持续集成

最佳实践

产品文档

腾讯云

文档目录

最佳实践
自动化部署
K8s 集群

最近更新时间：2023-12-29 11:44:51

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。

进入项目

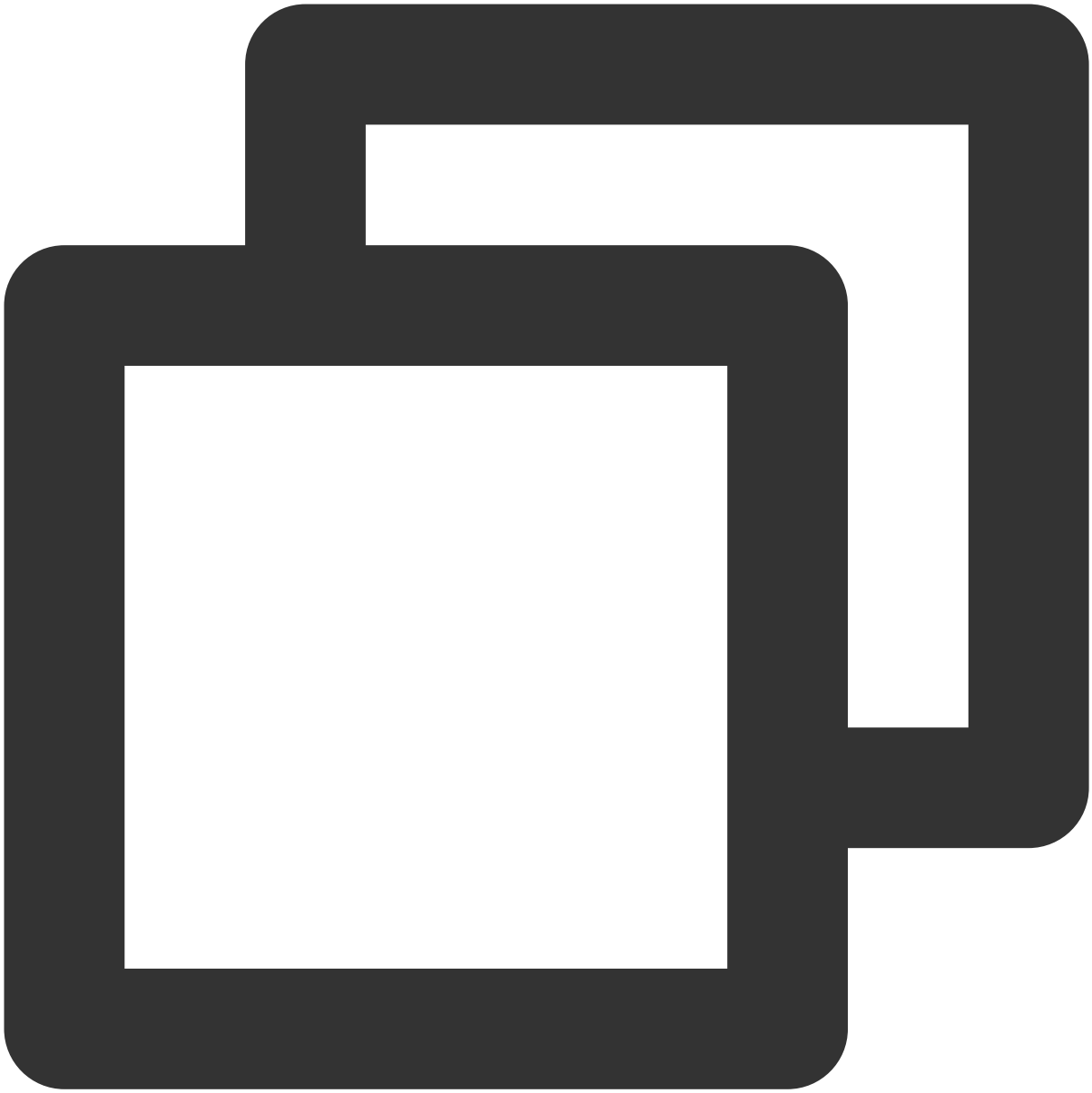
1. 登录 CODING 控制台，单击**团队域名**进入 CODING 使用页面。
2. 单击页面右上角的

，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_
      }
    }
    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_CREDENTIA
            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-dfd1-40b9-a12f-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-serve

