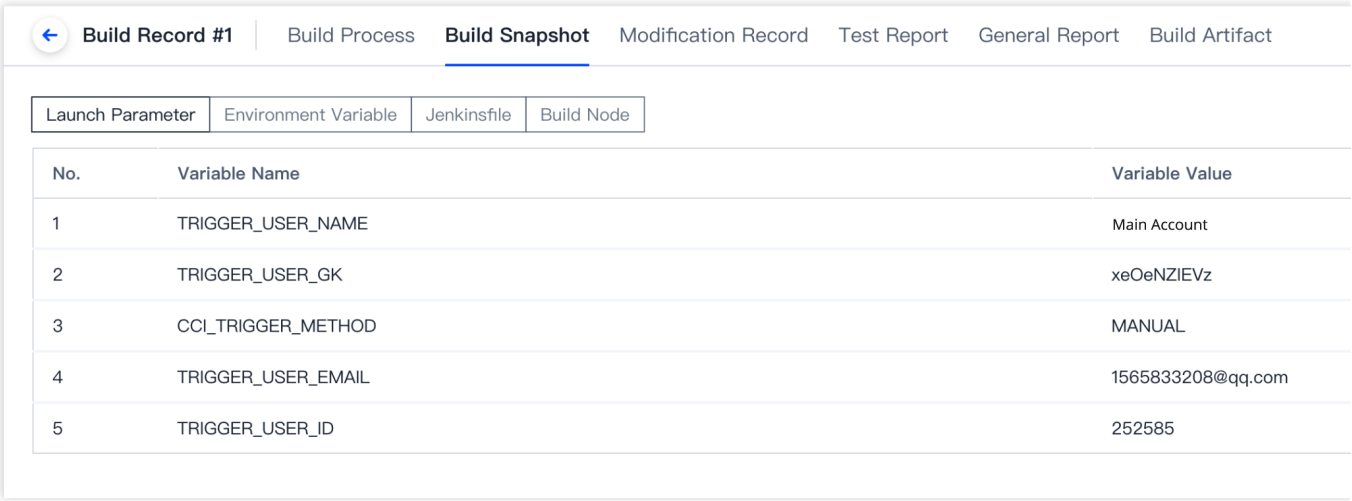


启动参数

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。

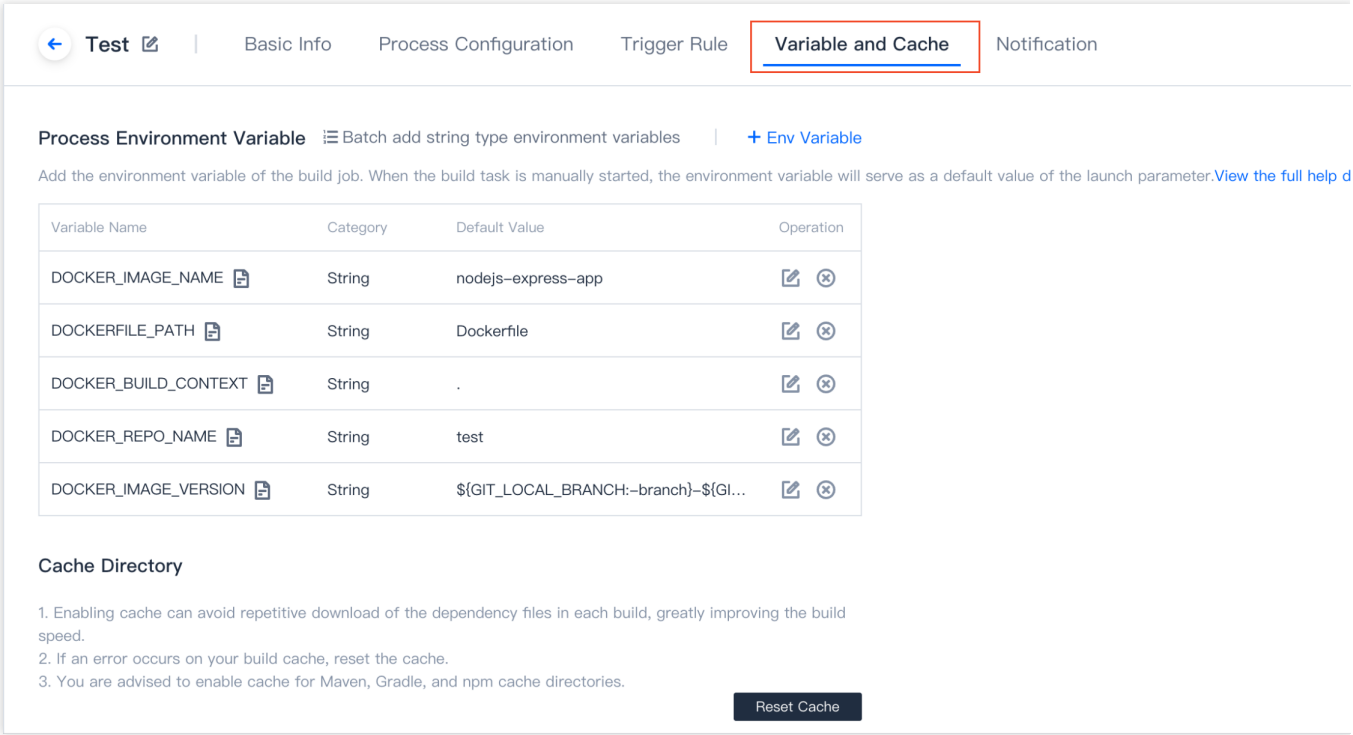

Build Now
Build Target*
Target:master
Launch Parameter
Go to set the default pa
DOCKER_IMAGE_NAI
=
python-flask-
DOCKER_BUILD_CON
=
.
DOCKER_IMAGE_VEI
=
${GIT_LOCAL
DOCKERFILE_PATH
=
Dockerfile
DOCKER_REPO_NAM
=
python-demo
+ Add Parameter
Build Now

在构建完成后，您可以在构建快照中看到配置的启动参数。

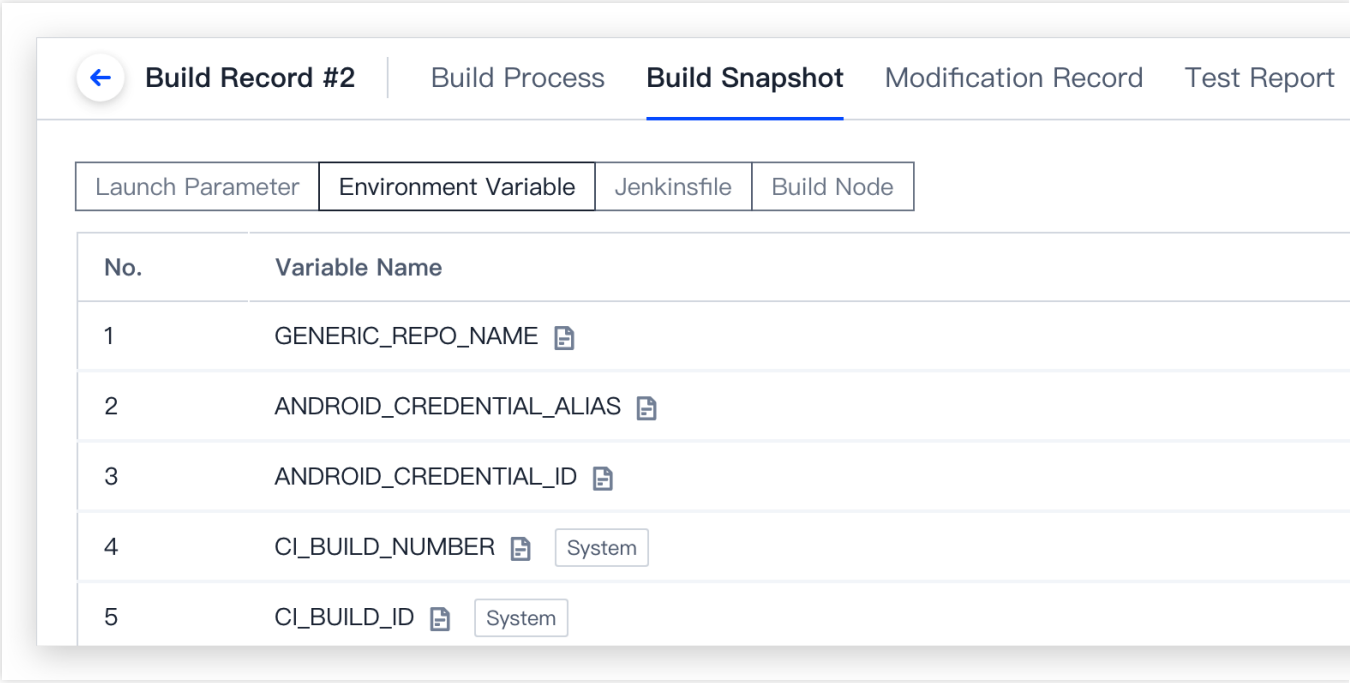


环境变量

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。

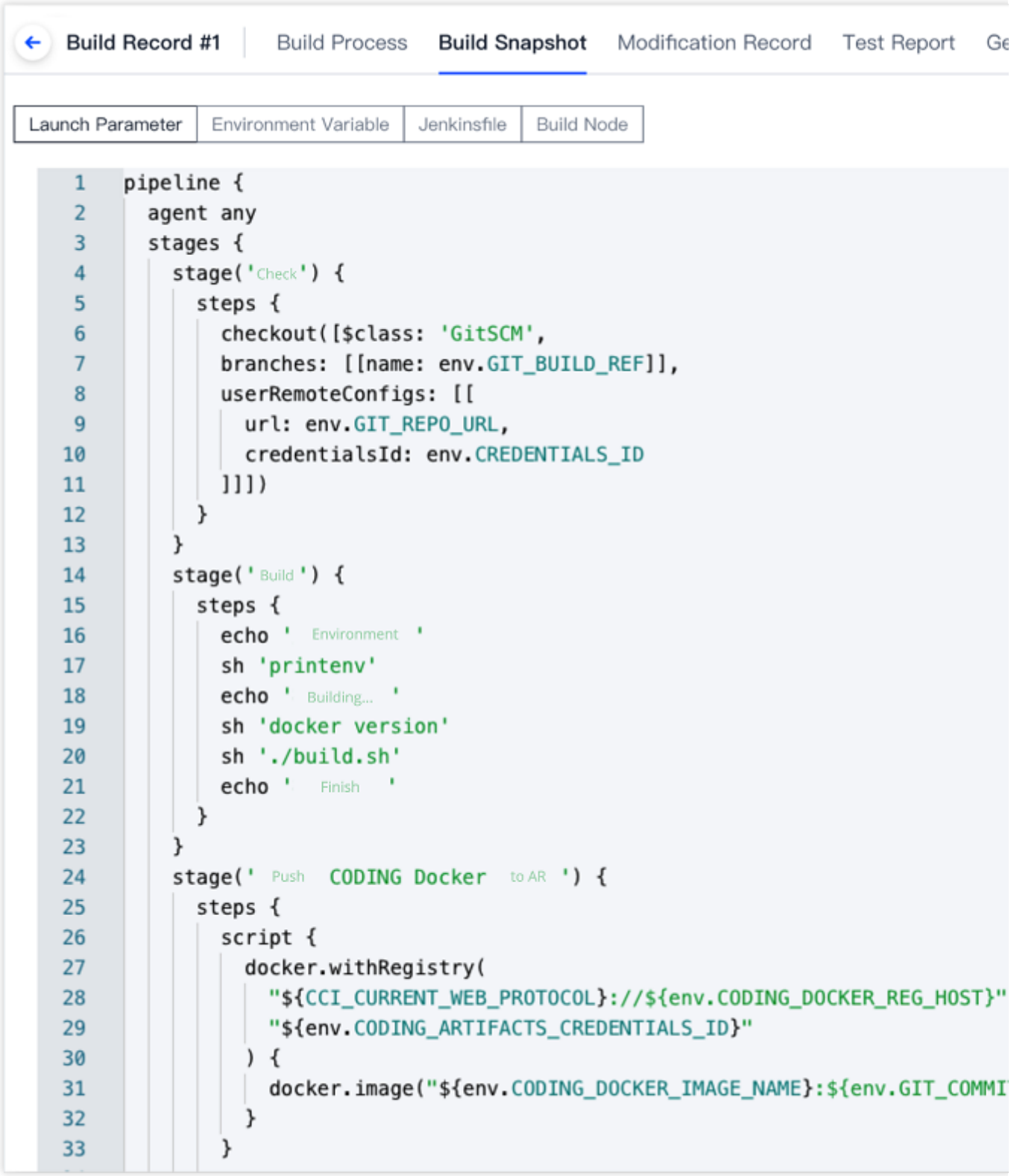


在环境变量选项卡内，用户可以参看到任务启动时，系统和用户为构建任务设置的环境变量。



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）。


Build Record #1
Build Process
Build Snapshot
Modification Record
Test Report
Launch Parameter
Environment Variable
Jenkinsfile
Build Node
pipeline {
agent any
stages {
stage('Check') {
steps {
checkout([$class: 'GitSCM',
branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[
url: env.GIT_REPO_URL,
credentialsId: env.CREDENTIALS_ID
]]])
}
}
stage('Build') {
steps {
echo ' Environment '
sh 'printenv'
echo ' Building... '
sh 'docker version'
sh './build.sh'
echo ' Finish '
}
}
stage(' Push CODING Docker to AR ') {
steps {
script {
docker.withRegistry(
"${CCI_CURRENT_WEB_PROTOCOL}://${env.CODING_DOCKER_REG_HOST}"
"${env.CODING_ARTIFACTS_CREDENTIALS_ID}"
) {
docker.image("${env.CODING_DOCKER_IMAGE_NAME}:${env.GIT_COMMI
}
}

==== 2020/08/13 ====

缓存目录

最近更新时间：2023-12-29 11:44:51

title: 缓存目录 - CODING 帮助中心
pageTitle: 缓存目录
pagePrevTitle: 构建快照
pagePrev: ci/configuration/snapshot.html
pageNextTitle: 构建节点类型
pageNext: ci/node/type.html
alias:
devops/ci/cache.html
ci/cache.html
practice/jenkins-dockerfile.html

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。比如使用 `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」。
如果使用在持续集成中使用 Docker 时需要把「缓存目录」挂载至 Docker 中。

