# 1、docker部署

|  |
| --- |
| docker run \  -u root \  --name=jenkins \  -d \  -p 8080:8080 \  -p 50000:50000 \  -v jenkins-data:/var/jenkins\_home \  -v /etc/localtime:/etc/localtime:ro \  -v /var/run/docker.sock:/var/run/docker.sock \  --restart=always \  jenkinsci/blueocean |

jenkinsci/jenkins 是没有 blueocean插件的，得自己装

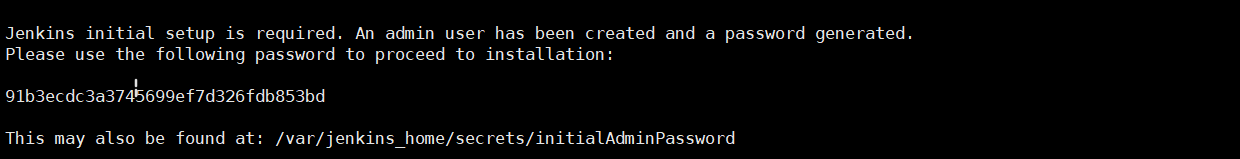
jenkinsci/blueocean：带了的

#自己构建镜像 RUN的时候就把时区设置好

#如果是别人的镜像，docker hub，UTC； 容器运行时 ， -v

/etc/localtime:/etc/localtime:ro

#docker logs -f jenkins



# 2、插件加速

需要在创建完用户，并且跳过推荐插件安装再更改。否则更改无效，还是会更新回去www.google.com

#cd /var/lib/docker/volumes/jenkins-data/\_data/updates

#cp -a default.json default.json.ori

sed -i 's#https://updates.jenkins.io/download#https://mirrors.tuna.tsinghua.edu.cn/jenkins#g' default.json