              // 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\\": {\\"tem
            }
          }
        }
      }
    }
  }
```

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。

Docker 服务器

最近更新时间：2023-12-29 11:44:51

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。

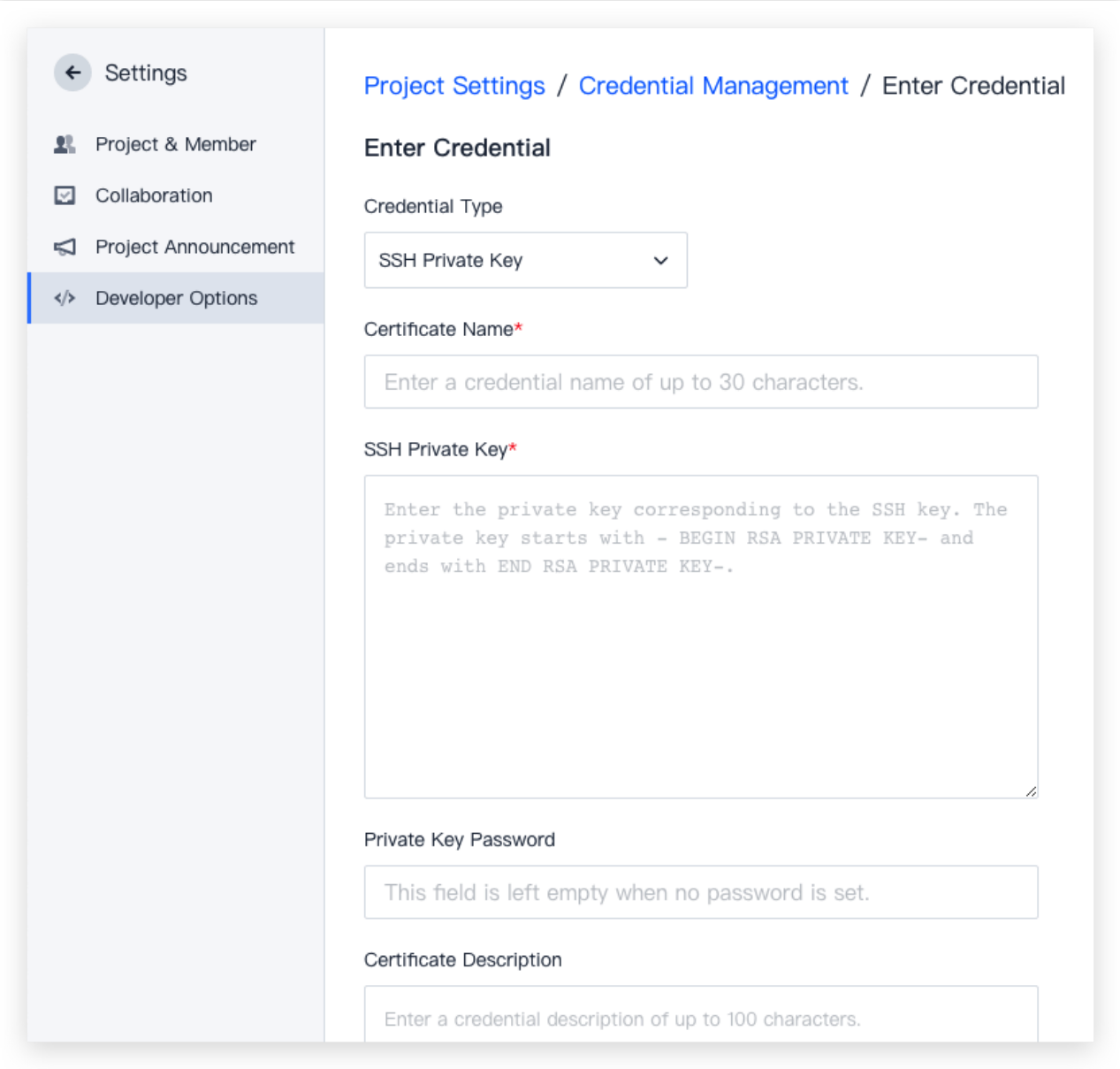
进入项目

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。

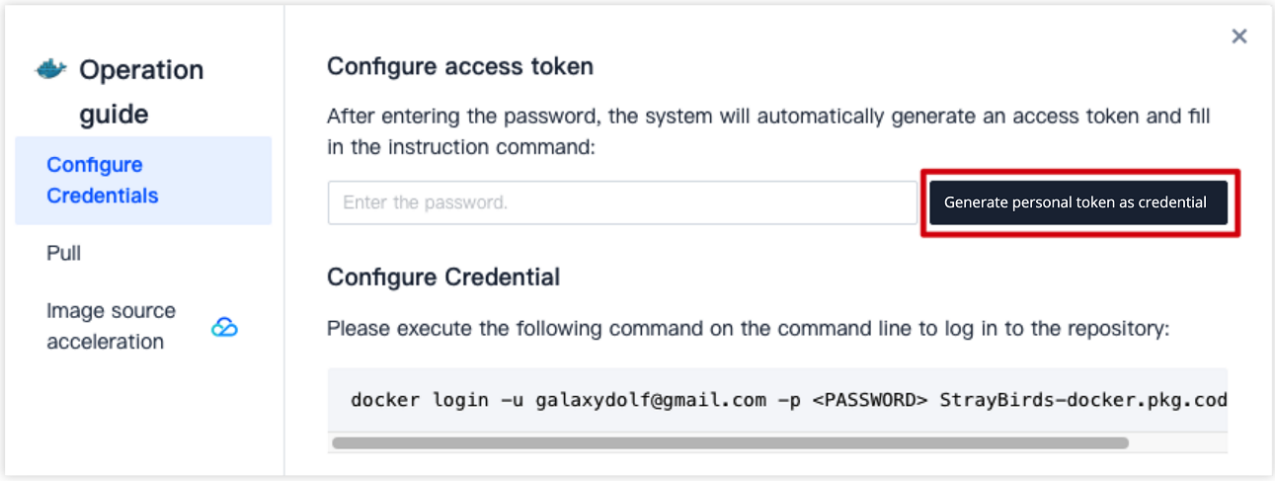
获取 SSH 密钥对

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 中的项目令牌中，将公钥 id_rsa.pub 的内容复制到服务器的 `~/.ssh/authorized_keys` 中。

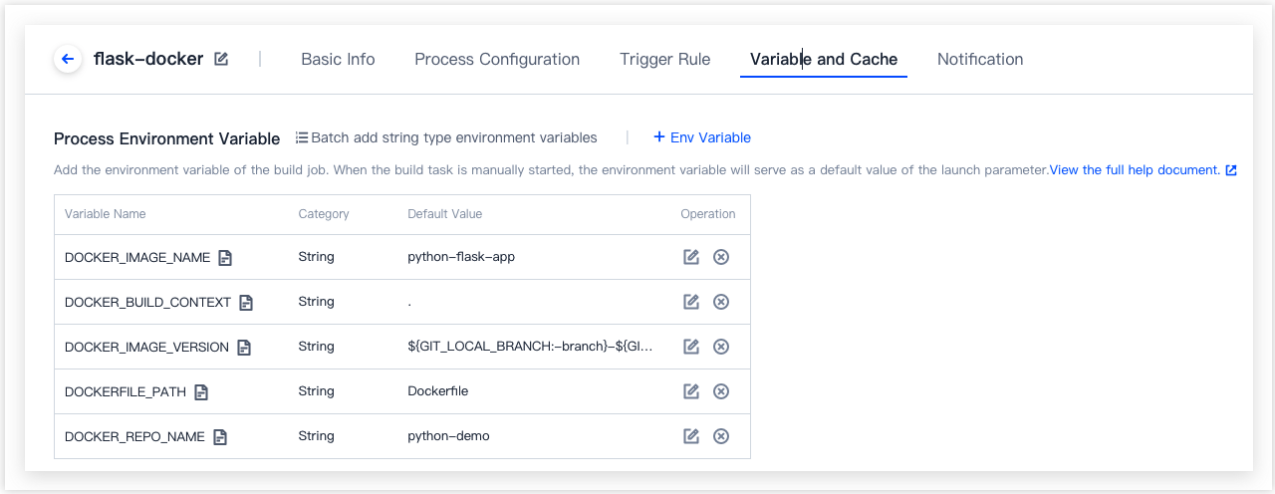


获取制品仓库信息

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。

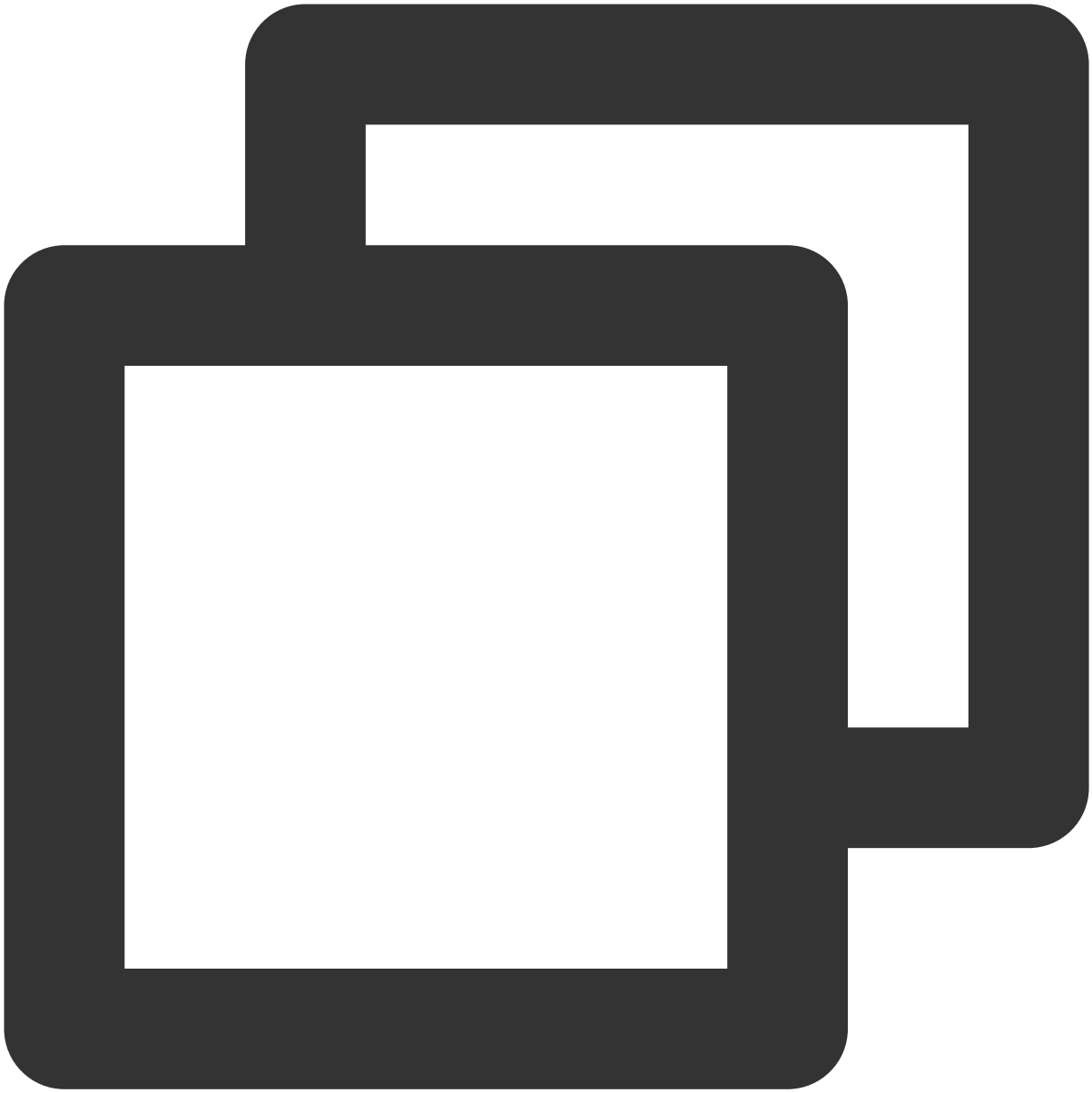


2. 在构建计划详情中的变量与缓存处填写。



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。

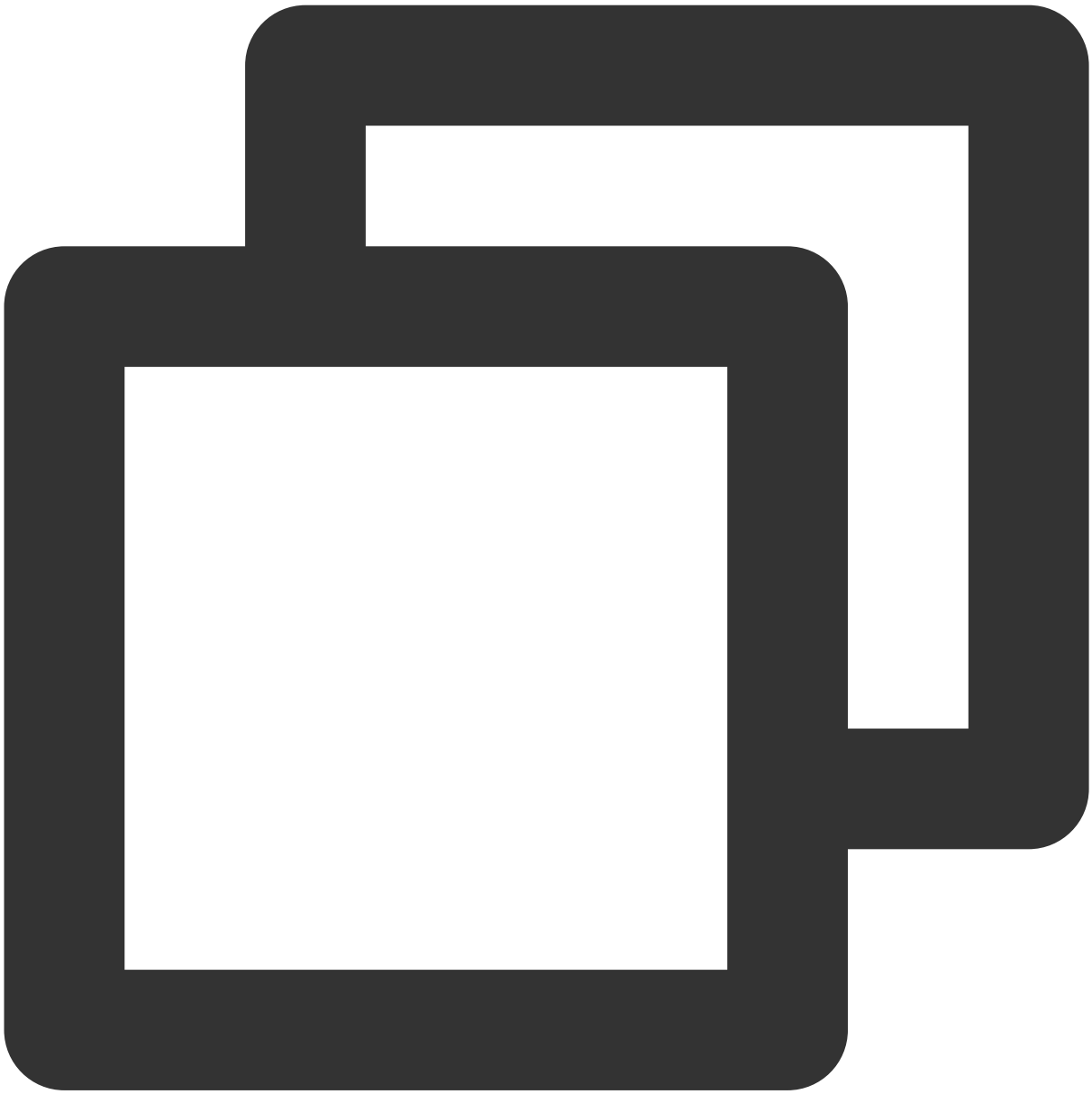