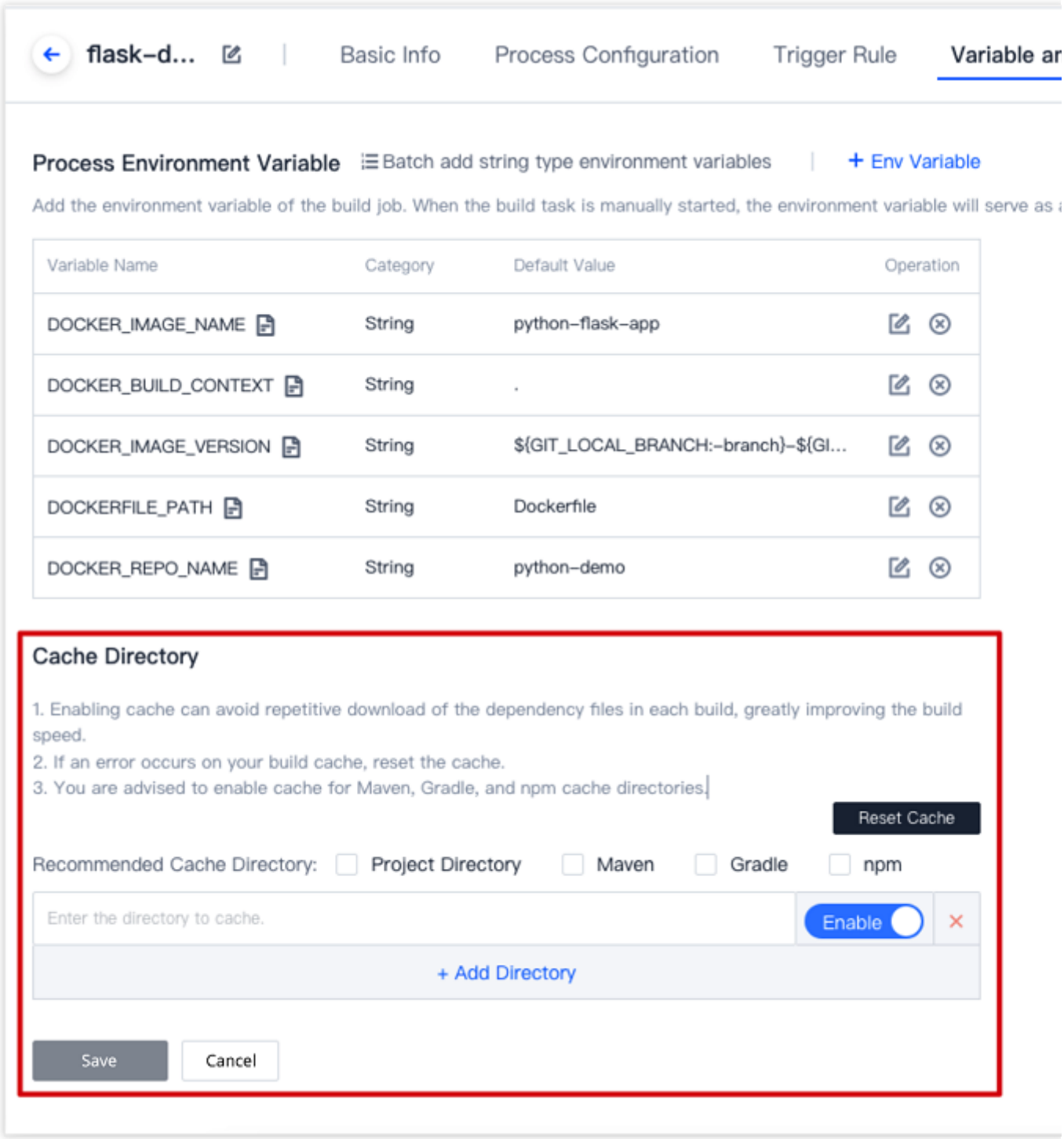
默认构建节点

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主机，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/

pip3	/root/.cache/pip/
yarn	/usr/local/share/.cache/yarn/

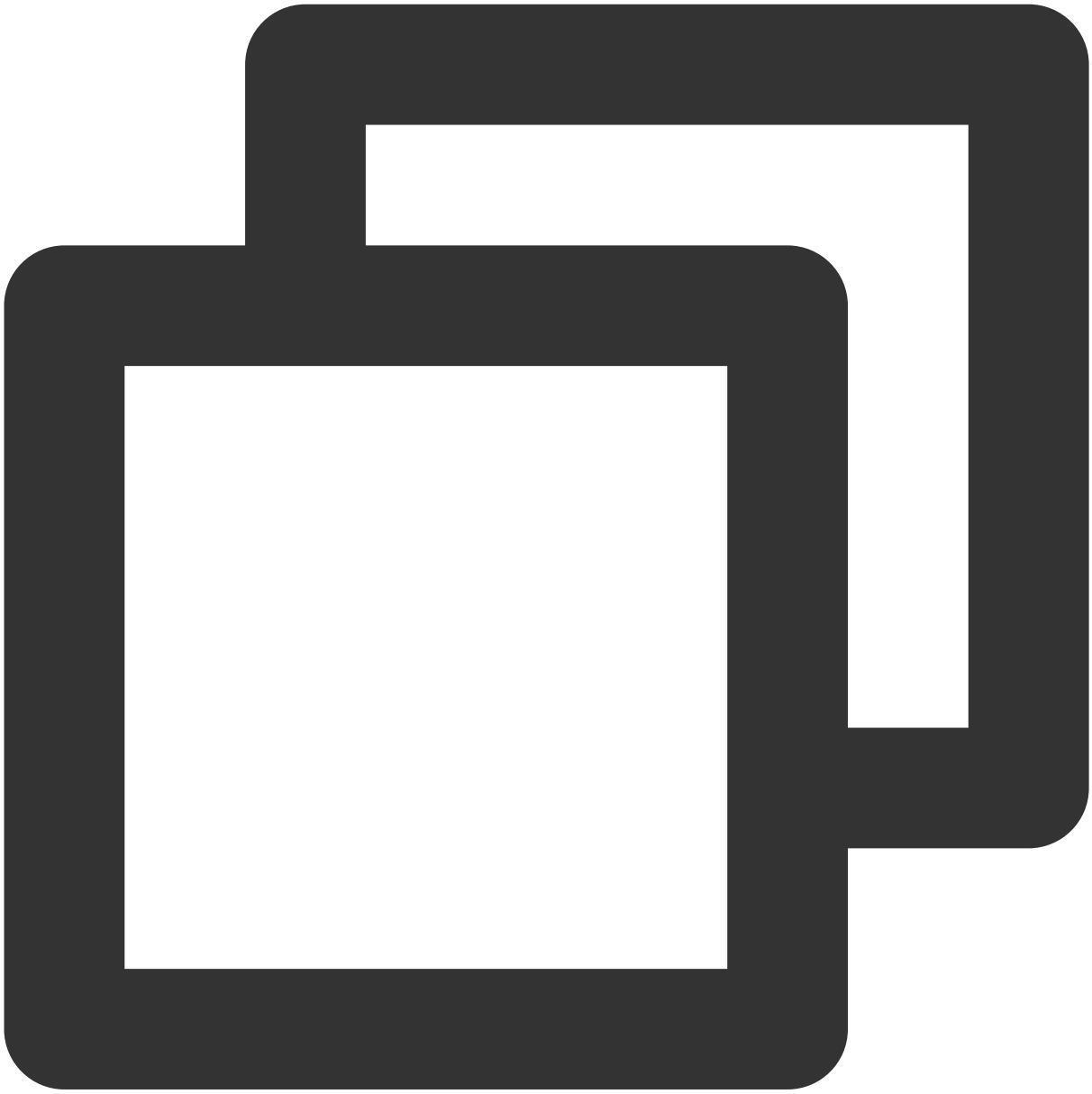
你可以在构建计划设置中的「变量与缓存」中勾选缓存目录，如果未找到目标目录还支持自行录入。



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

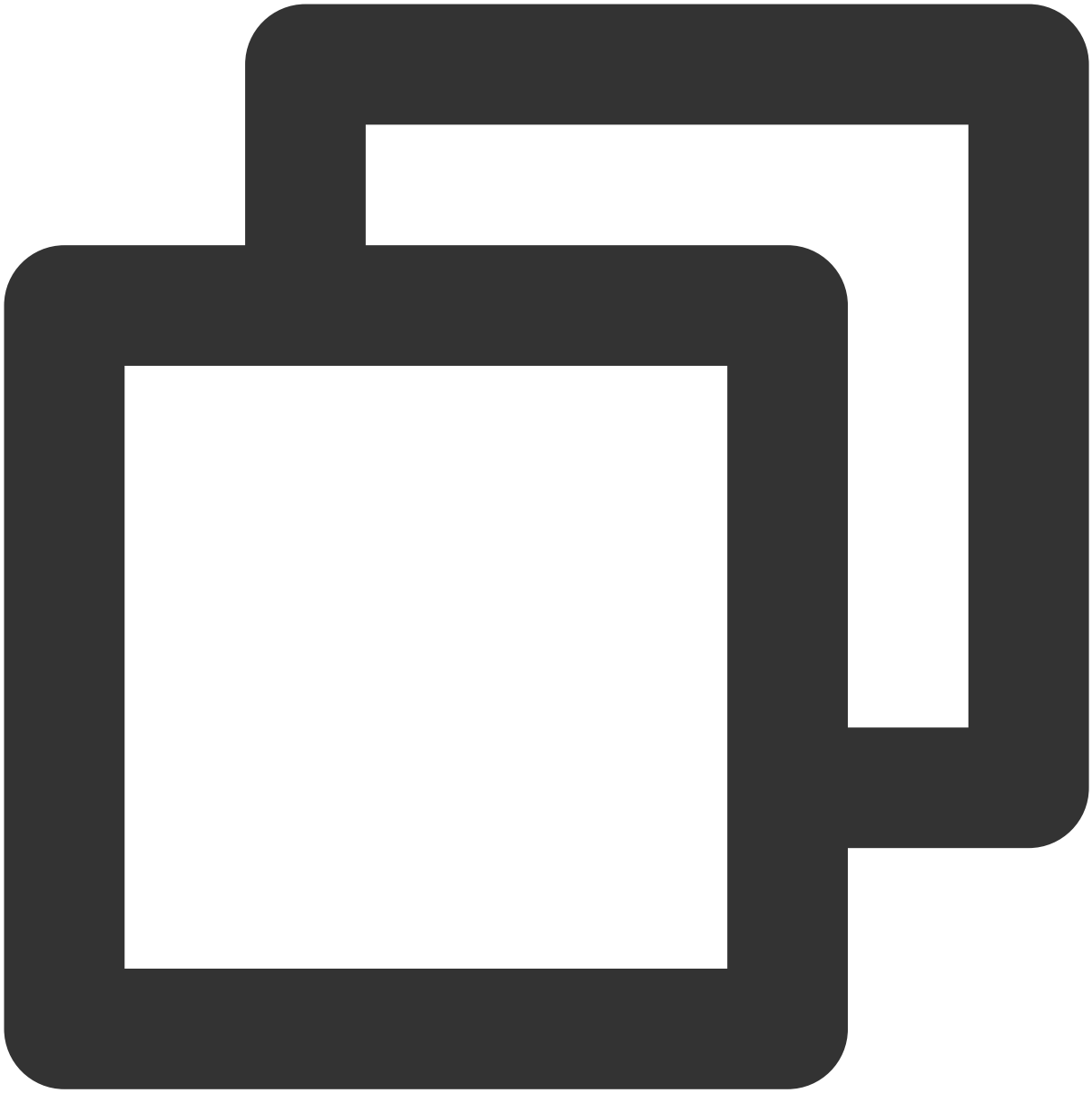


```
pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
```

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

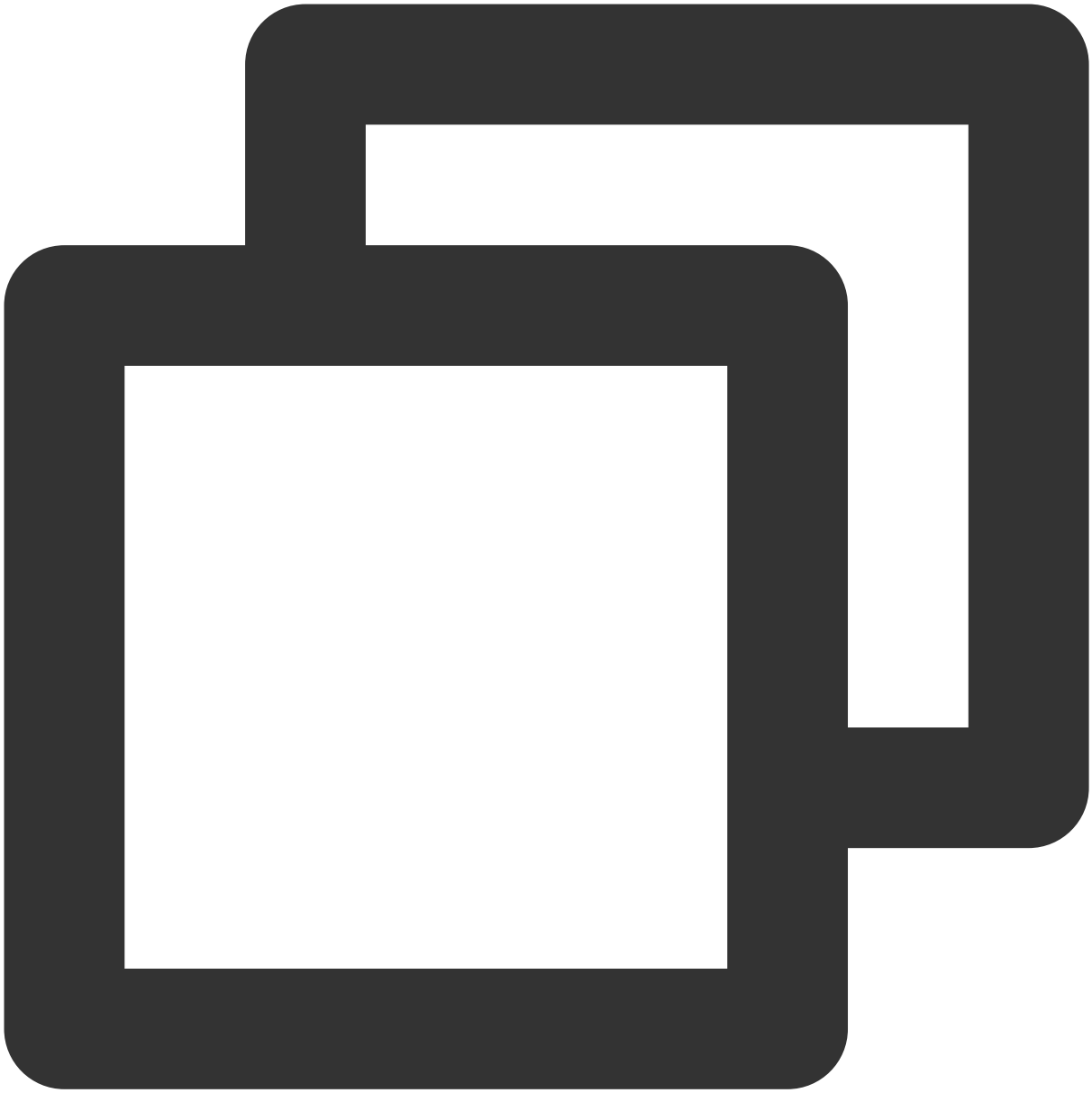
在自定义构建节点中使用 Docker 环境时需按照服务器用户名找到对应的缓存目录，例如当 Ubuntu 服务器的默认用户名为 ubuntu 时，缓存目录为 `/home/ubuntu/.npm/`，那么对应的代码为：



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

缓存 Docker 基础镜像

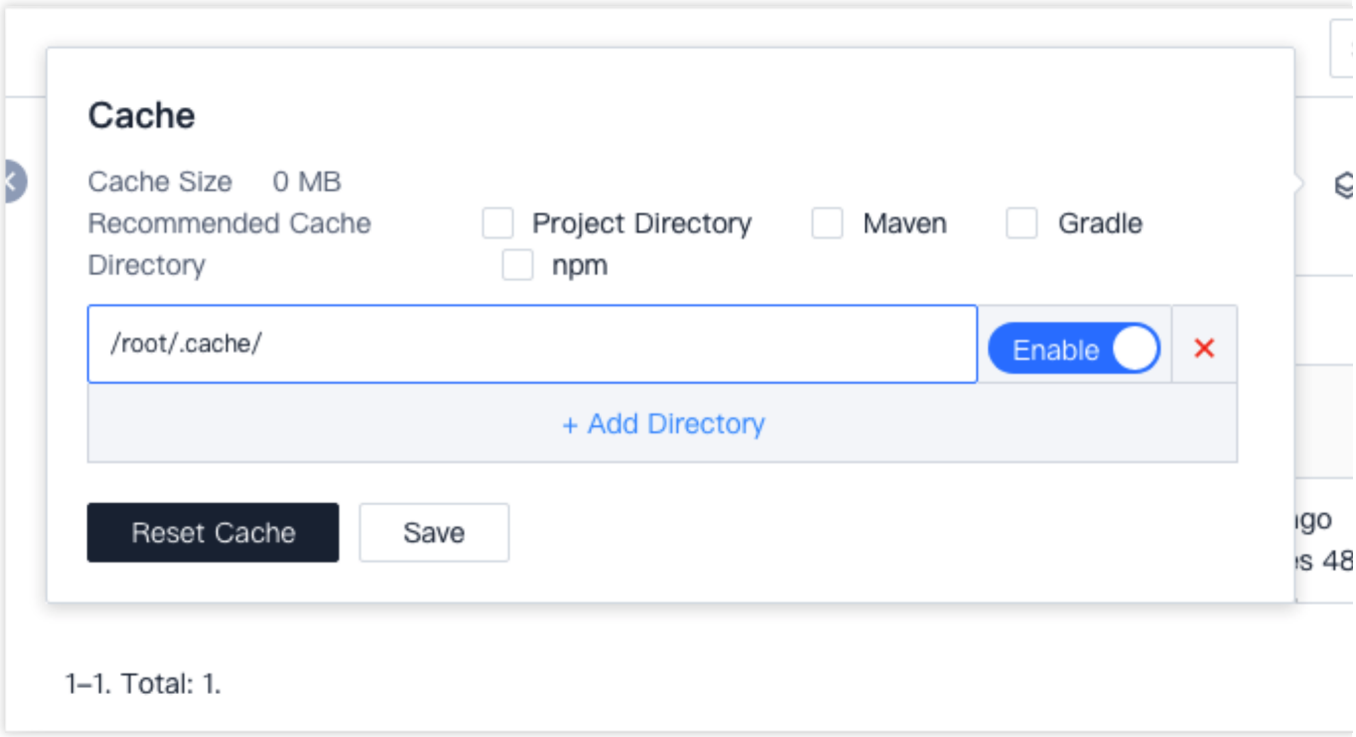
如果每次构建都需要拉取 Docker 基础镜像，例如 `Dockerfile` 基础镜像、 CI agent 镜像，那么通常会耗费大量时间，此时就可以通过缓存进行加速。
参考以下 `Jenkinsfile` ，修改镜像名称即可复用：



```
pipeline {
  agent any
  environment{
    DOCKER_CACHE_EXISTS = fileExists '/root/.cache/docker/php-8.0-cli.tar'
  }
  stages {
    stage('加载缓存') {
      when { expression { DOCKER_CACHE_EXISTS == 'true' } }
      steps {
        sh 'docker load -i /root/.cache/docker/php-8.0-cli.tar'
      }
```

```
    }
    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'
      }
    }
  }
}
```