sed -i 's#http://www.google.com#https://www.baidu.com#g' default.json

#docker restart jenkins

# 3、创建Jenkins 流水线项目

## 3.1、Jenkins的工作流程

1、先定义一个流水线项目，指定项目的git位置

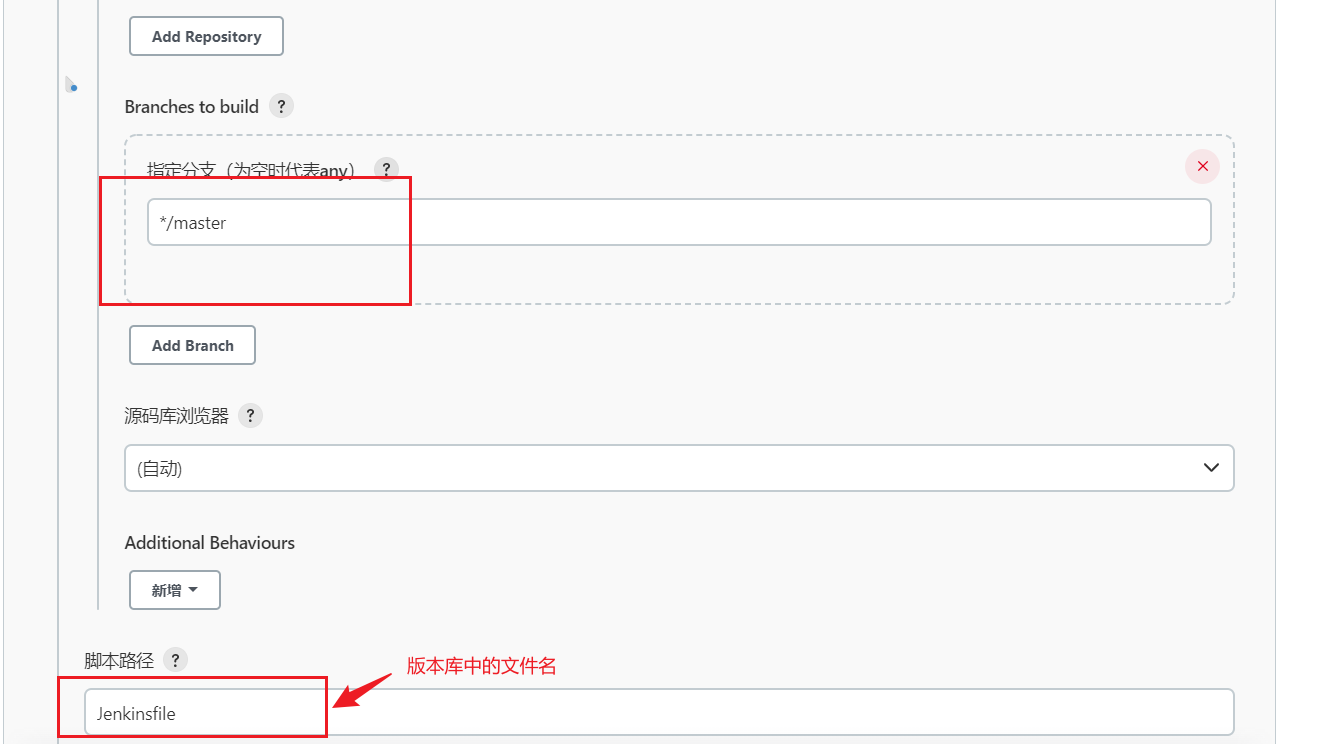
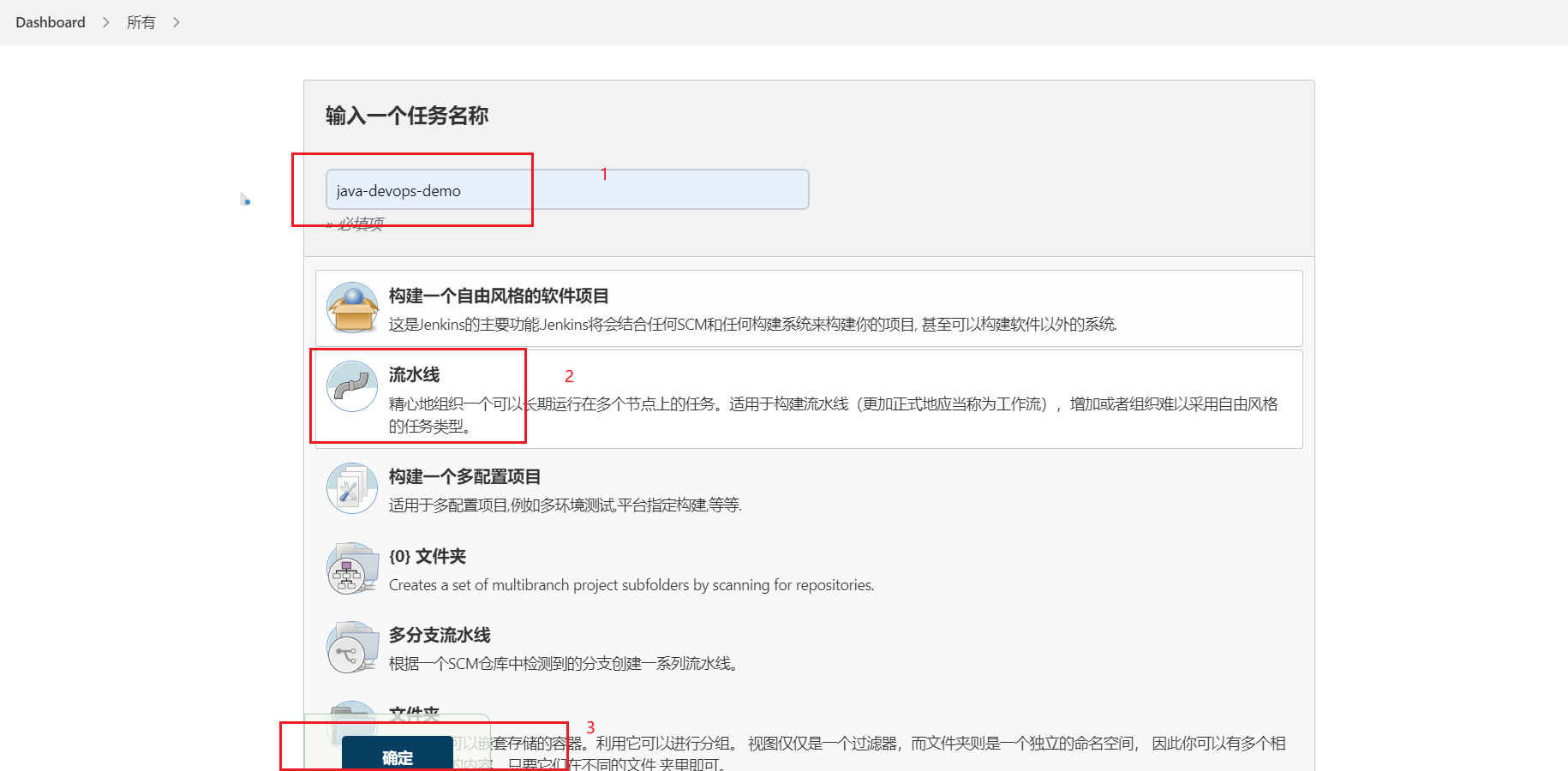
**流水线启动**