```
pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
        checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
        userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_
      }
    }
    stage('构建') {
      steps {
```

```
        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_CREDENTIA
            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 = '106.54.86.239'
            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-402a-4f38-9c8
              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 gnu
              sshCommand remote: remote, command: "docker login -u ${env.DOCKER_USER}
              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 ${imageNa
            }
          }
        }
      }
    }
}
```

Docker Compose

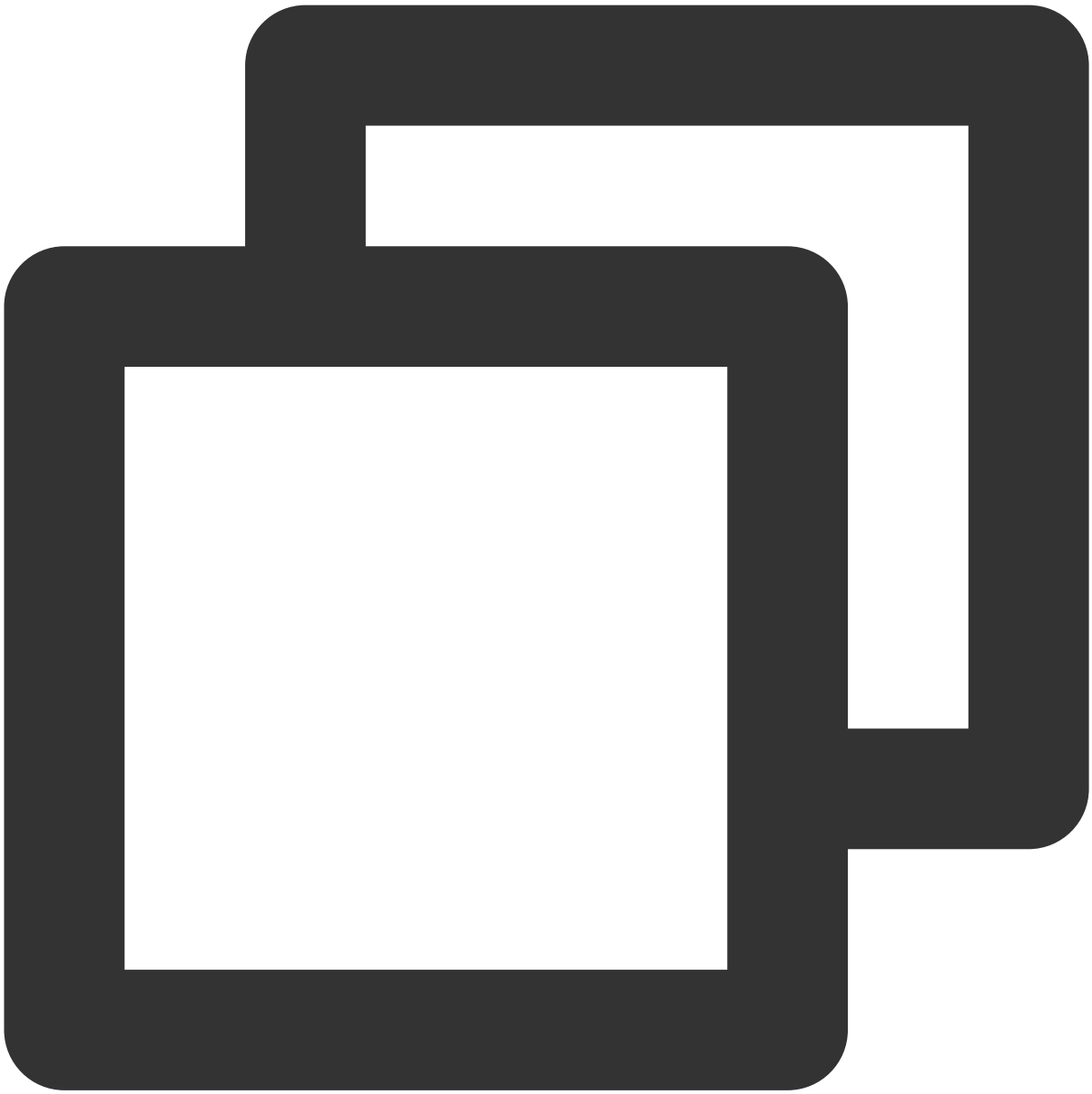
Docker Compose 的代码和上述类似，仅有少许不同：



```
sshCommand remote: remote, sudo: true, command: "apt-get install -y gnupg2 pass"
sshCommand remote: remote, command: "docker login -u ${env.DOCKER_USER} -p ${env.DO
sshCommand remote: remote, sudo: true, command: "mkdir -p /var/www/site/"
sshCommand remote: remote, sudo: true, command: "chmod 777 /var/www/site/"
sshPut remote: remote, from: 'docker-compose.yml', into: '/var/www/site/'
sshCommand remote: remote, command: "cd /var/www/site/ && echo IMAGE=${imageName} >
sshCommand remote: remote, command: "cd /var/www/site/ && docker-compose down --rem
```

```
sshCommand remote: remote, command: "cd /var/www/site/ && docker-compose up -d --no
```

`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"
```

COS 存储桶

最近更新时间：2023-12-29 11:44:51

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

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。

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

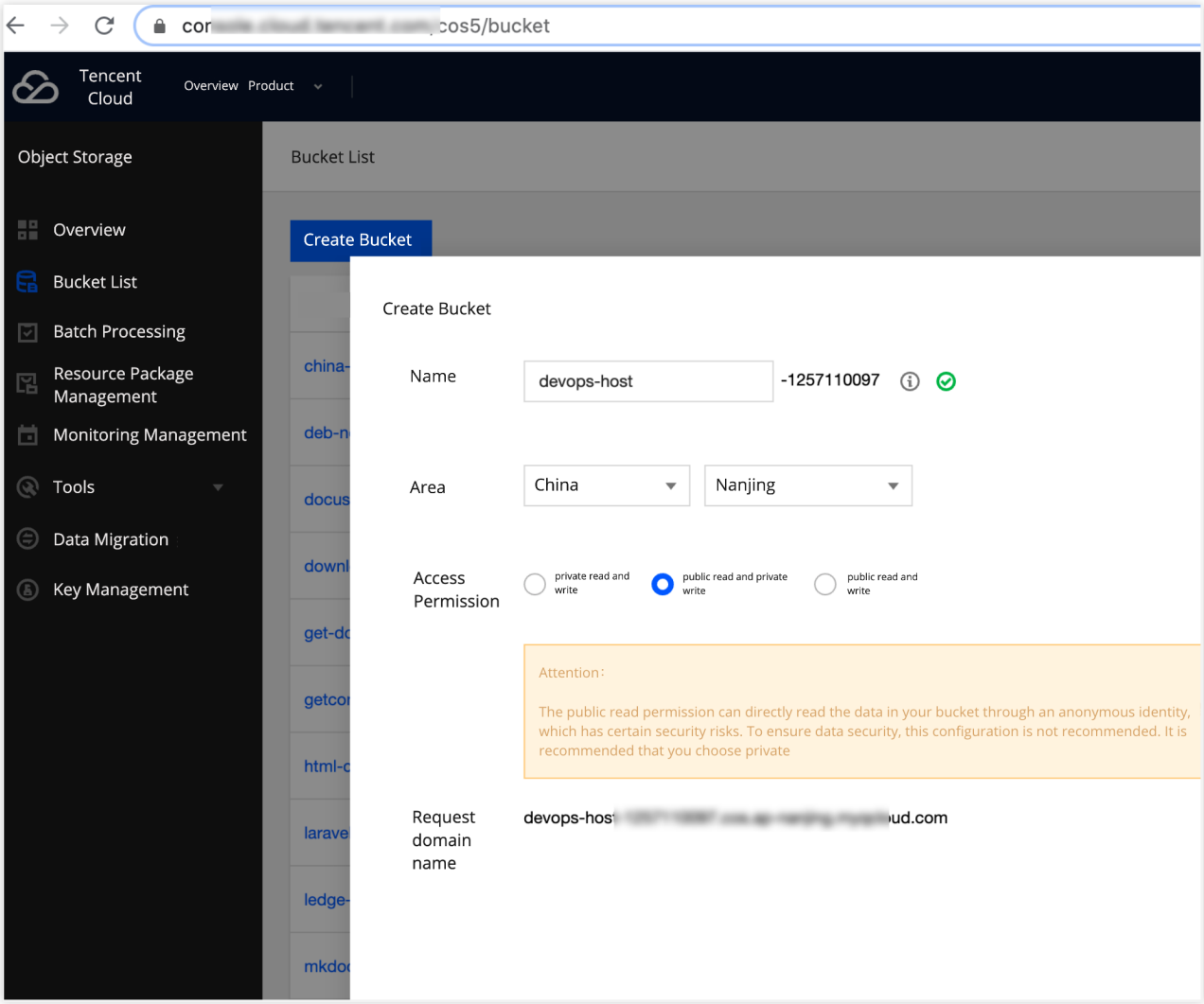
，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