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：

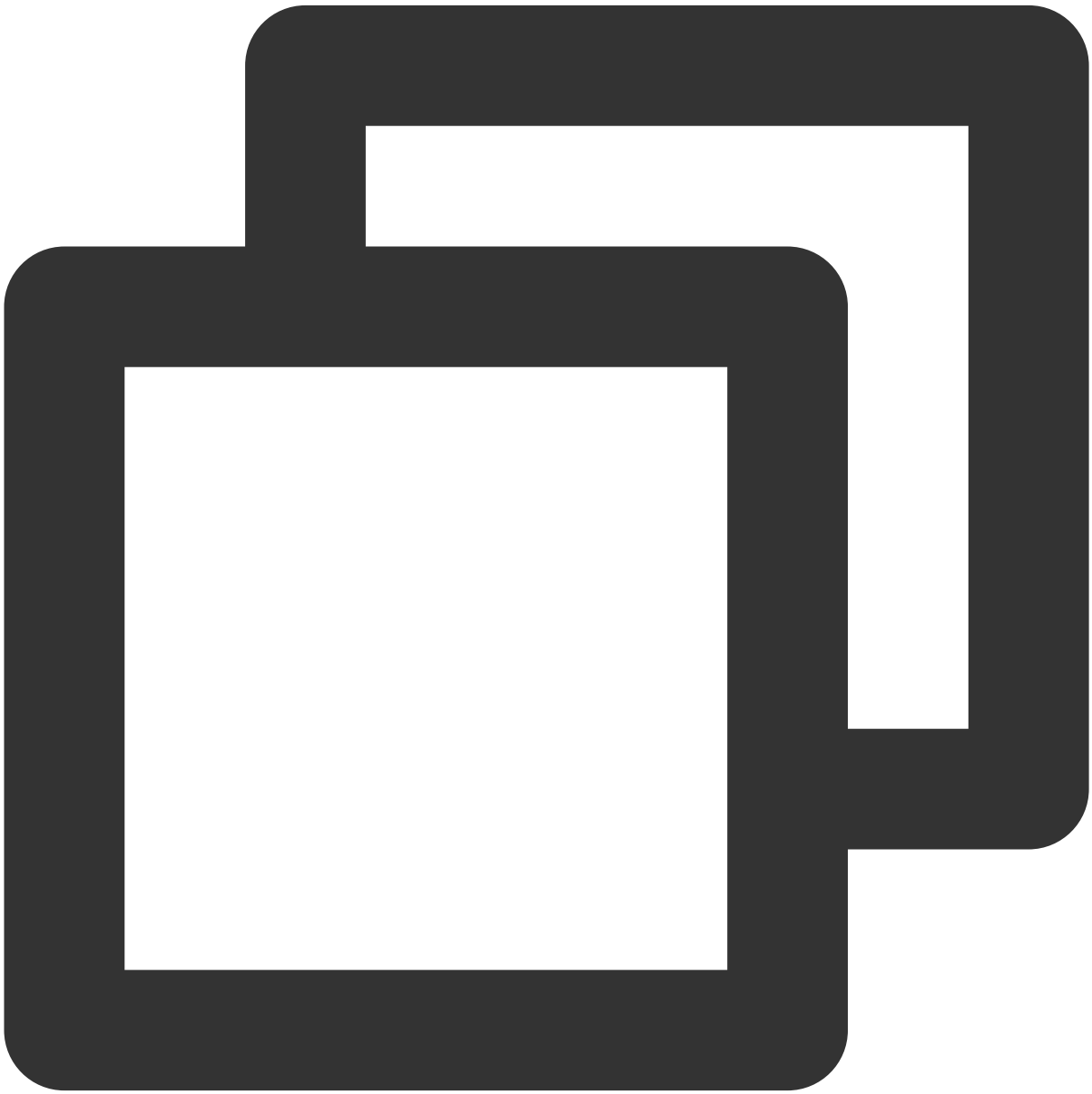


⚠ 缓存镜像会逐渐过时，建议定时清除，与官方更新保持一致。

保存 Dockerfile

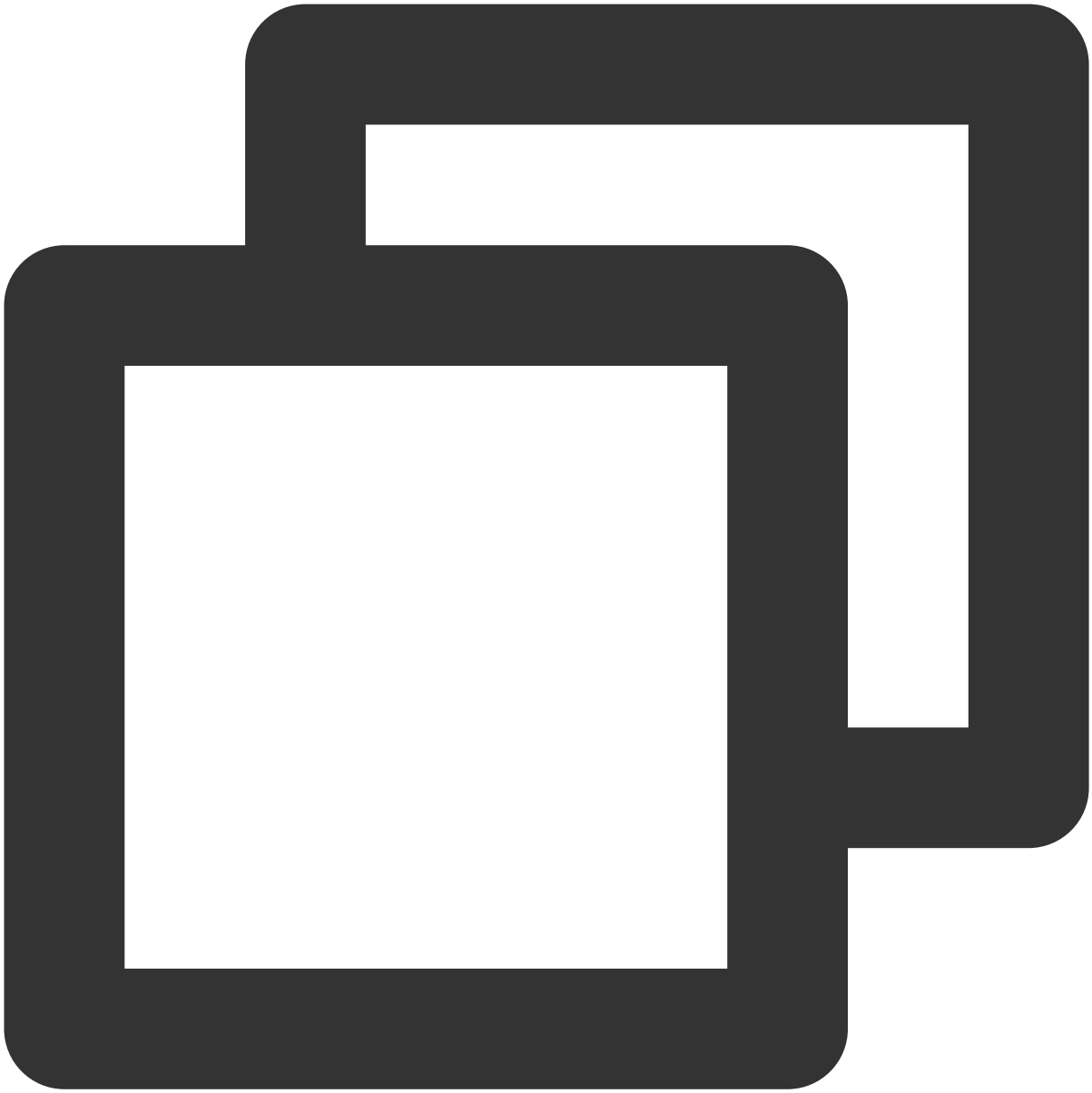
在持续集成中使用 `Dockerfile` 作为构建环境，需要运行 `docker build` 命令用以初始化，较为不便。可以将已构建的 Docker 镜像保存到仓库，方便二次拉取复用。

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 $imageFullName > /dev/null",
def testImage = null
if (dockerNotExists) {
    testImage = docker.build("$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「镜像是否已存在」。



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

==== 2021/09/17 ====

构建制品

Docker

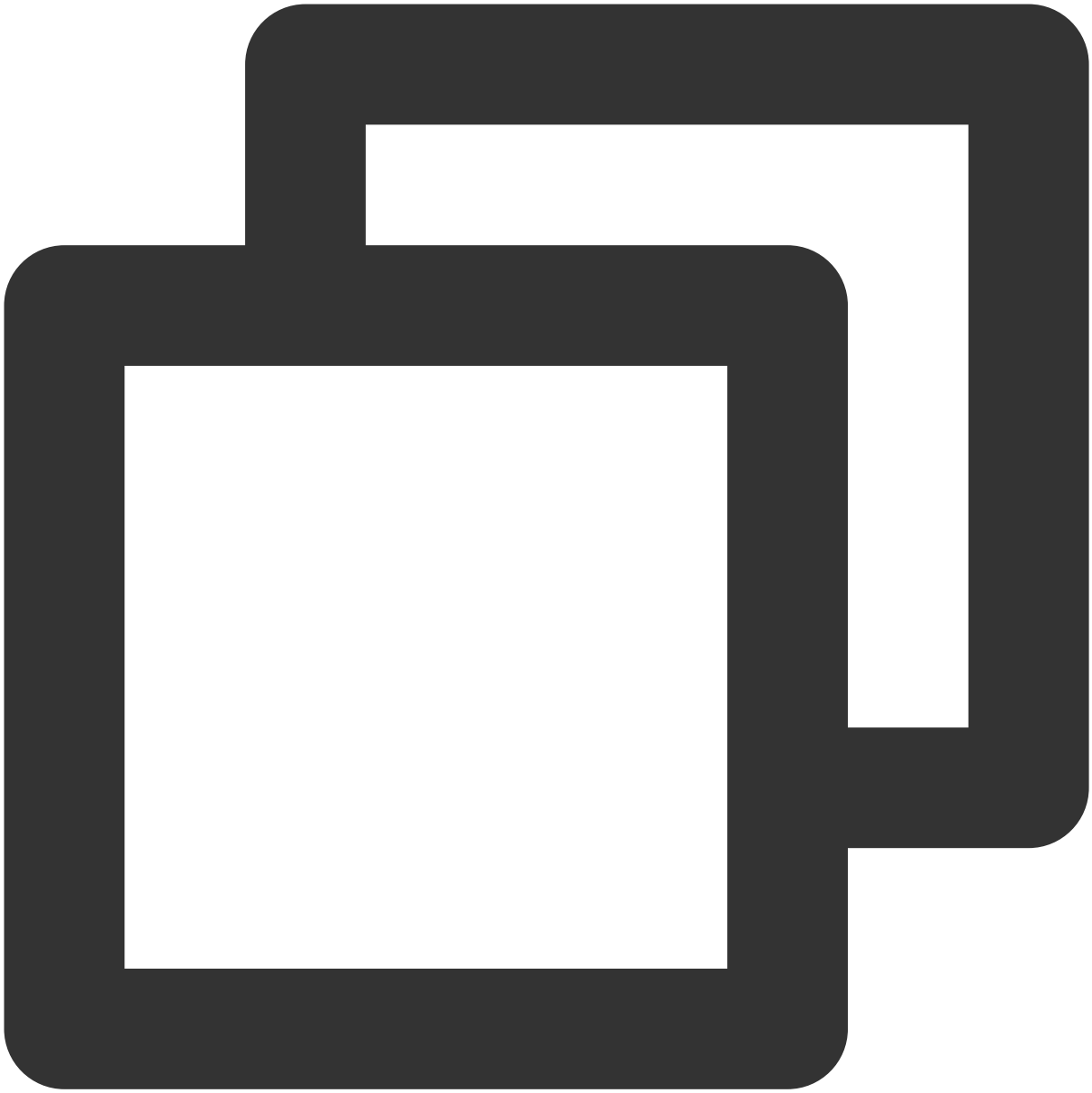
最近更新时间：2023-12-29 11:44:51

title: Docker - CODING 帮助中心
pageTitle: Docker
pagePrevTitle: 构建 Composer 制品
pagePrev: ci/artifacts/composer.html
pageNextTitle: 构建文件类型制品
pageNext: ci/artifacts/generic.html
alias:
devops/ci/artifacts/docker.html
ci/artifacts/docker.html

功能介绍

本文将给出如何使用持续集成任务构建 Docker 镜像的示例 Jenkinsfile。构建完成后可以使用预置插件便捷的上传至 CODING 制品仓库中。在使用该功能之前，请确保您对 Docker 类型制品库有初步了解。

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.CREDENTIAL
        ])
      }
```