1、先去git位置自动拉取代码

2、解析拉取代码里面的Jenkinsfile文件

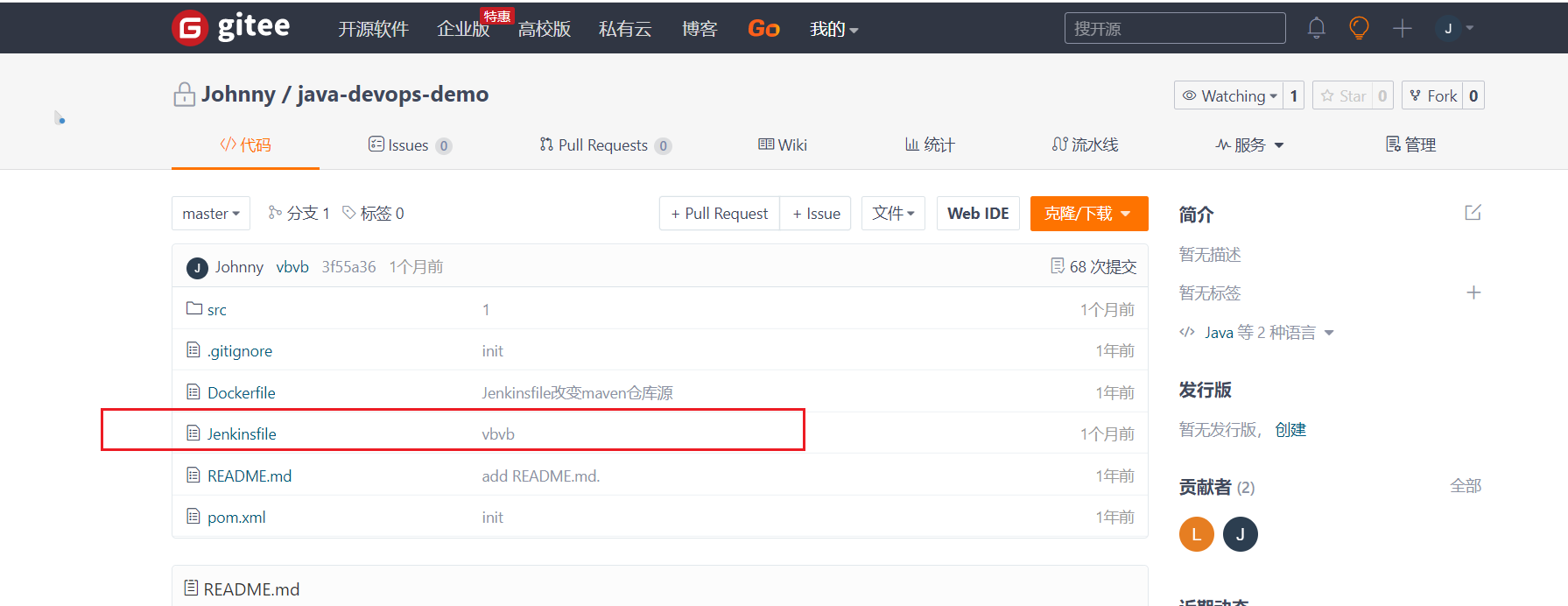
3、按照Jenkinsfile指定的流水线开始加工项目

## 3.2、创建任务



版本库如下截图

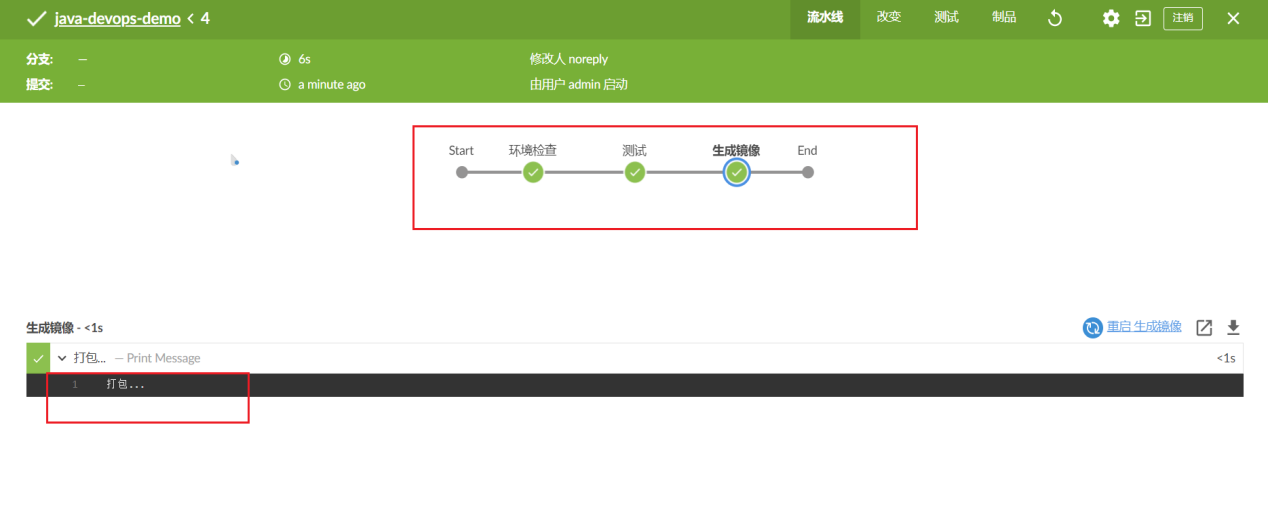
<https://gitee.com/itsutdy/java-devops-demo/tree/master>



## 3.3、简单流水线

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  echo "正在检测基本信息"  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |

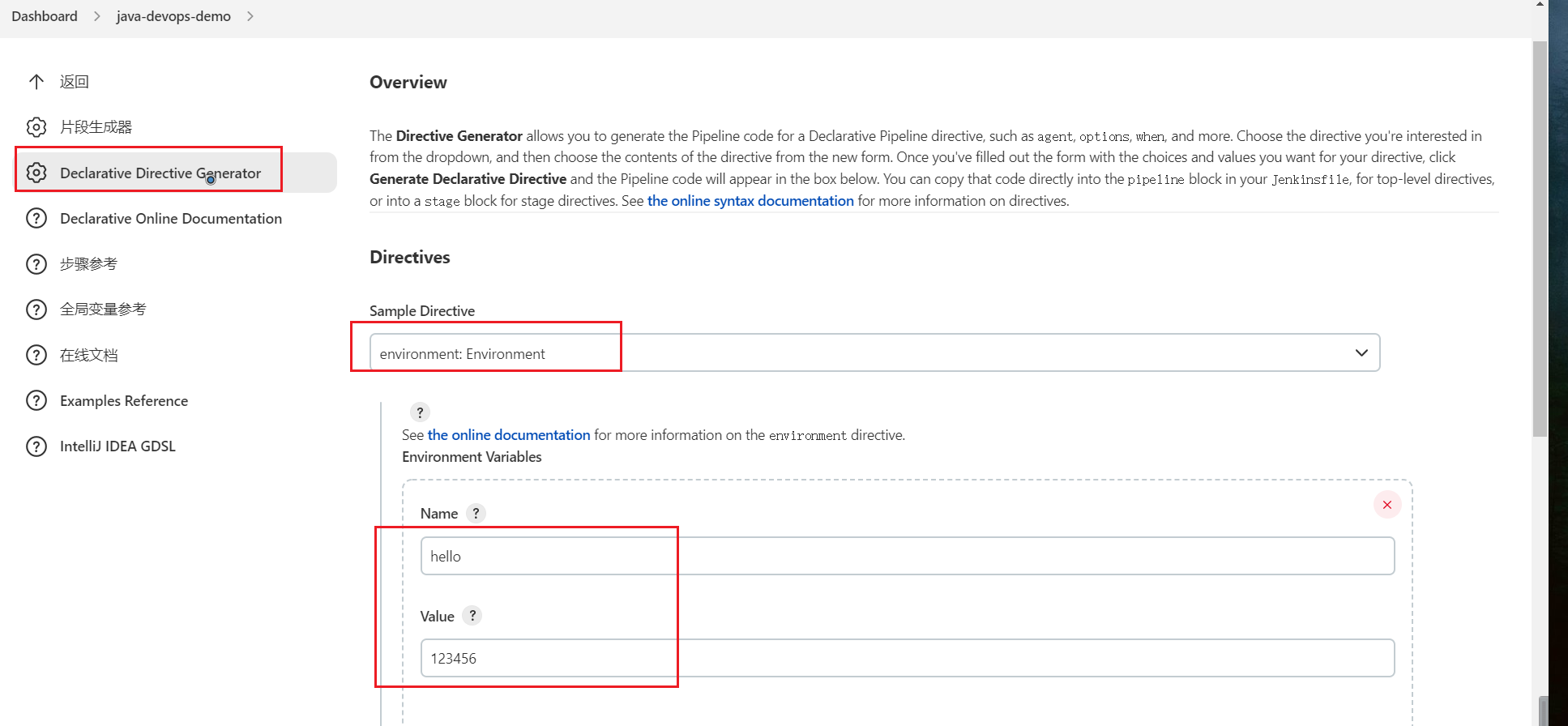
## 3.4、构建成功如下

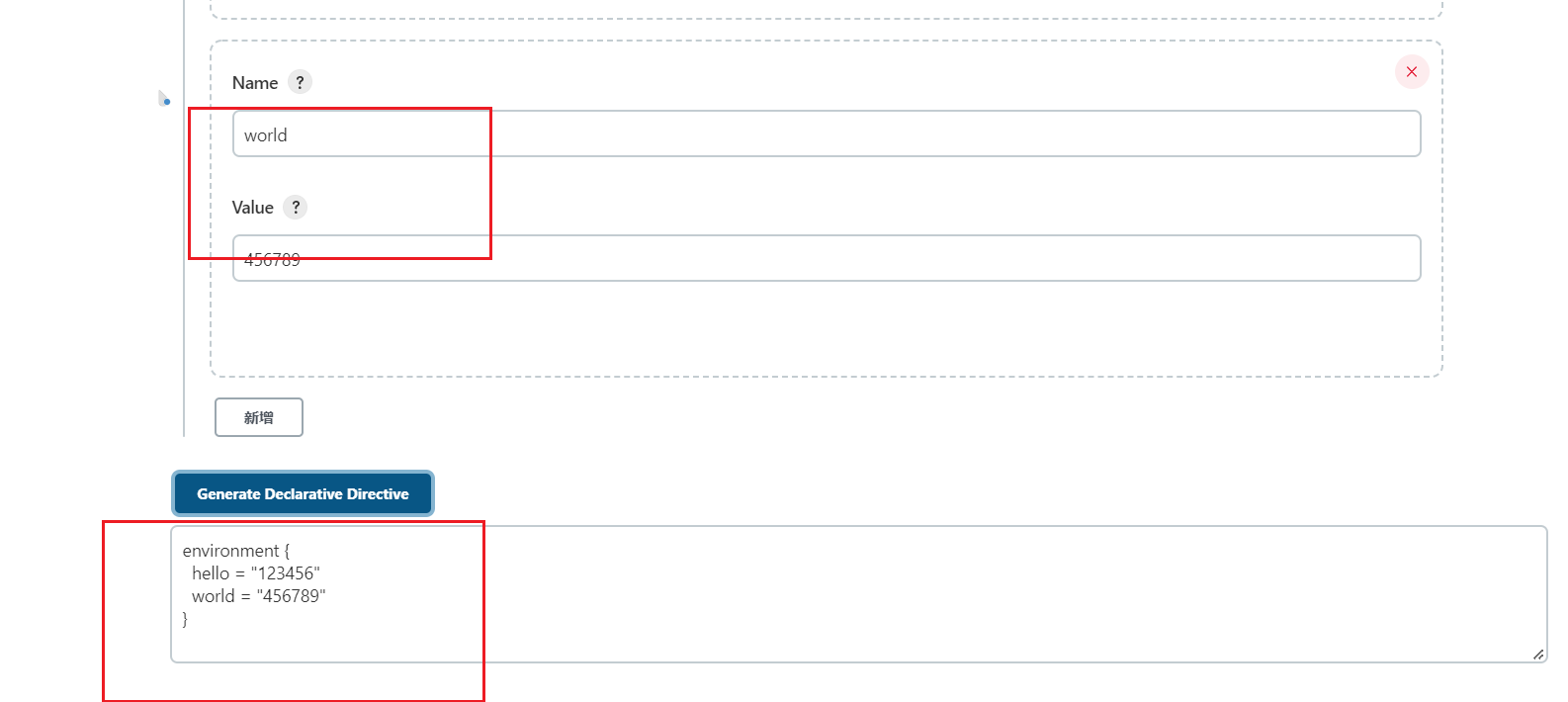


## 3.5、添加环境变量

### 3.5.1、流水线语法生产