功能介绍

得益于腾讯云存储的自动扩容功能，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。

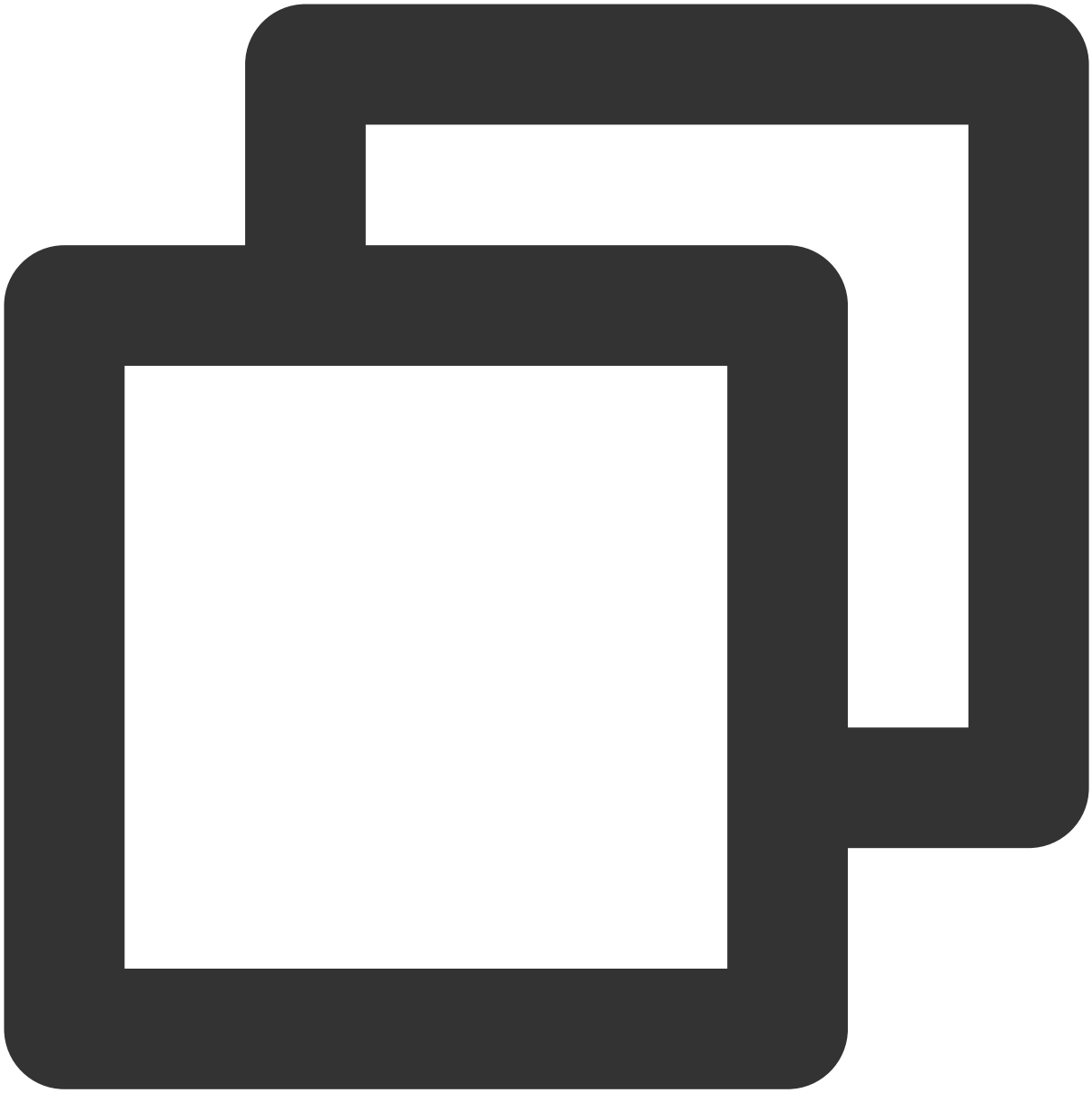
新建存储桶

在云存储（例如 腾讯云 COS 对象存储）中创建一个**存储桶**，获取名称、区域、密钥。



Jenkinsfile

在持续集成中参考并写入下述 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_
      }
    }
    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

在持续集成中使用 Docker

最近更新时间：2023-12-29 11:44:51

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。

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

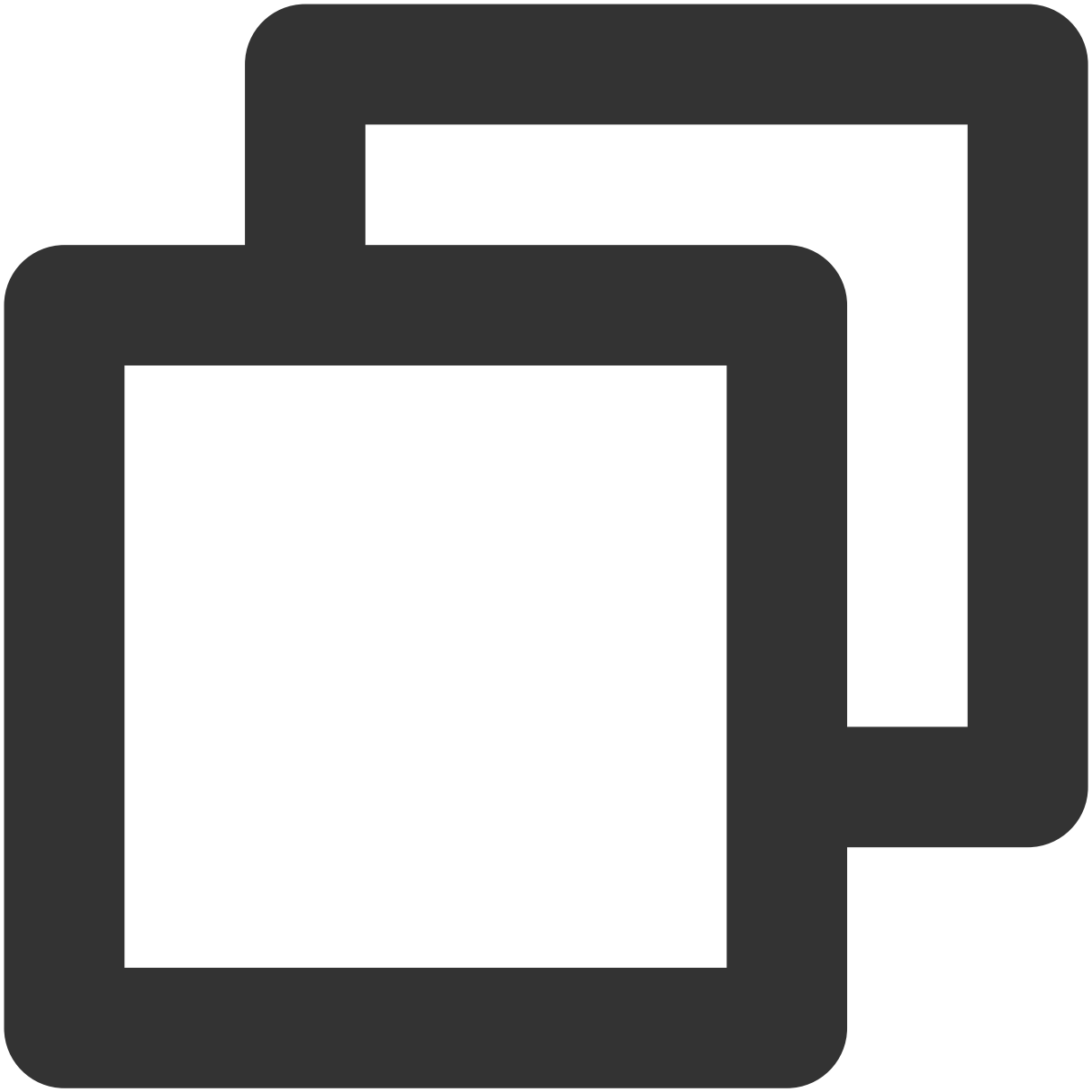
，进入项目列表页面，单击**项目图标**进入目标项目。
3. 进入左侧菜单栏的**持续集成**功能。