```
    }
    stage('构建 Docker 镜像') {
      steps {
        script {
          ARTIFACT_VERSION = "1.2.0"
          // 注意：创建项目时链接标识不要使用下划线，而是连字符，比如 My Project 的标识应为 my
          // 请修改 build/my-api 为你的制品库名称和镜像名称
          CODING_DOCKER_IMAGE_NAME = "${env.PROJECT_NAME.toLowerCase()}/build/my-ap
          // 本项目内的制品库已内置环境变量 CODING_ARTIFACTS_CREDENTIALS_ID，无需手动设置
          docker.withRegistry("https://${env.CCI_CURRENT_TEAM}-docker.pkg.coding.ne
            docker.build("${CODING_DOCKER_IMAGE_NAME}:${ARTIFACT_VERSION}").push()
          }
        }
      }
    }
  }
}
```

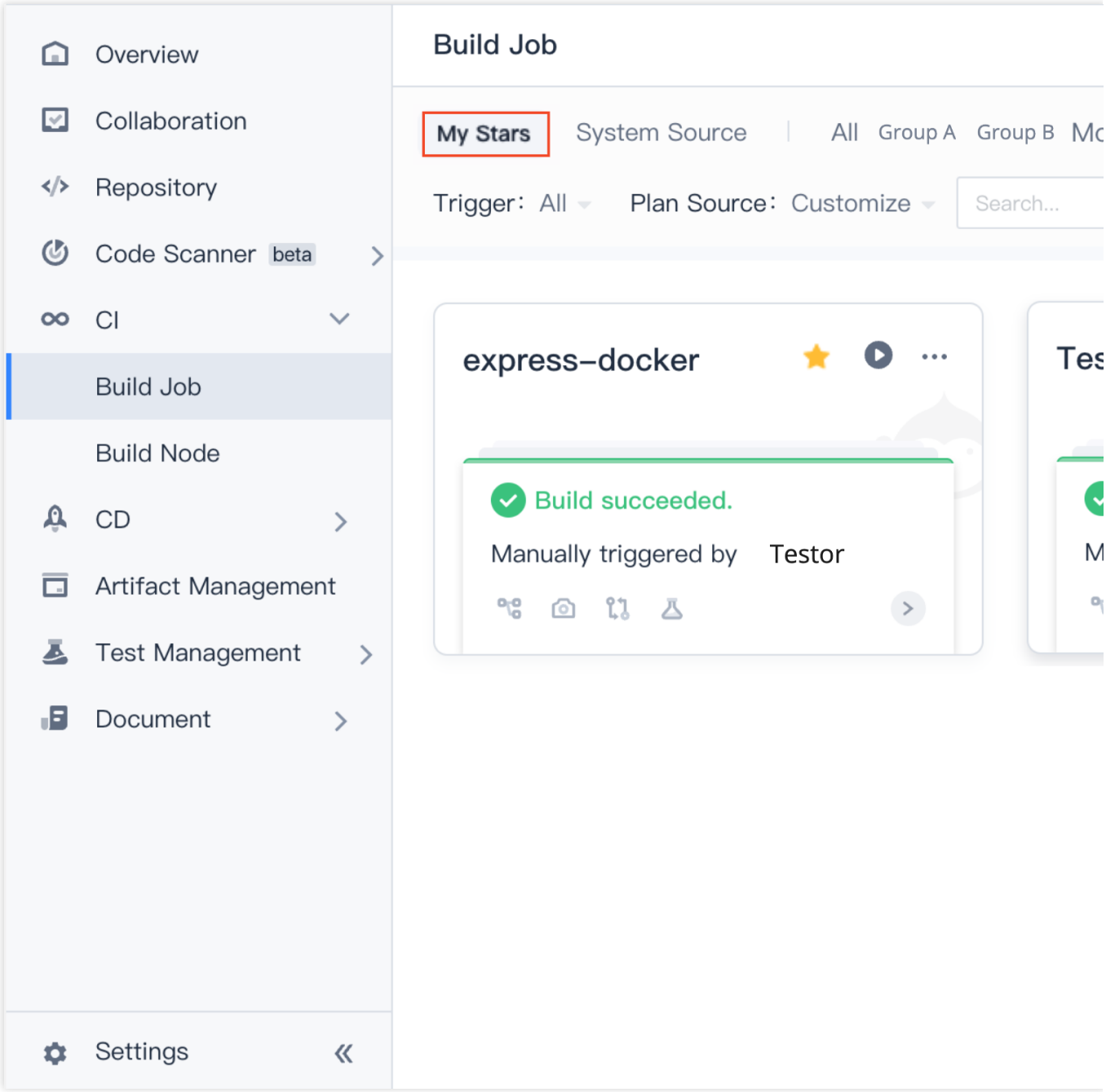
管理构建计划
分组管理

最近更新时间：2023-12-29 11:44:51

title: 分组管理 - CODING 帮助中心
pageTitle: 分组管理
pagePrevTitle: 构建节点池
pagePrev: ci/node/pool.html
pageNextTitle: 团队构建模板
pageNext: ci/manage/team-template.html
alias:
devops/ci/group.html
ci/group.html

星标与分组

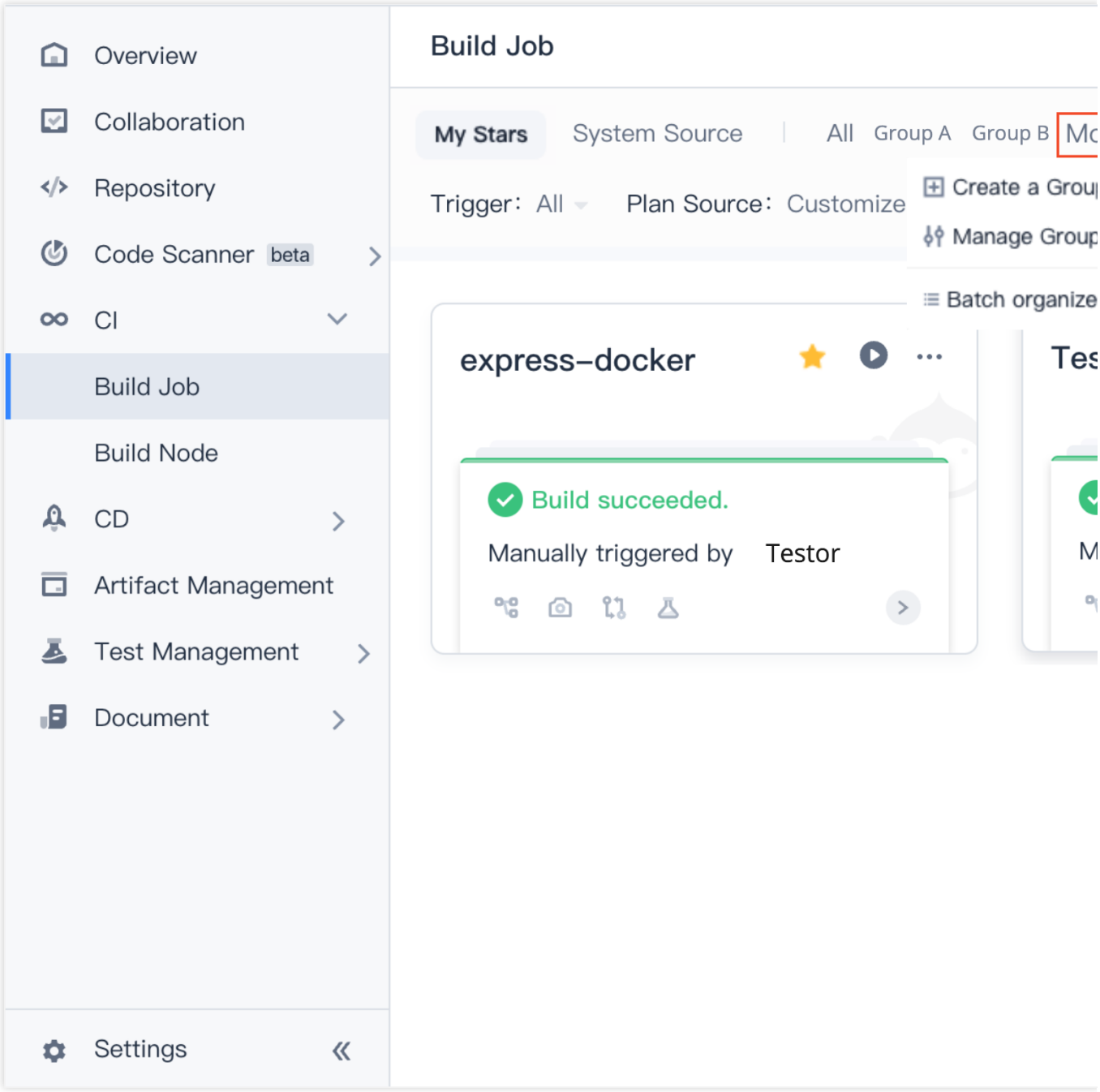
构建计划还支持星标与分组，可以帮助快速定位到自己关注的构建计划。
星标功能
个人选项，设置后仅对个人生效。点击单个构建计划区域中的星标按钮后，可以在「我的星标」 tab 栏中仅查看星标构建计划。



分组功能

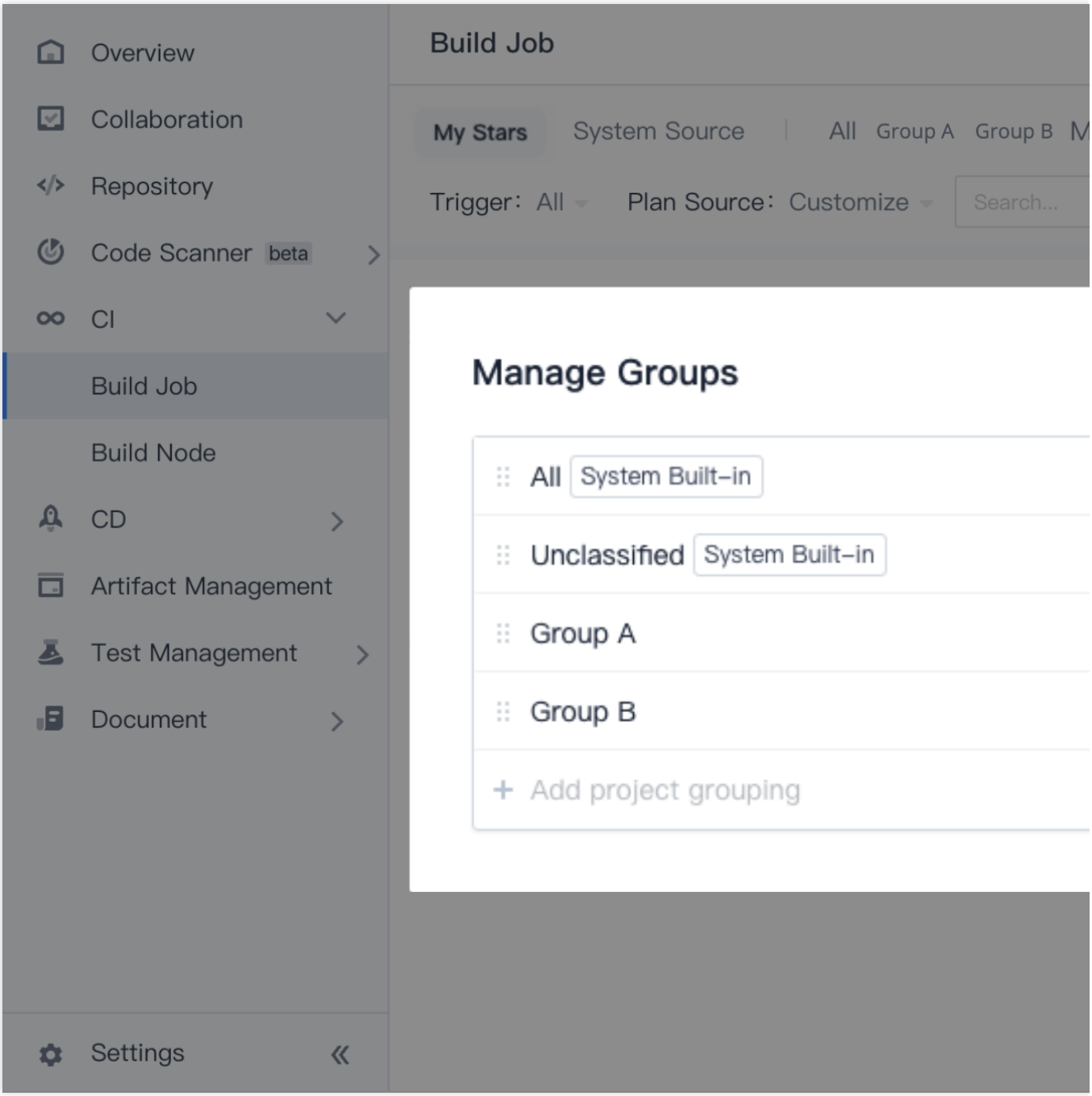
全局选项，仅开放给「持续集成管理」权限的用户。设置后的构建计划分组归类对项目内内成员可见，方便项目内构建计划的整理。

点击【更多】->【创建分组】可以创建分组，点击后输入分组名即可创建。

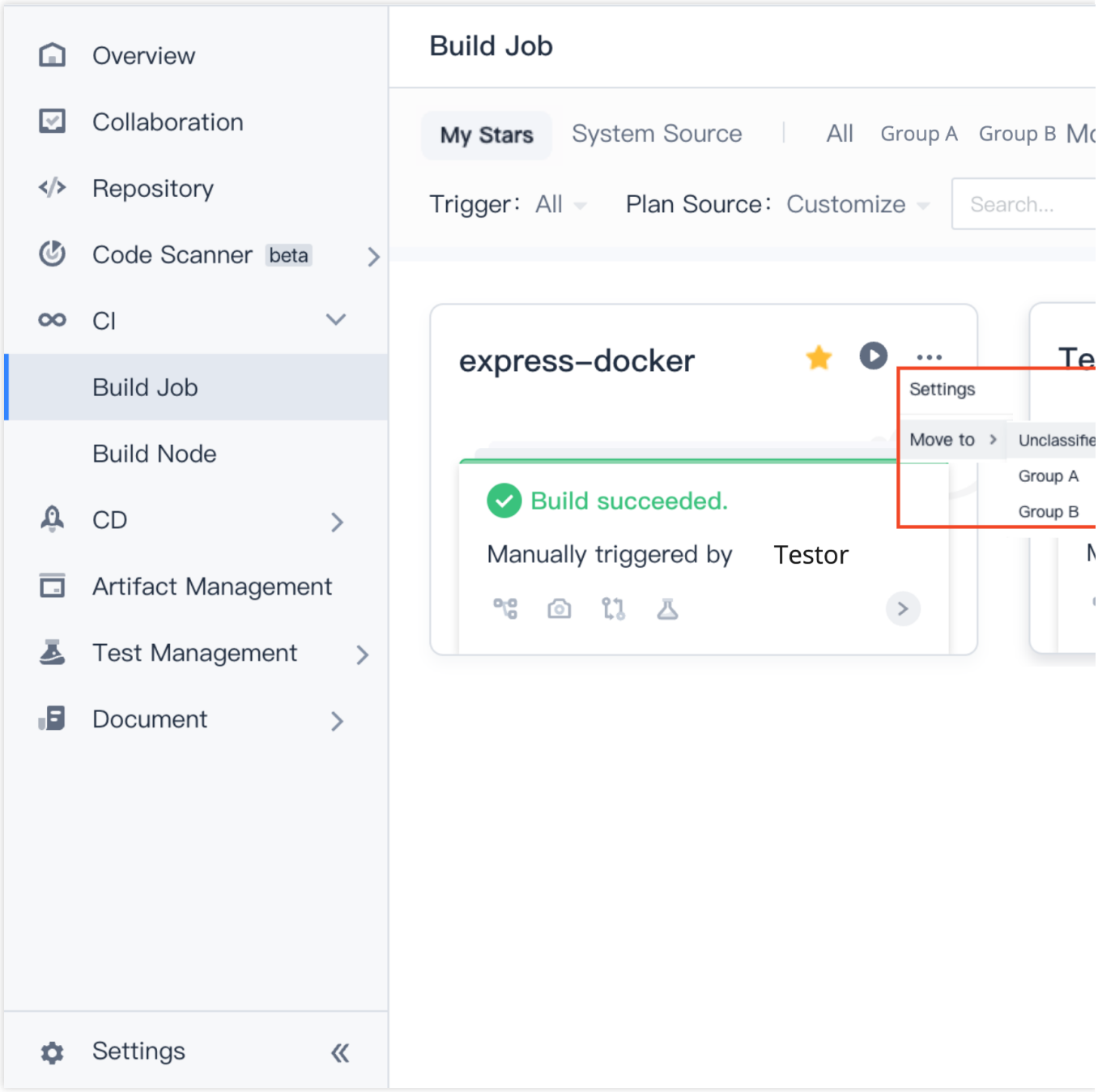


还可以修改分组名称、排序，也可以创建和删除分组。

注意：删除分组不会删除分组中的构建计划，分组删除后，分组内的构建计划将会被归类到“未分组”类别。

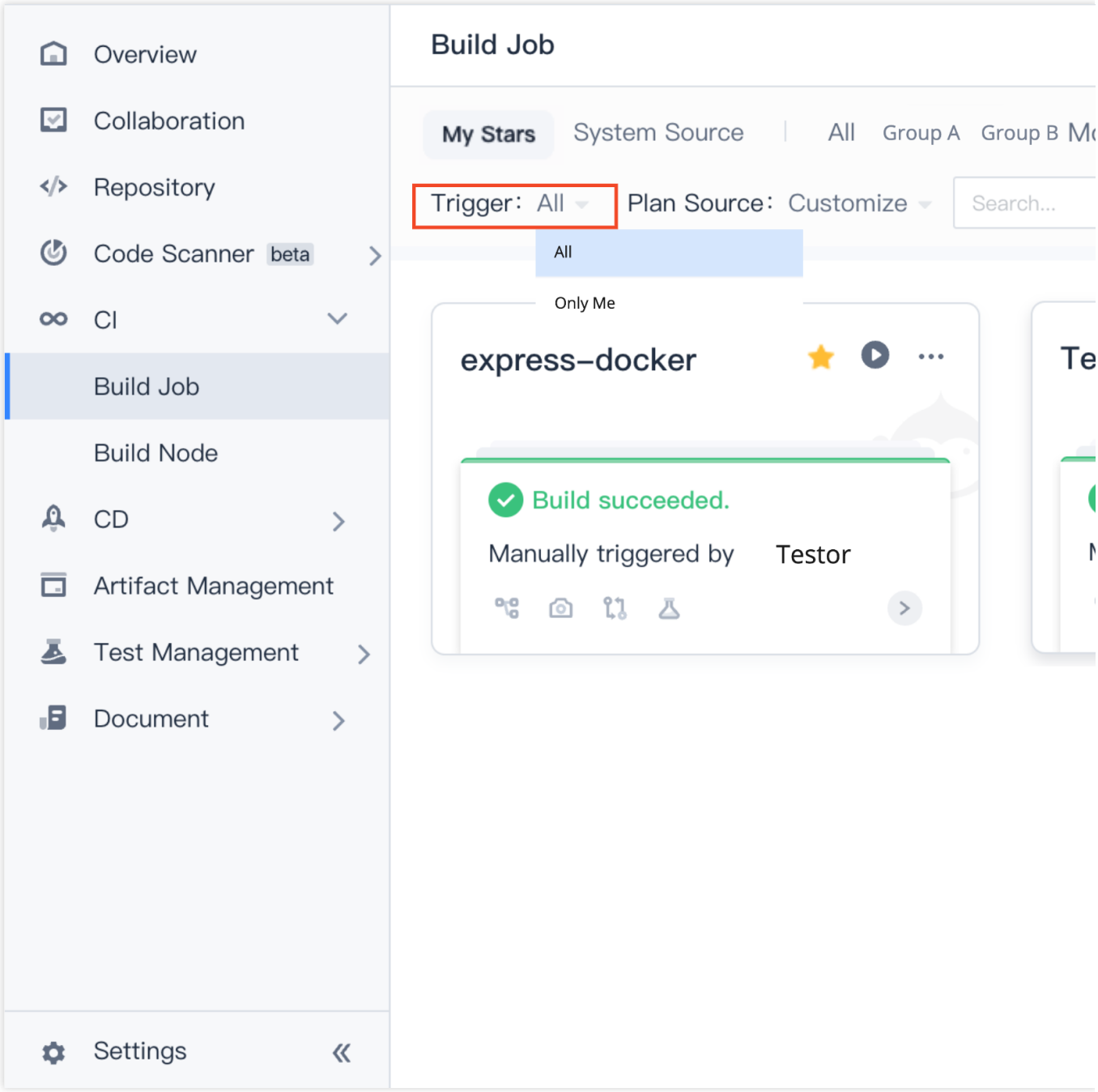


点击【批量整理构建计划】可以进入构建计划整理页面，可以一次性选择多个构建计划至同一个分组当中。添加完成后就可以在单独的分组 tab 页中看到勾选的构建计划。



筛选与排序

在构建计划页面右侧的搜索栏中可以根据构建计划名称进行筛选。选择【筛选器】->【只看我触发的】，构建计划中将会显示由用户本人最新一次触发的构建记录，并且在构建记录展示页，也可以开启此筛选按钮。



除此之外，还可以按照构建计划最新构建记录的触发时间排序。

==== 2020/09/25 ====

构建计划模板

最近更新时间：2023-12-29 11:44:51

title: 构建计划模板 - CODING 帮助中心
pageTitle: 构建计划模板
pagePrevTitle: 分组管理
pagePrev: ci/manage/group.html
pageNextTitle: 上传 Generic 制品
pageNext: ci/plugins/generic.html
alias:
ci/node/overview.html
ci/team-template.html

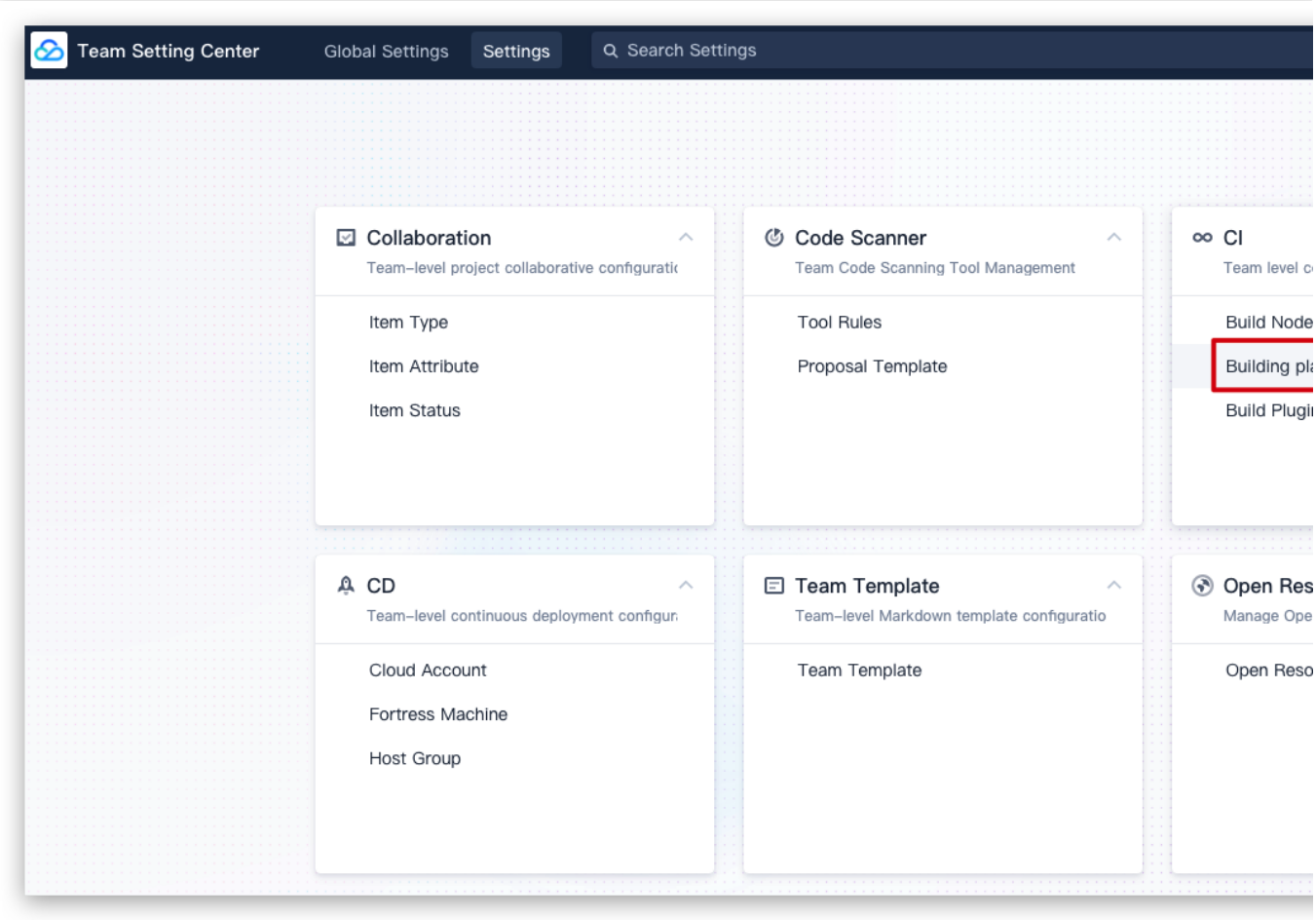
功能介绍

CODING 持续集成支持编制统一的**构建计划模板**。一次配置，就能在团队内的跨项目协作中复用统一而规范的构建计划模板，优化团队成员配置构建流程的效率，集中管理团队中通用的构建计划。

新建构建计划模板

点击团队首页右上角的齿轮图标

进入团队设置中心，你可以在「功能设置」→「构建计划模板」中新建团队构建模板。

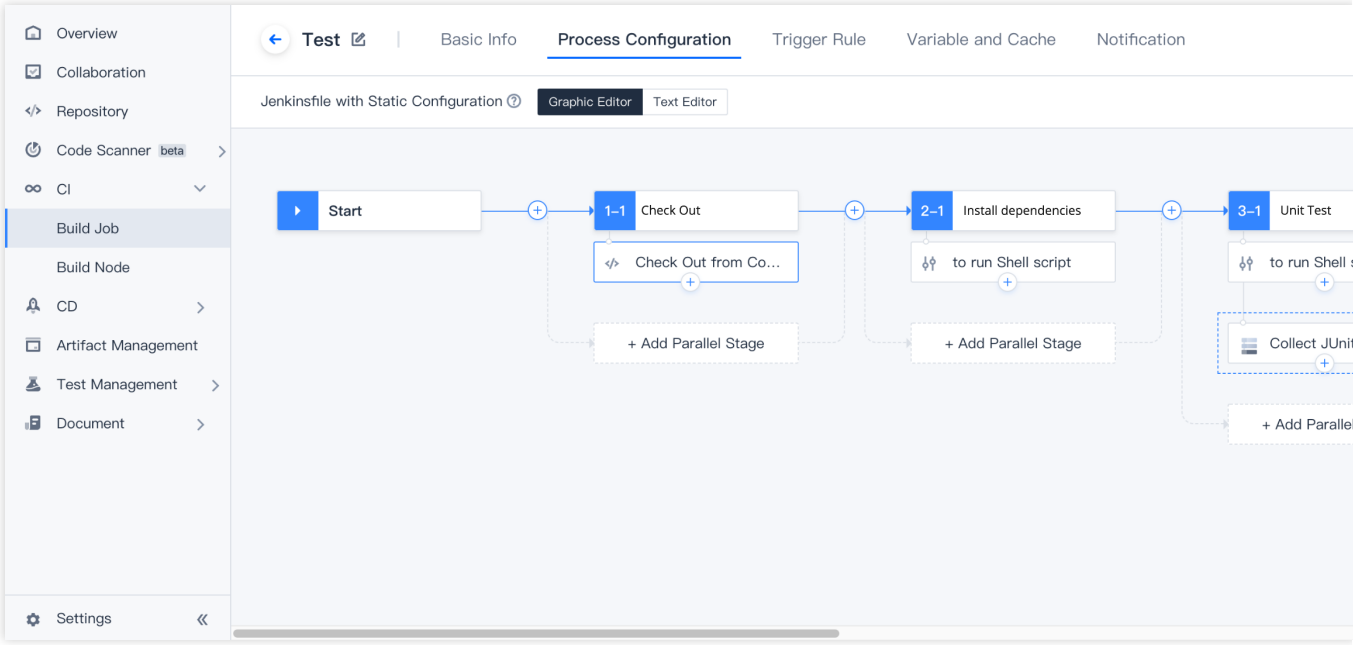


编辑构建计划模板

在模板内能够编辑流程配置、基础配置、触发规则以及变量与缓存。

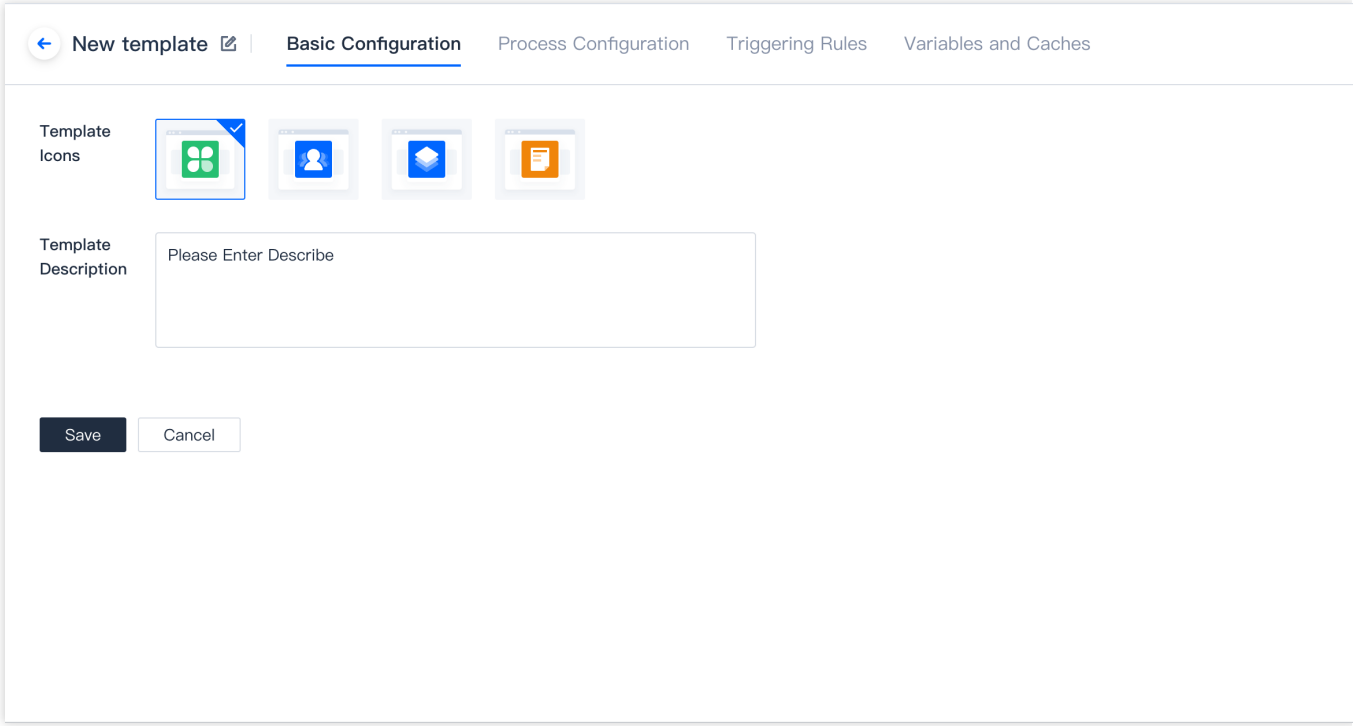
流程配置

您可以使用「图形化编辑器」或「文本编辑器」编写该构建模板的执行流程。图形化编辑器的优势在于能够获得边看边写的可视化体验，在图形化编辑器中增删的所有步骤都可以转换成文本，但反之则不一定能够兼容。



基础配置

在基础配置页能够更改模板名称、图标。点击右侧的操作下拉菜单能够执行删除模板或同步模板。



当模板有更新时，模板作者可以通过「同步模板」操作将更新同步至所有使用该模板创建的构建计划。该操作将覆盖对应构建计划的配置，点击查看场景举例。

触发规则

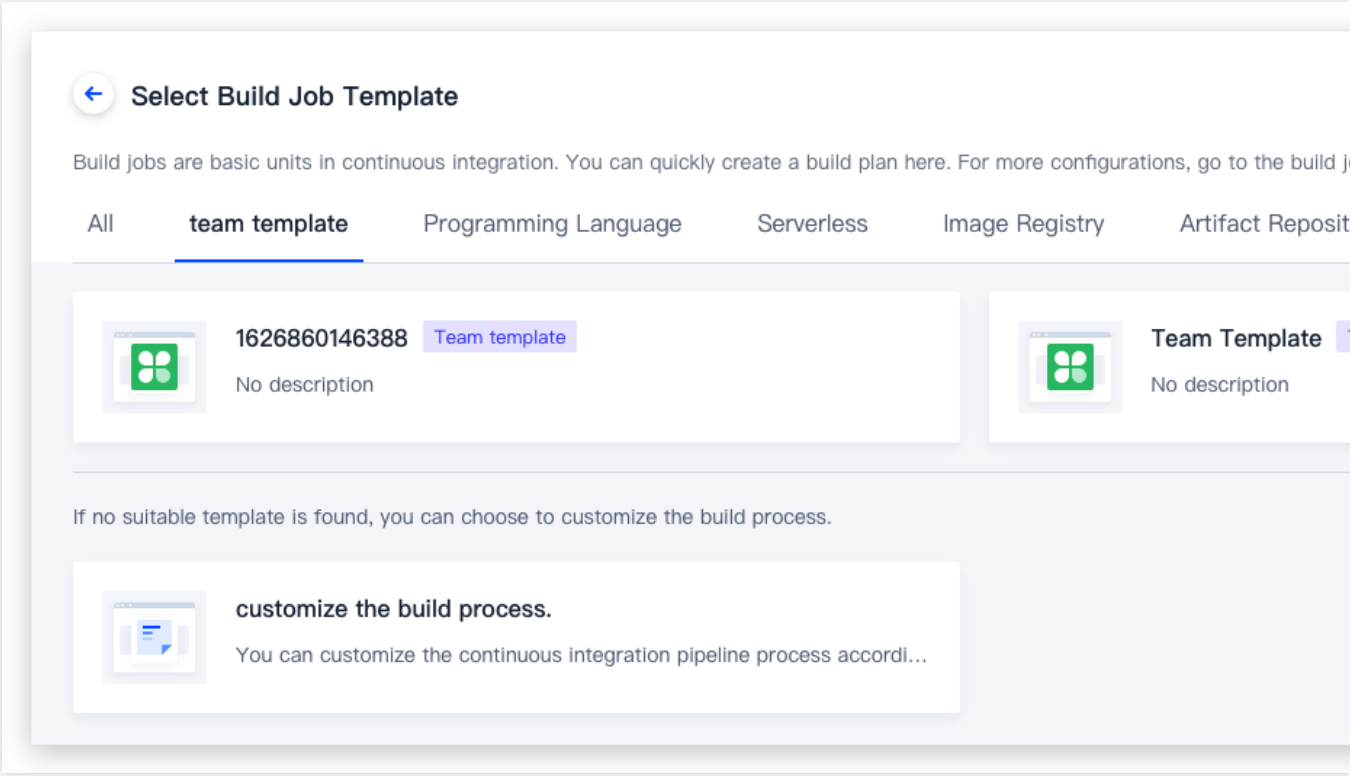
支持代码源触发、定时触发及手动触发。与普通构建计划下的触发规则设置方式一致，具体说明请参考《触发规则》。

变量与缓存

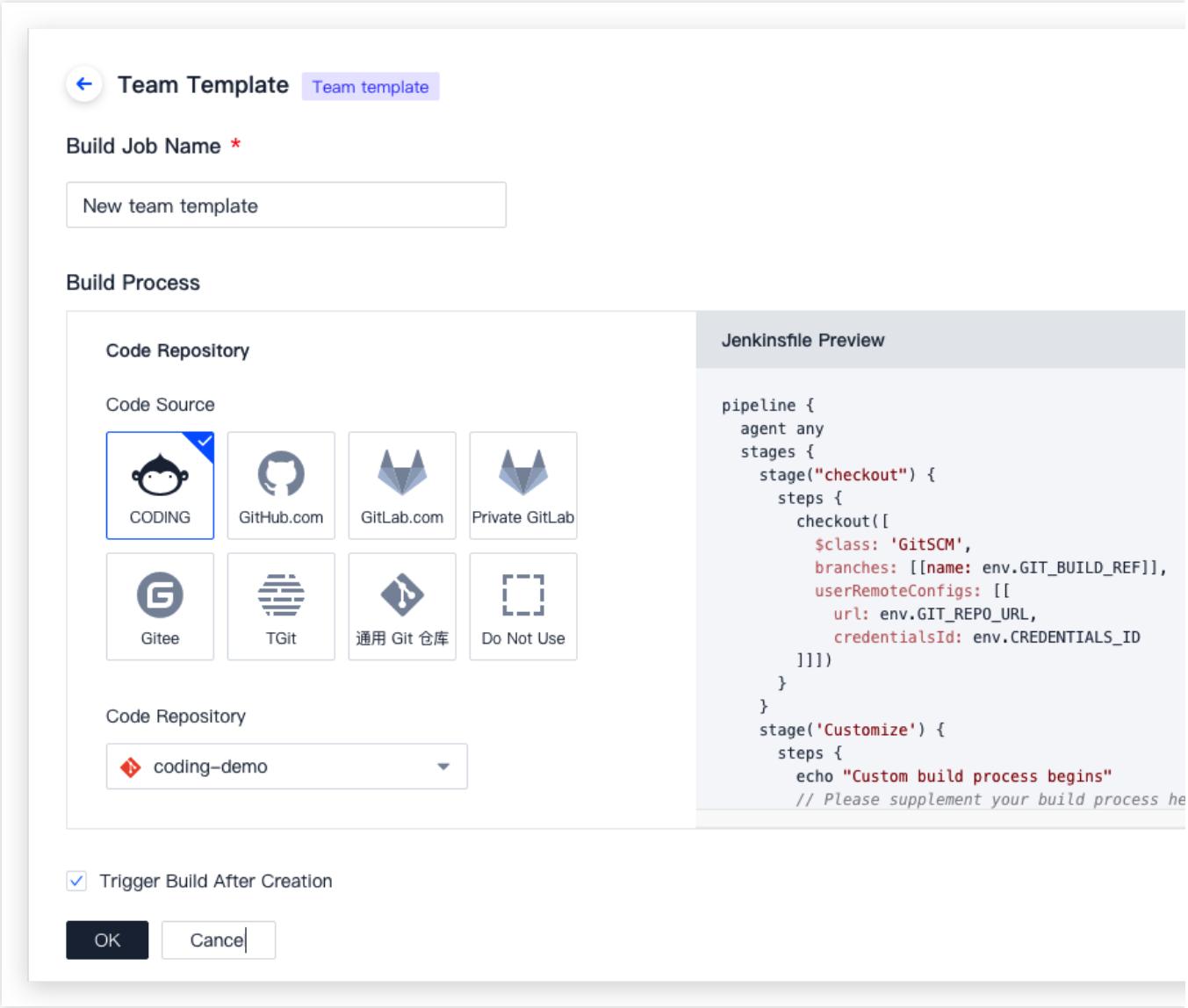
您可以在此处添加构建计划的环境变量。在手动启动构建任务时，环境变量也将作为启动参数的默认值，具体配置说明请参考《环境变量》。

使用构建计划模板

模板作者完成团队构建模板的编写后，其他团队成员就可以在任一项目中使用该构建计划模板。



构建计划左上角处会标注为构建计划模板，使用者可以按照项目需求选择不同的代码源。

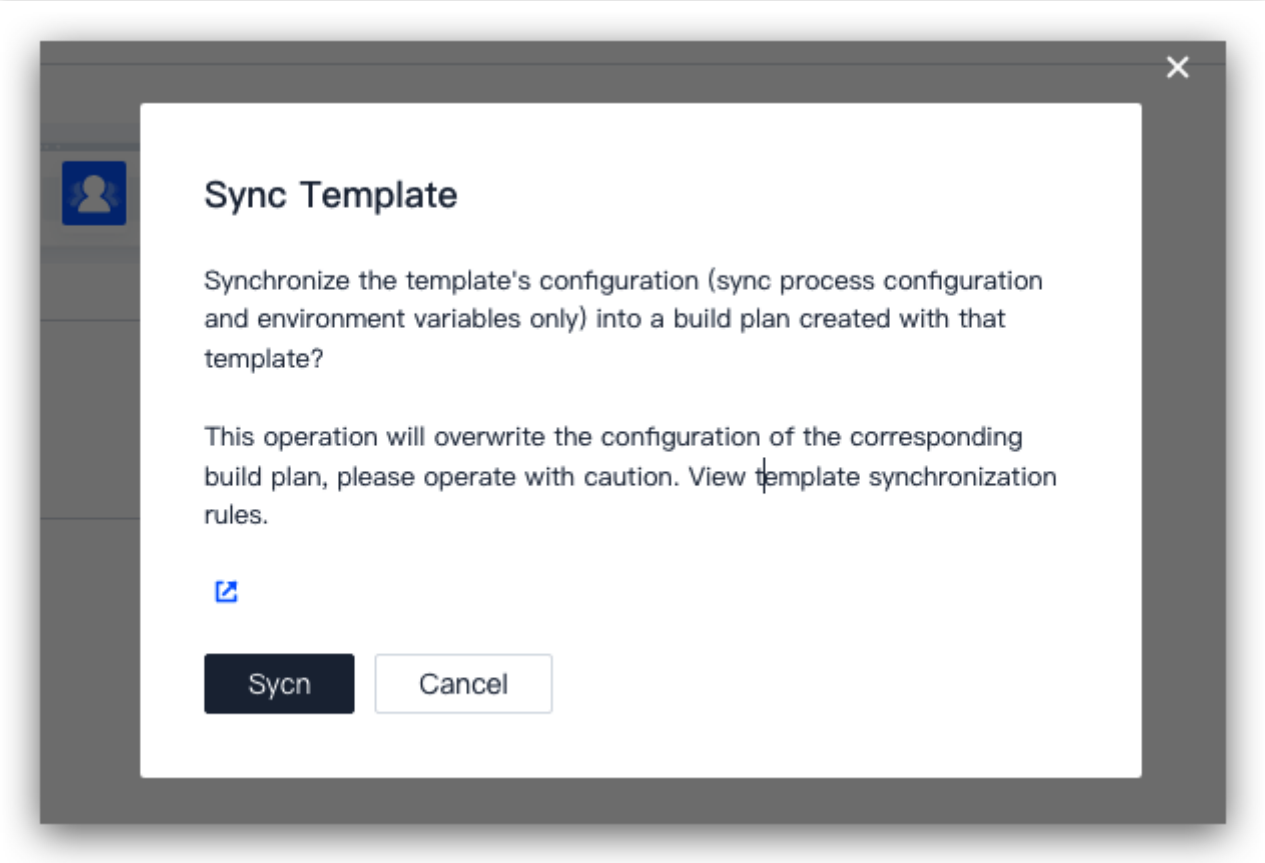


创建成功后，构建计划流程、触发配置、环境变量及默认值与模版保持一致，使用者可以按照项目的实际情况进行修改。基于模板做出的不会影响模板内容，如需修改模板请前往设置中心的【功能设置】>【构建计划模板】处进行更新。

同步构建计划模板