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。

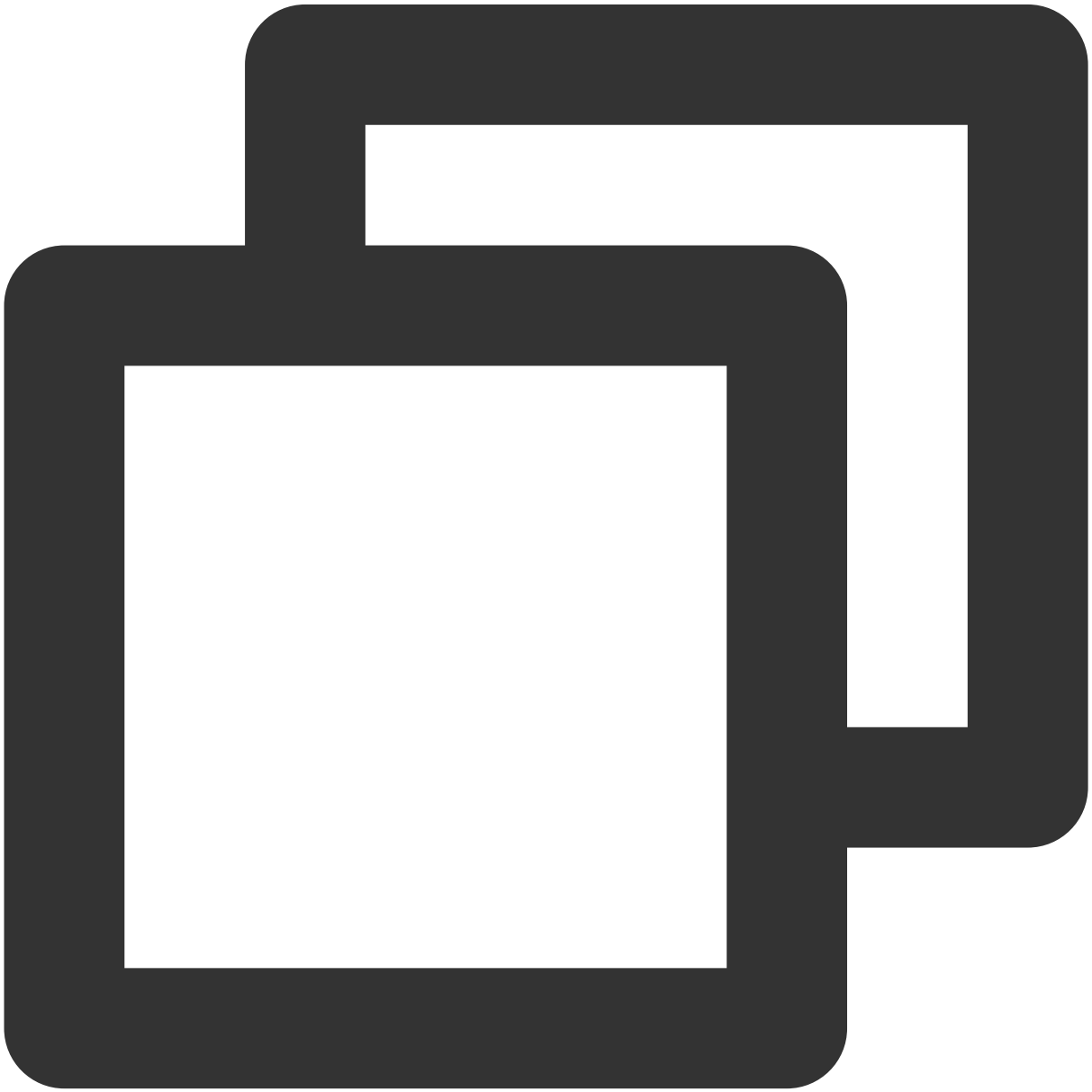


```
pipeline {
    agent any
    stages {
        stage('Test') {
            steps {
                  script {
                    docker.image("ubuntu").inside('-e MY_ENV=123') {
                          sh 'echo ${MY_ENV}'
                    }
                  }
            }
```

```
            }
        }
    }
```