当团队内部使用了某项构建计划模板作为其他构建计划的基石时，修改构建计划模板后使用同步功能，就能够让其他构建计划向构建计划模板对齐。

使用场景举例：A 团队已在内部全面推行持续集成构建规范，大部分构建计划皆是基于某一规范性构建计划模板进行编写。随着项目推进，旧有规范亟待更新，此时模板作者仅需完成构建计划模板的更新并使用同步功能，就能让其他构建计划与模板进行对齐。



同步功能并不会覆盖其他构建计划中的所有内容，下图为同步所导致的「变更」效果图：

⚠ 使用同步功能前请确保已知晓该操作可能会造成的影响。

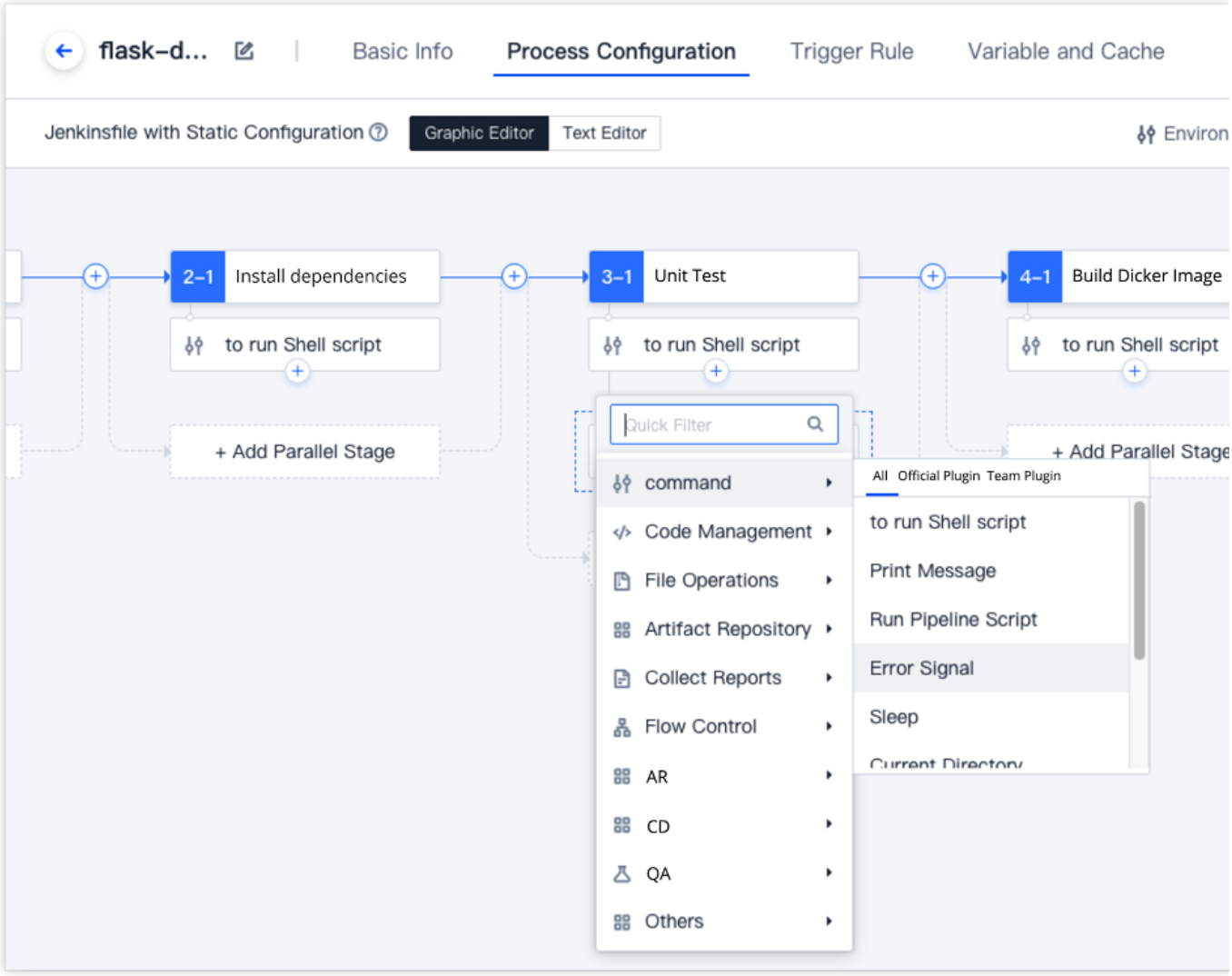
==== 2020/11/27 ====

系统插件
错误信号

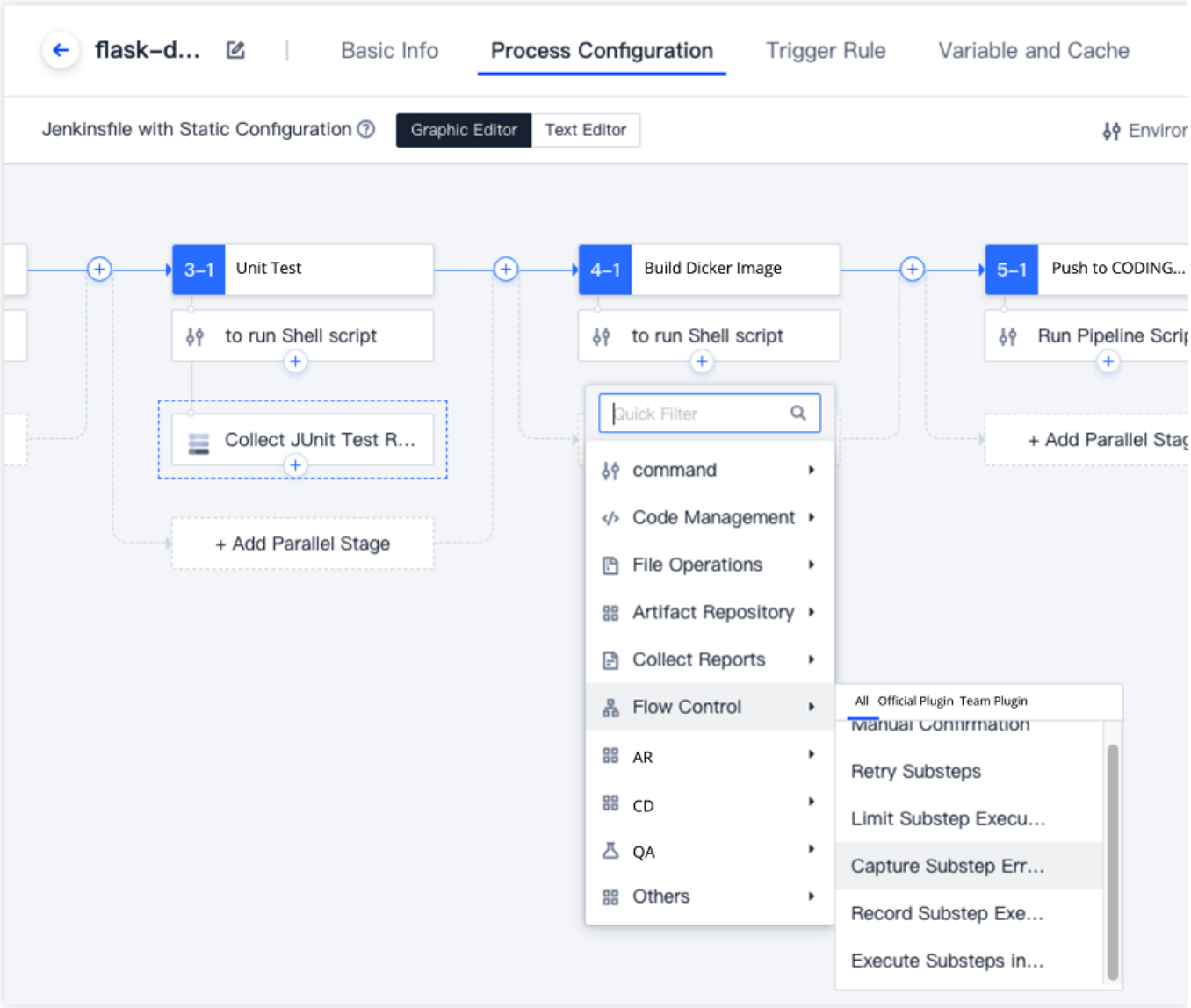
最近更新时间：2023-12-29 11:44:51

title: 错误信号 - CODING 帮助中心
pageTitle: 错误信号
pagePrevTitle: 将镜像更新至 K8s 集群
pagePrev: ci/plugins/html-report.html
pageNextTitle: 推送到 CODING Docker 制品仓库
pageNext: ci/plugins/cci-push-docker.html
alias: devops/ci/plugins/api-doc.html

持续集成的「错误信号」步骤可以理解为构建的终止符，运行至此步骤后将停止余下步骤，直接中断构建过程。



在持续集成中添加「捕获错误子步骤」，能够将运行结果作为是否中断持续集成任务的信号。若成功运行，则继续执行余下步骤；即使失败也将执行余下步骤，但构建任务会被判断为失败。


flask-d...
Basic Info
Process Configuration
Trigger Rule
Variable and Cache
Jenkinsfile with Static Configuration
Graphic Editor
Text Editor
Enviror
3-1
Unit Test
to run Shell script
Collect JUnit Test R...
+ Add Parallel Stage
4-1
Build Dicker Image
to run Shell script
Quick Filter
command
Code Management
File Operations
Artifact Repository
Collect Reports
Flow Control
AR
CD
QA
Others
All
Official Plugin
Team Plugin
Manual Confirmation
Retry Substeps
Limit Substep Execu...
Capture Substep Err...
Record Substep Exe...
Execute Substeps in...
5-1
Push to CODING...
Run Pipeline Scrip
+ Add Parallel Stag

==== 2021/06/30 ====

上传 Generic 类型制品

最近更新时间：2023-12-29 11:44:51

title: 上传 Generic 类型制品 - CODING 帮助中心
pageTitle: 上传 Generic 类型制品
pagePrevTitle: 团队模板
pagePrev: ci/manage/team-template.html
pageNextTitle: 调取已录入的凭据
pageNext: ci/plugins/credentials.html
alias: devops/ci/plugins/generic.html

功能介绍

在实际的生产环境中，有着许多重复性较强的工作。CODING 持续集成具备插件功能，它能够帮助您高效处理繁琐重复的工作，并且还可以通过自定义参数来解决个性化需求。CODING 即将支持更多的内置插件。目前 CODING 的持续集成已支持以下四种便捷的插件；
上传 Generic 类型制品
调取已上传的凭据
在合并请求中自动添加评审者
人工确认

使用插件上传 Generic 类型制品

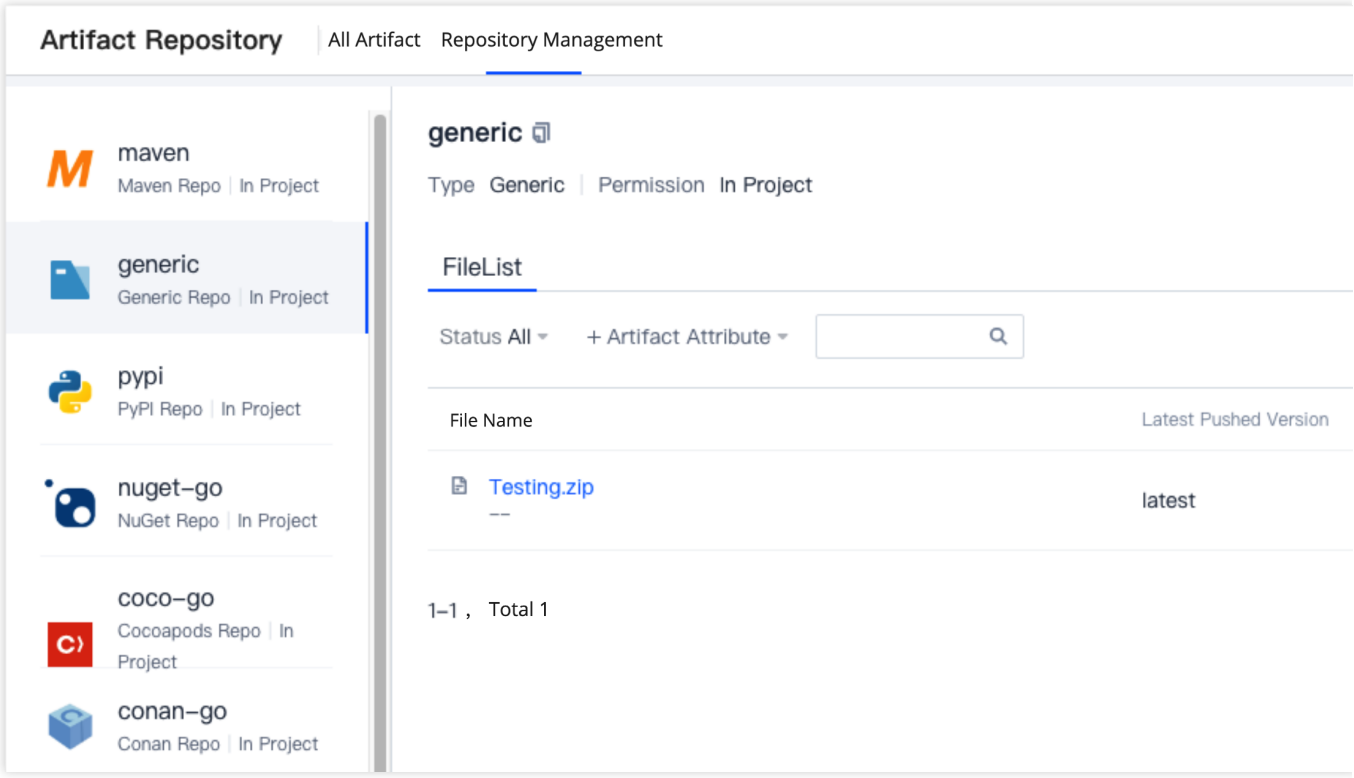
在 CODING 持续集成任务构建过程当中，您可以选择将构建物上传至 CODING 制品库。Generic 制品上传插件能够让您更方便地在持续集成当中上传 Generic 类型构建物，目前单文件大小最高支持 5GB。在使用该功能之前，请确保您对 Generic 类型制品库有初步了解，您可以点击阅读使用 Generic 制品库了解更多。

快速开始

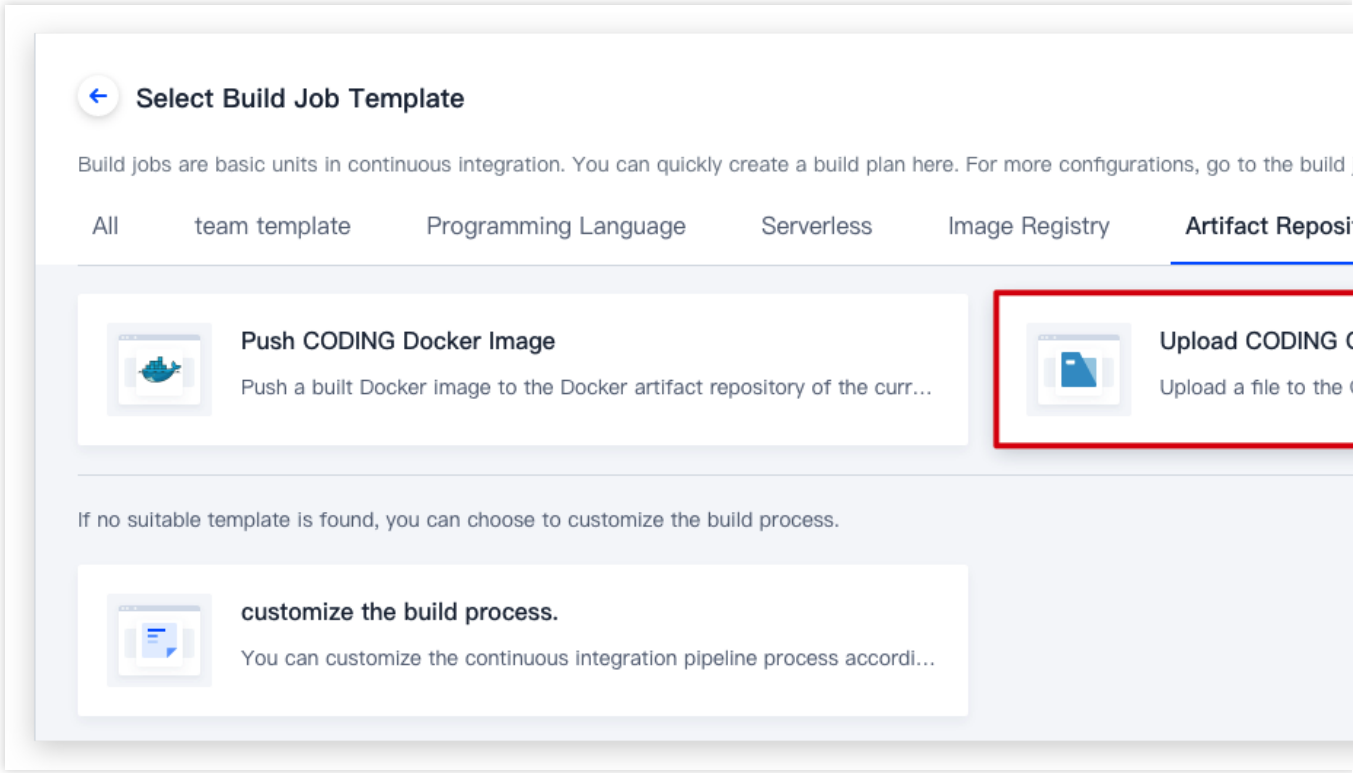
您可以选择使用固定模板或 Jenkinsfile 配置两种方式来上传 Generic 类型制品。

固定模板

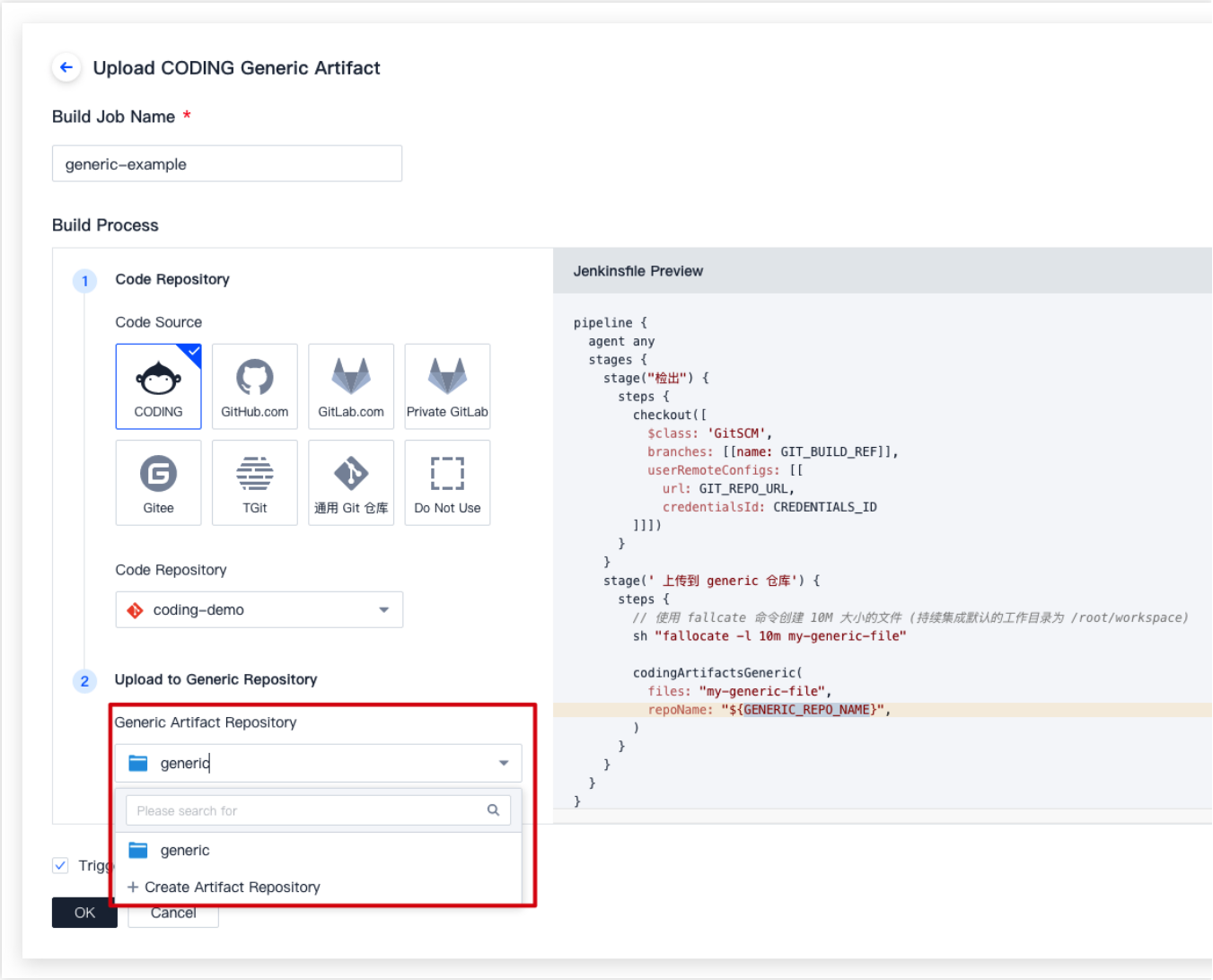
1. 在使用插件上传前，请确保您已经创建了 Generic 类型制品库，下方的演示将会以 test 制品库为例。



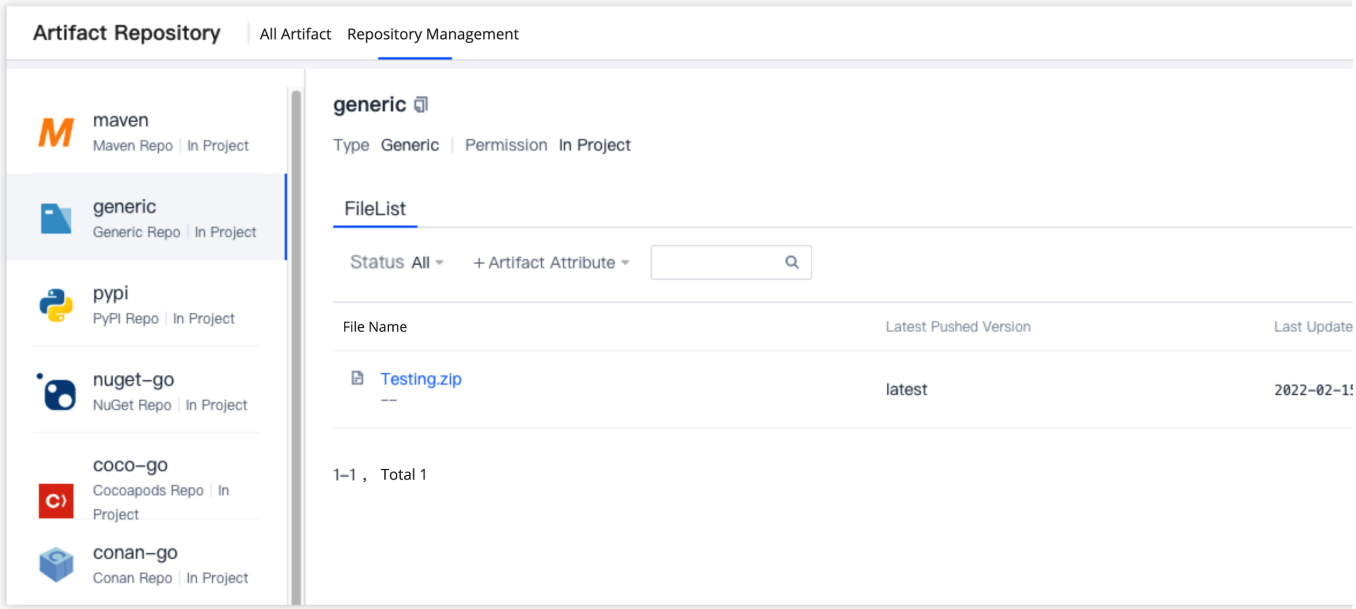
2. 点击新建构建计划后，选择【制品库】中的 CODING Generic 制品上传。



3. 在默认值选中刚刚创立的 test 制品库，您也可以在代码中填写您的制品库地址。

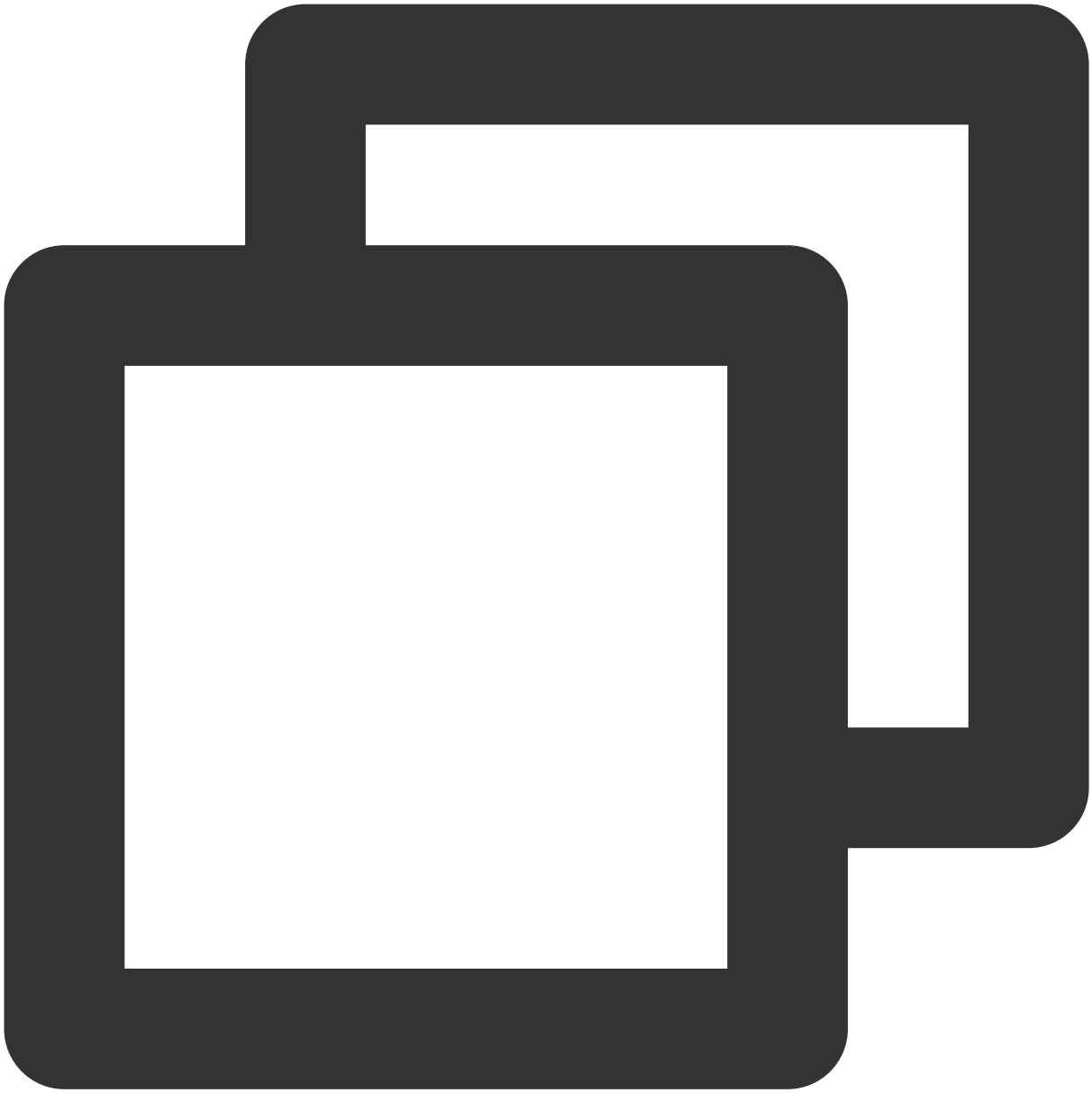


4. 勾选创建后触发构建，点击确定后完成构建。
5. 构建完成后在制品库中会出现刚刚上传的文件。



Jenkinsfile 配置

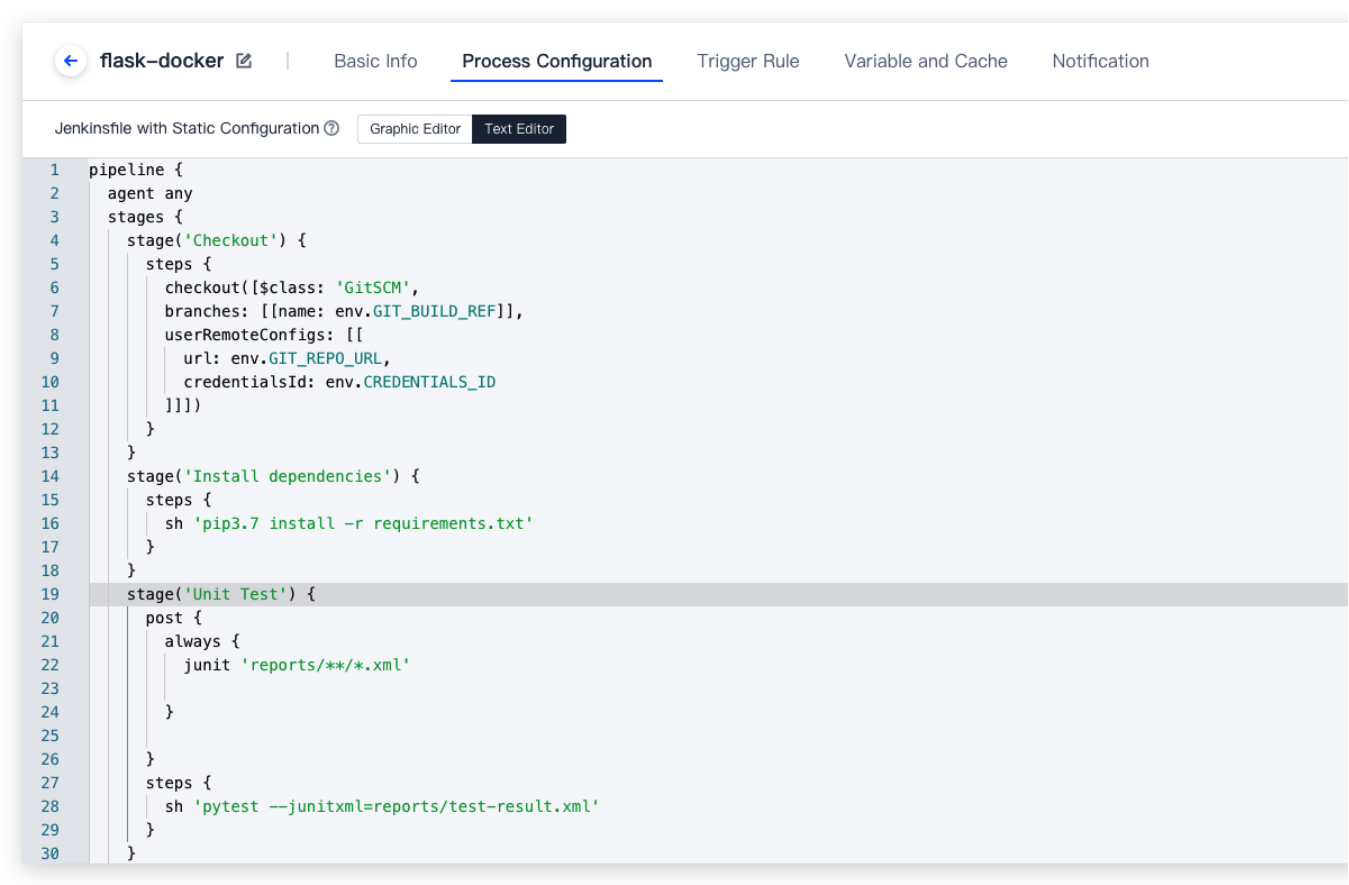
1. 在选择代码仓库的时候，请确保代码仓库中已含有如下配置的 Jenkinsfile 文件。



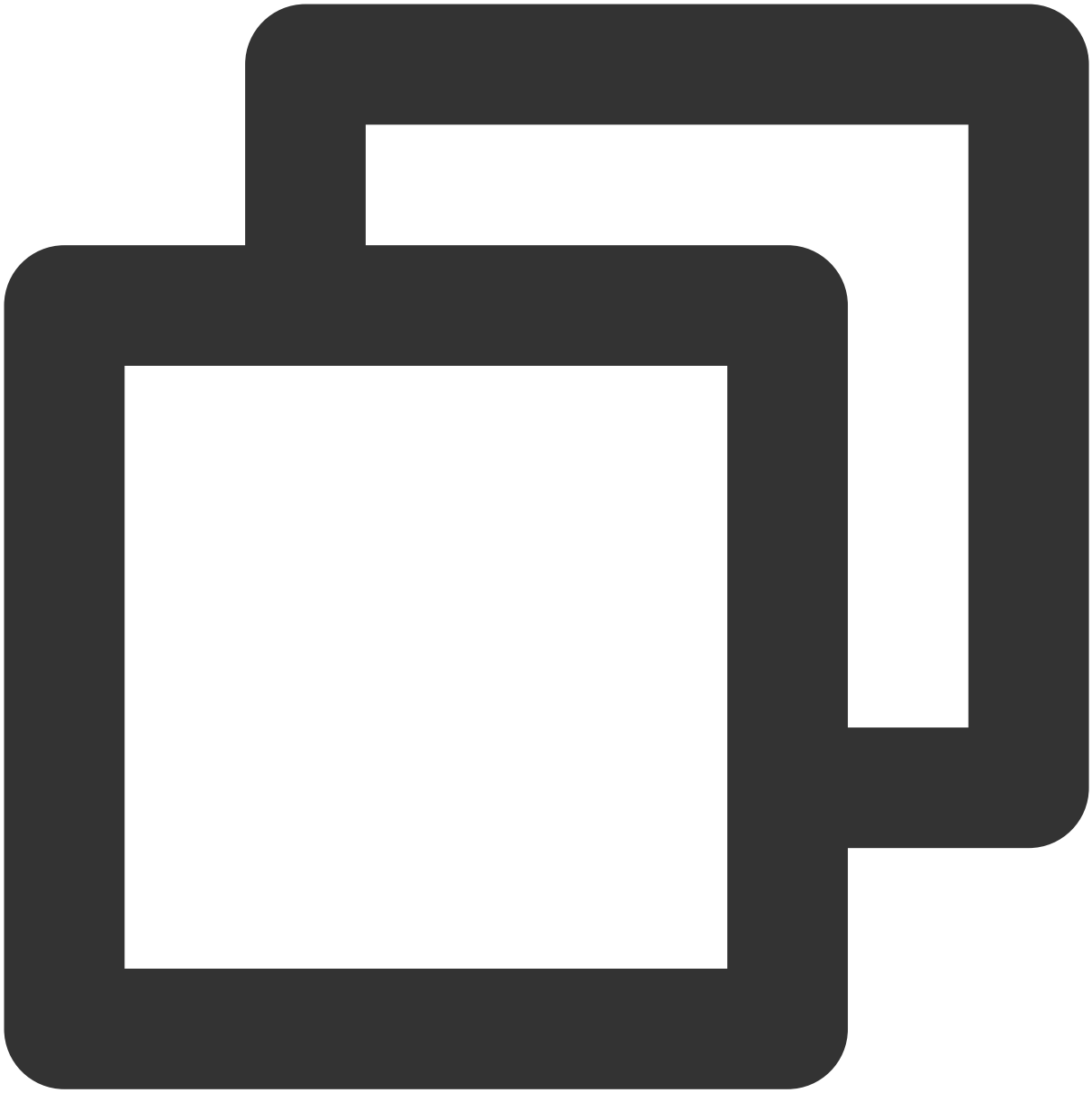
```
pipeline {
    agent any
    stages {
        stage('上传到 generic 仓库') {
            steps {
                // 使用 fallcate 命令创建 10M 大小的文件 (持续集成默认的工作目录为 /root/wo
                sh 'fallocate -l 10m my-generic-file'