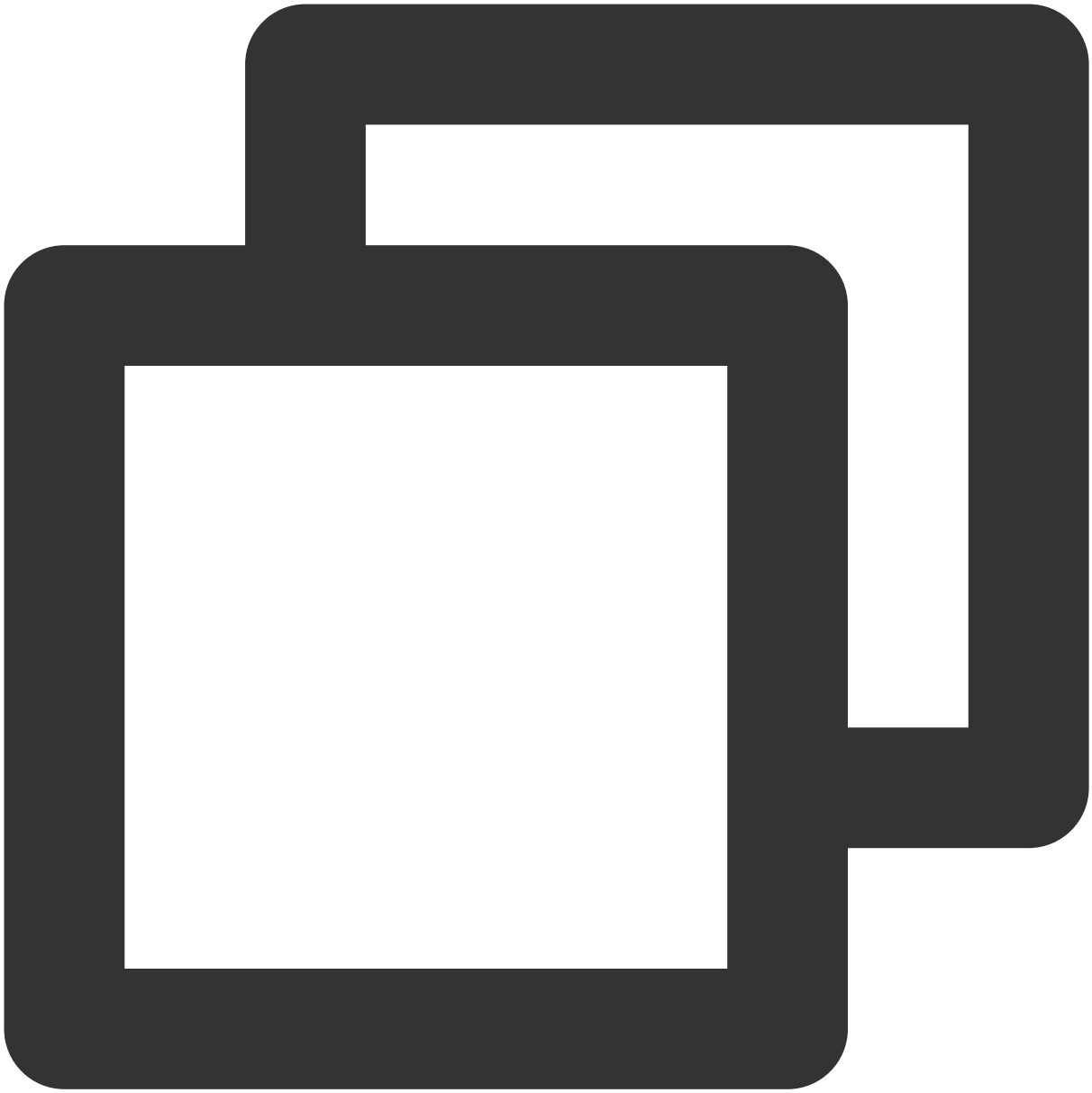
运行指定 Registry 的 Docker 镜像

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'
                        }
                    }
                  }
            }
        }
    }
}
```

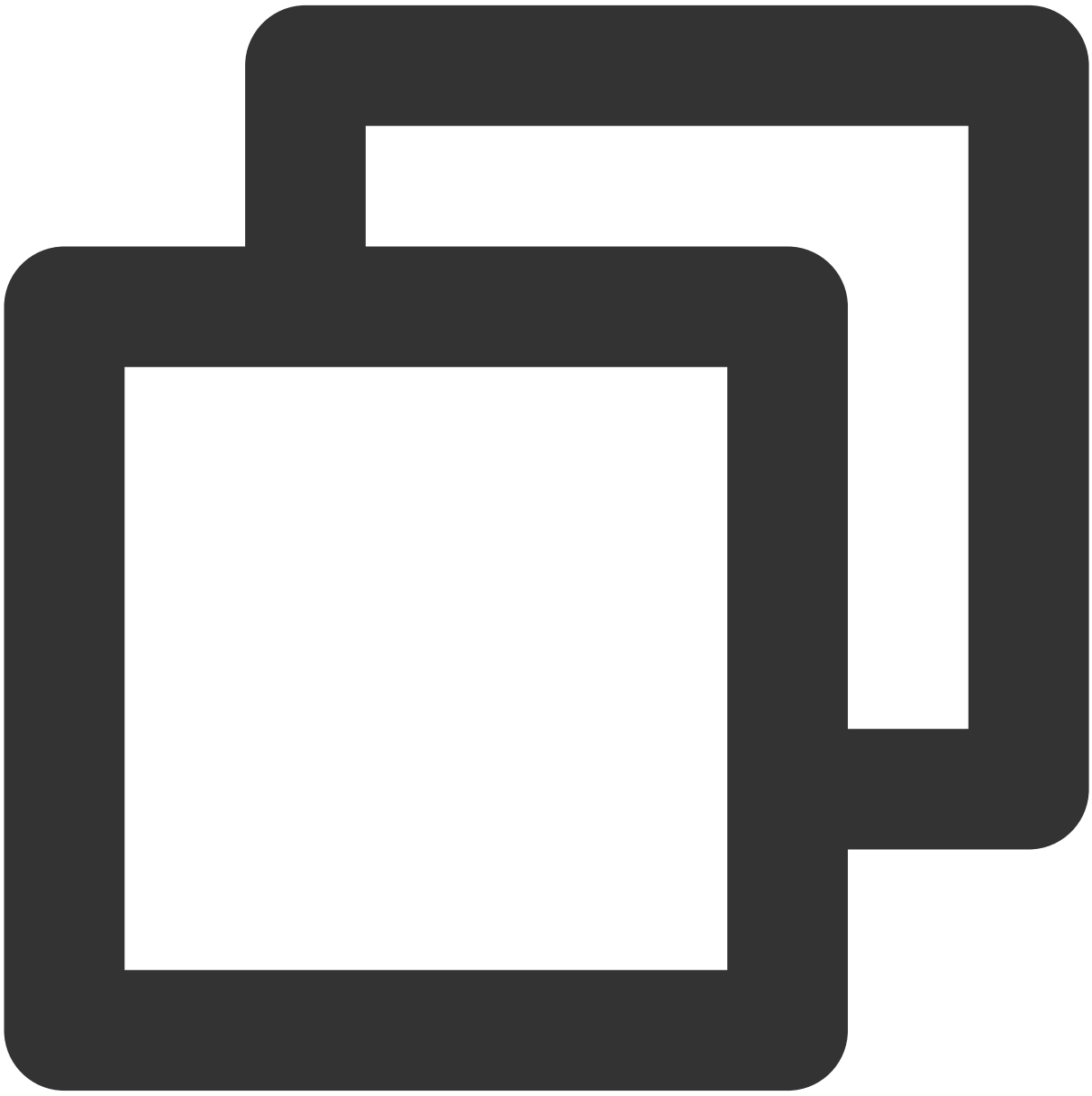
若所配置的 registry 对拉取操作带有鉴权，需要您提供有效的凭证 ID，以下是相应的 Jenkinsfile 参考。



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

                    }
                  }
            }
```

```
        }
    }
}
```

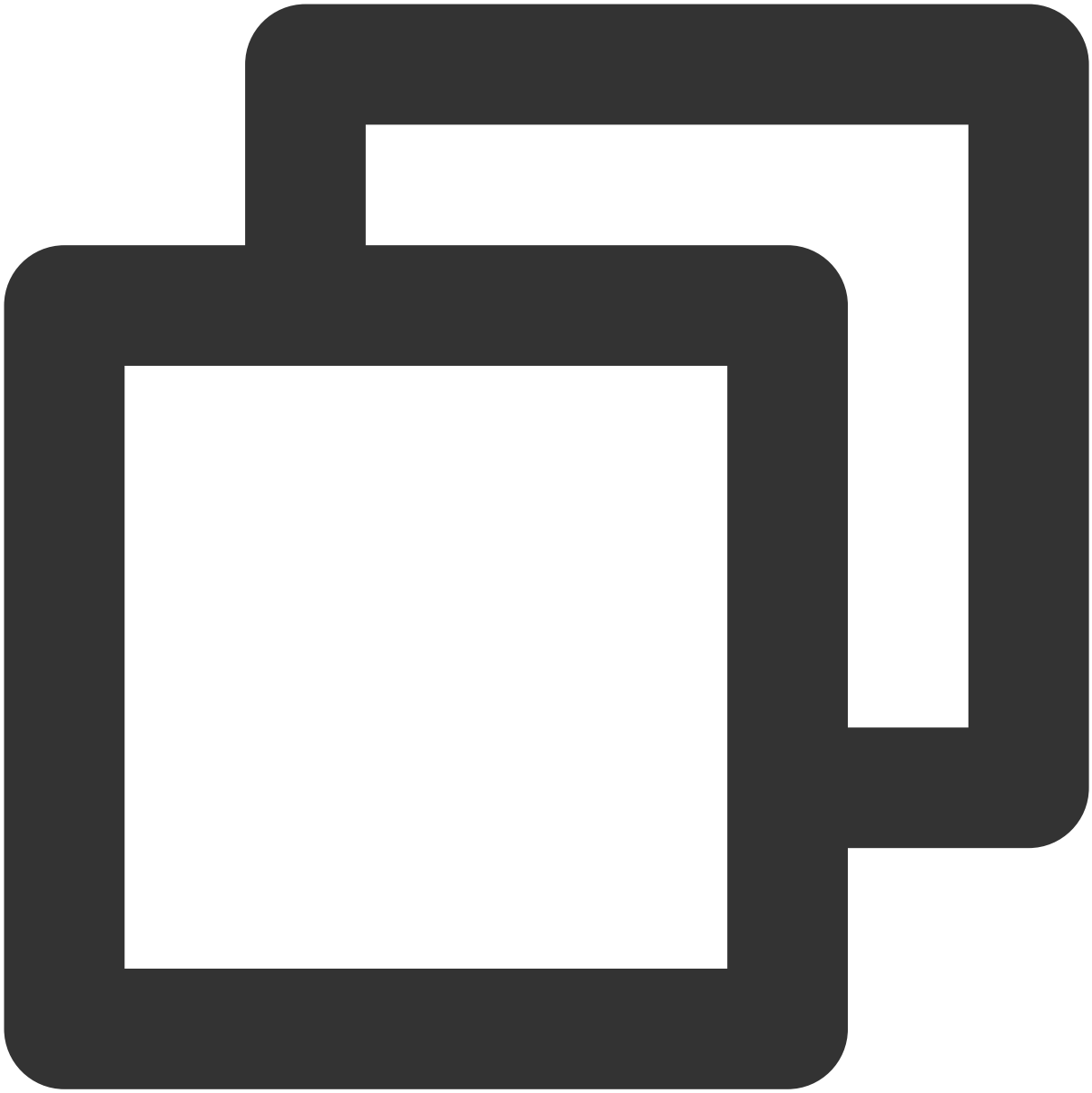
在持续集成过程中构建 Docker 镜像



```
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.
            }
        }
        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。

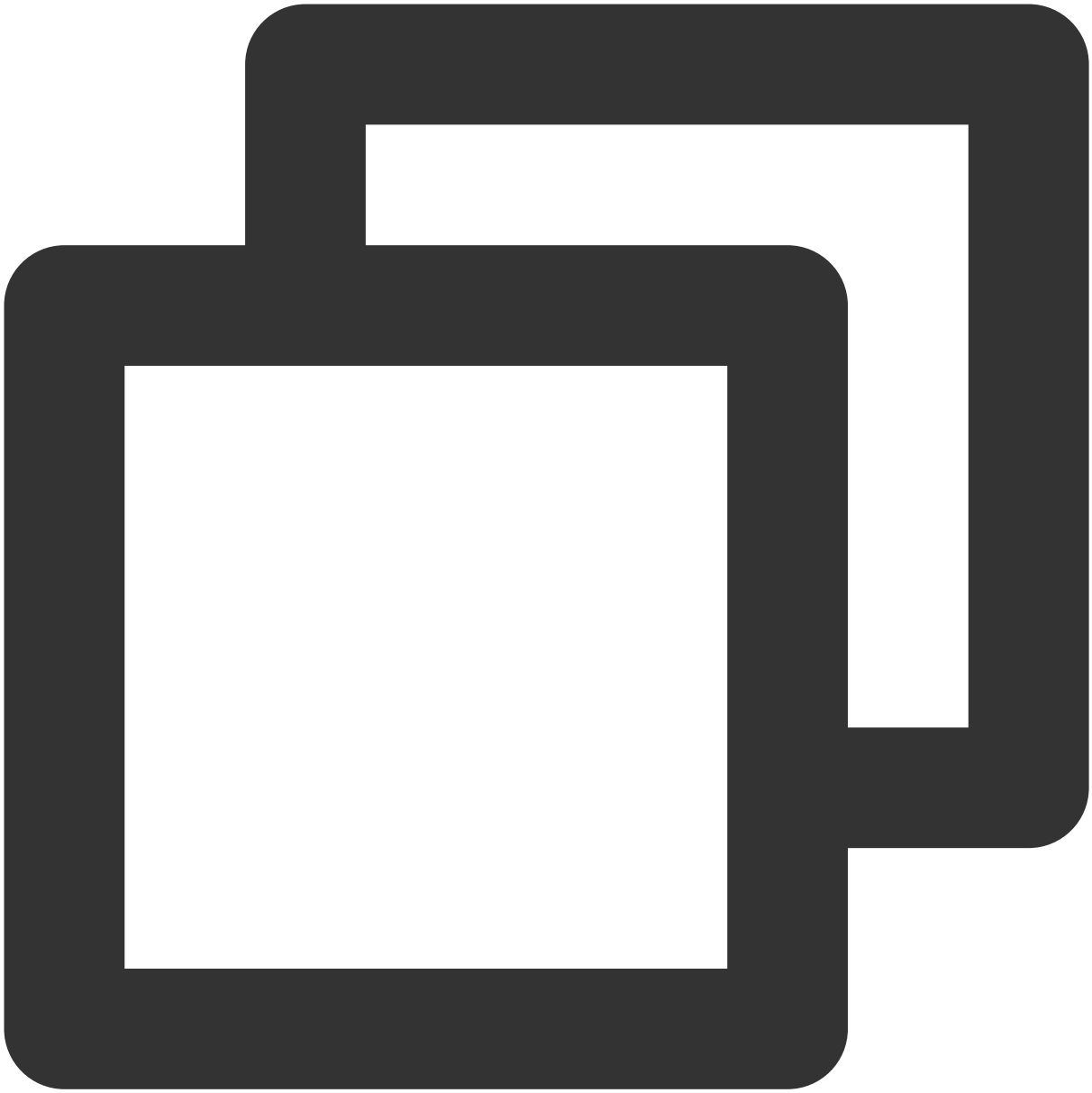