                codingArtifactsGeneric(
                    files: 'my-generic-file',
                    repoName: 'myrepo', //此处填写您的仓库参数，例如${env.GENERIC_REPO_I
```

```
                    )
                }
            }
        }
    }
```

2. 在流程配置中也可以对 Jenkinsfile 配置进行修改。



更多参数演示



```
pipeline {
    agent any
    stages {
        stage('上传到 generic 仓库') {
            steps {
                // 使用 fallcate 命令创建 10M 大小的文件 (持续集成默认的工作目录为 /root/wo
                sh 'fallocate -l 10m my-generic-file'

                codingArtifactsGeneric(
                    files: 'my-generic-file',
                    repoName: 'myrepo',
```

```
                    )
                }
            }
        }
    }
```

参数说明

参数名称	必填	参数类型	图形化参数类型	默认值
files	是	-	需要上传的文件列表，支持通配符 `build/libs/**/xx`	-
repoName	是（若单独设置了repoURL，则可以不填）	string	制品库名称	-
version	否	string	string	latest
zip	否	-	string	string
credentialsId	否	string	凭据（用户名+密码）	env.CODING_ARTIFACTS_CREDENT
repoURL	否	string	string	https:// `<` env.CCI_CURRENT_TEAM> generic. `<` env.CCI_CURRENT_DOM/
withBuildProps	否	boolean	boolean	true
workspace	否	string	否	-

==== 2020/08/24 ====