```
pipeline {
    agent any
    stages {
        // 需要检出代码后，才可以使用代码仓库内的 Dockerfile
        stage('Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: env.GIT_BUILD_REF]],
                    userRemoteConfigs: [[url: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

```
        }
        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 ./do
                }
            }
        }
    }
}
```

将 Docker 镜像推送到指定的 Registry

以下是相应的 Jenkinsfile 参考。

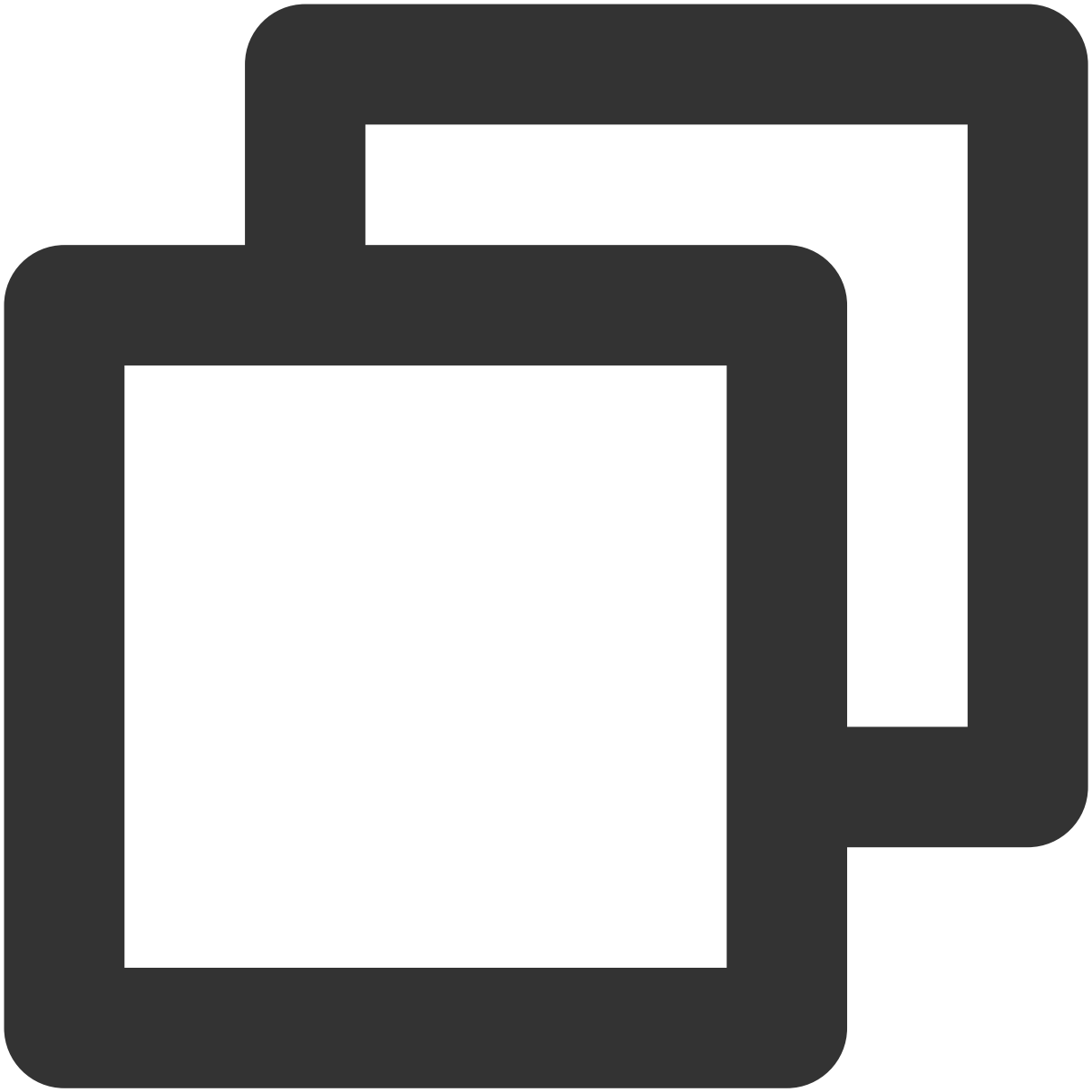


```
pipeline {
    agent any
    stages {
        // 需要检出代码后，才可以使用代码仓库内的 Dockerfile
        stage('Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: env.GIT_BUILD_REF]],
                    userRemoteConfigs: [[url: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

```
            }
        }

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

                    docker.withRegistry('https://registry.example.com', 'my-credent
                        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）。

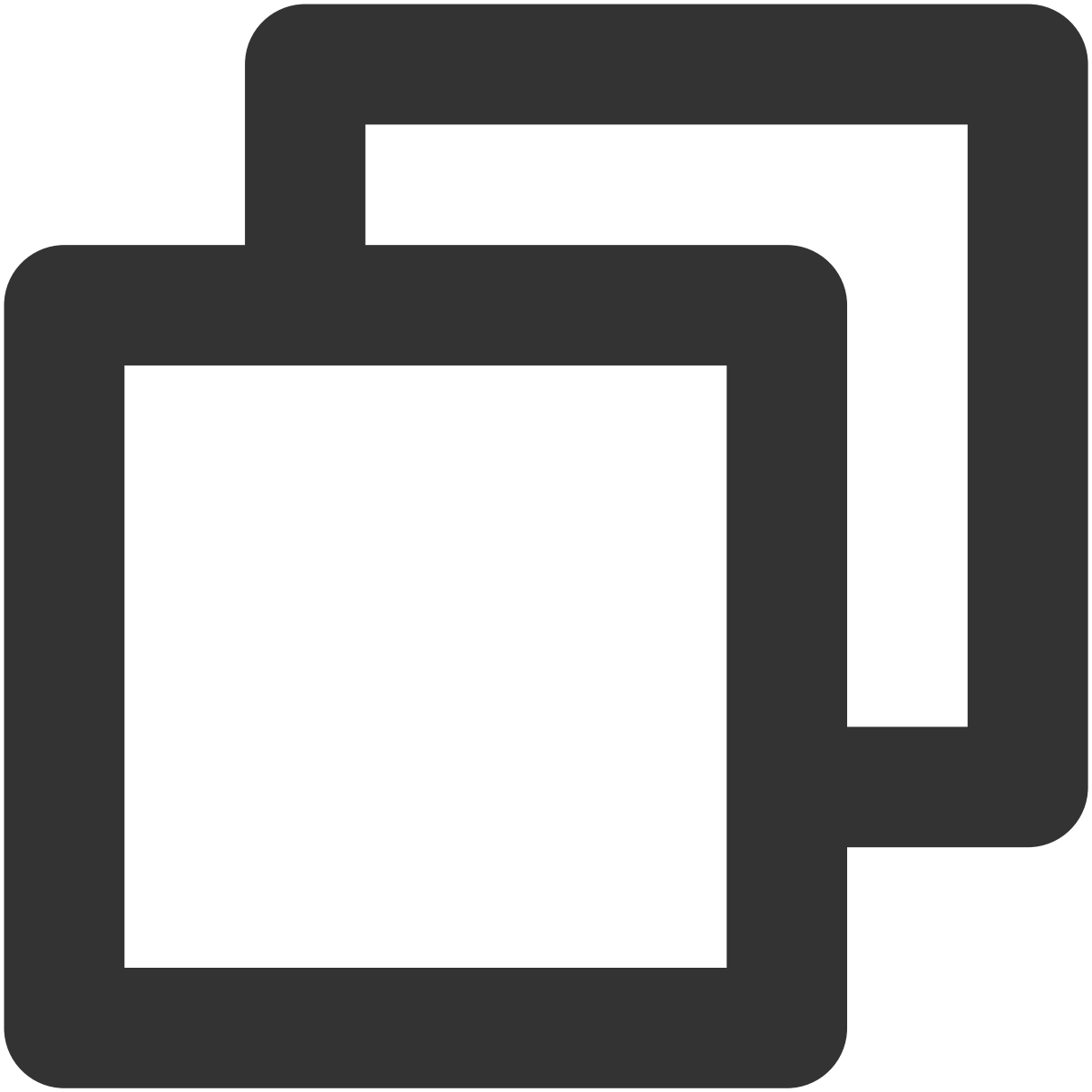


```
pipeline {
    agent any
    stages {
        stage('Test') {
            steps {
                  script {
                    docker.image('mysql:5').withRun('-e "MYSQL_ROOT_PASSWORD=my-sec
                        // 注意：这里 callback 的运行环境并不是上面运行的 mysql:5 环境内，

                        // 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; do
                        }

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

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-sec
                        // 注意：这里 callback 的运行环境并不是上面运行的 mysql:5 环境内，

                        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
```

```
                        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

在持续集成中使用 SSH

最近更新时间：2023-12-29 11:44:51

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。

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。

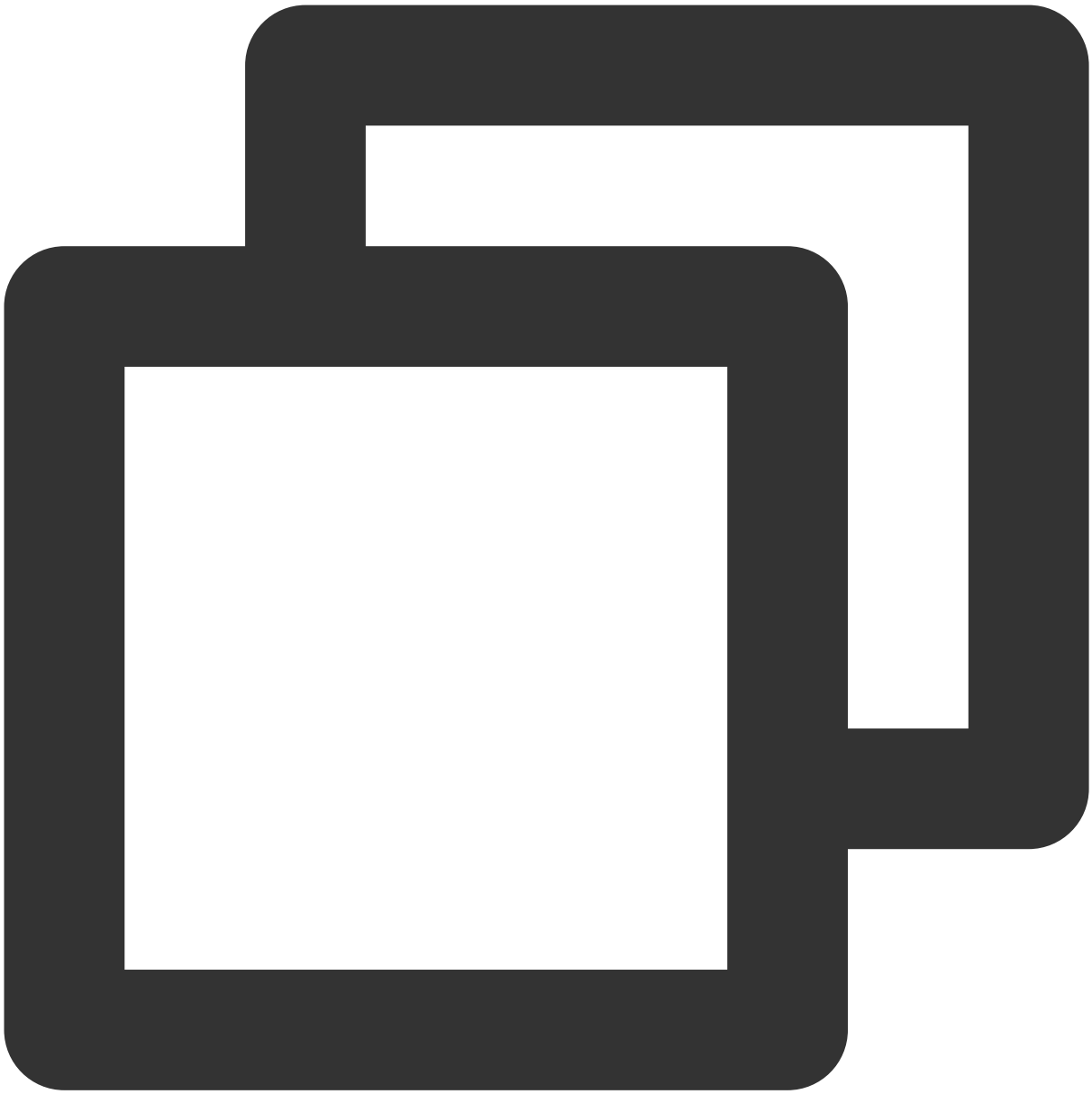
功能介绍

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：

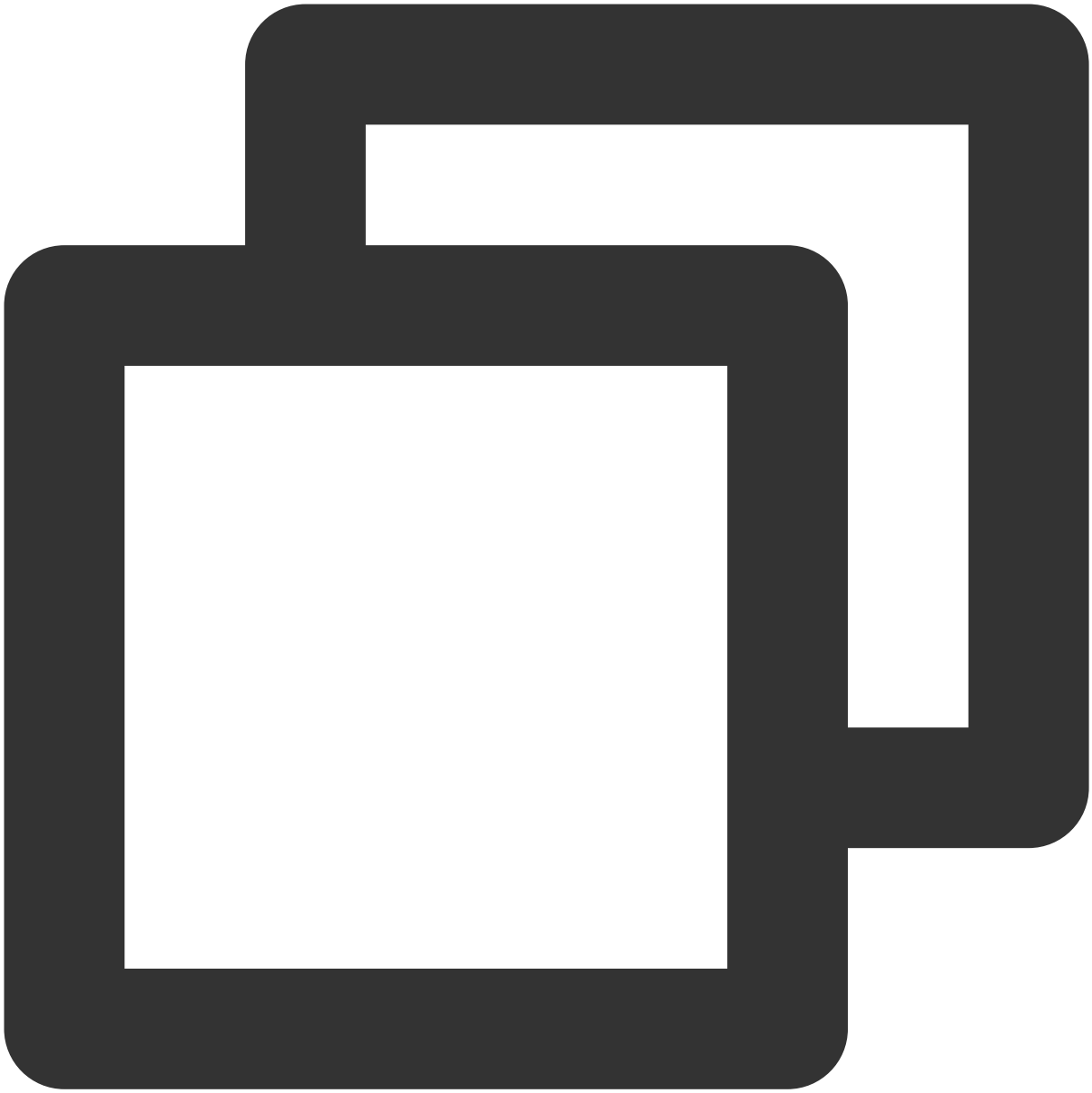


```
def remote = [:]
remote.name = "node"
remote.host = "node.abc.com"
remote.allowAnyHosts = true

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 ; slee
            sshScript remote: remote, script: 'test.sh'
            sshPut remote: remote, from: 'test.sh', into: '.'
            sshGet remote: remote, from: 'test.sh', into: 'test_new.sh', override:
            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: '
        // ssh 登陆用户名
        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 ; slee
            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。