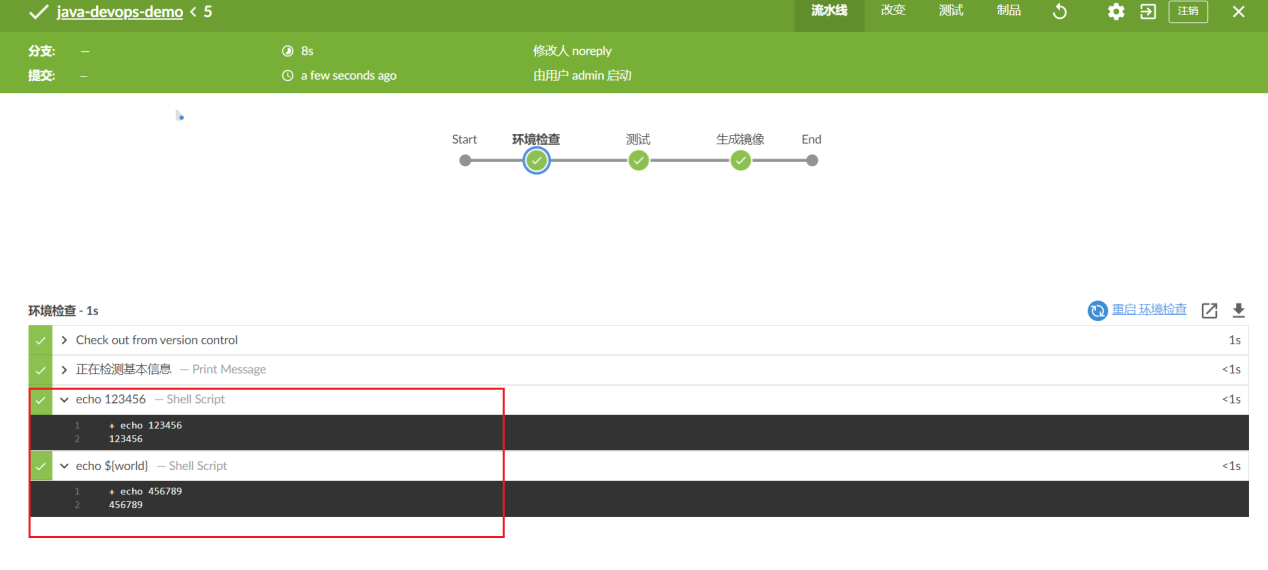




### 3.5.2编辑Jenkinsfile

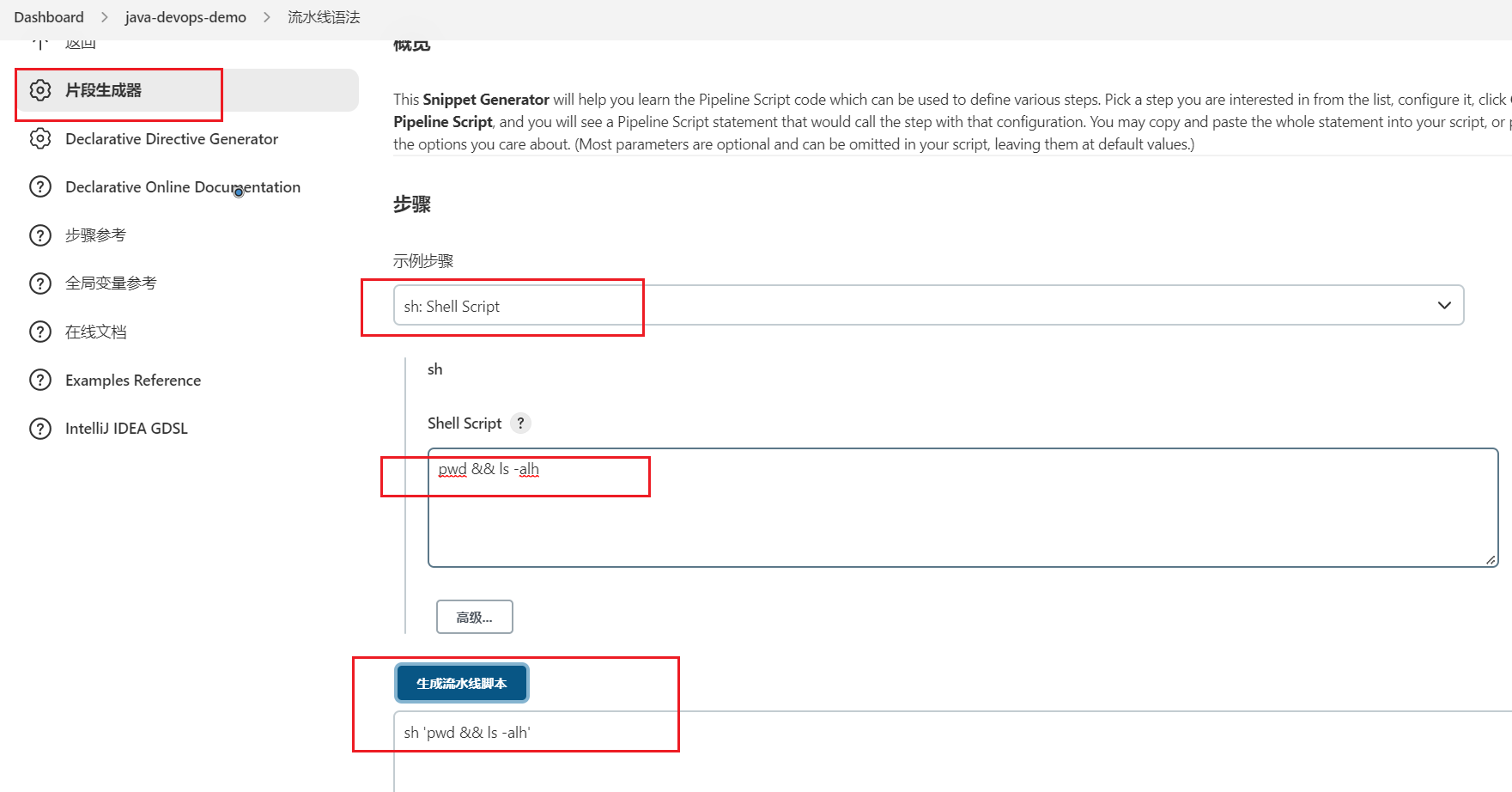
|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  echo "正在检测基本信息"  sh "echo $hello"  sh 'echo ${world}'  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |

### 3.5.3、结果如下



## 3.6、Sh & prientenv

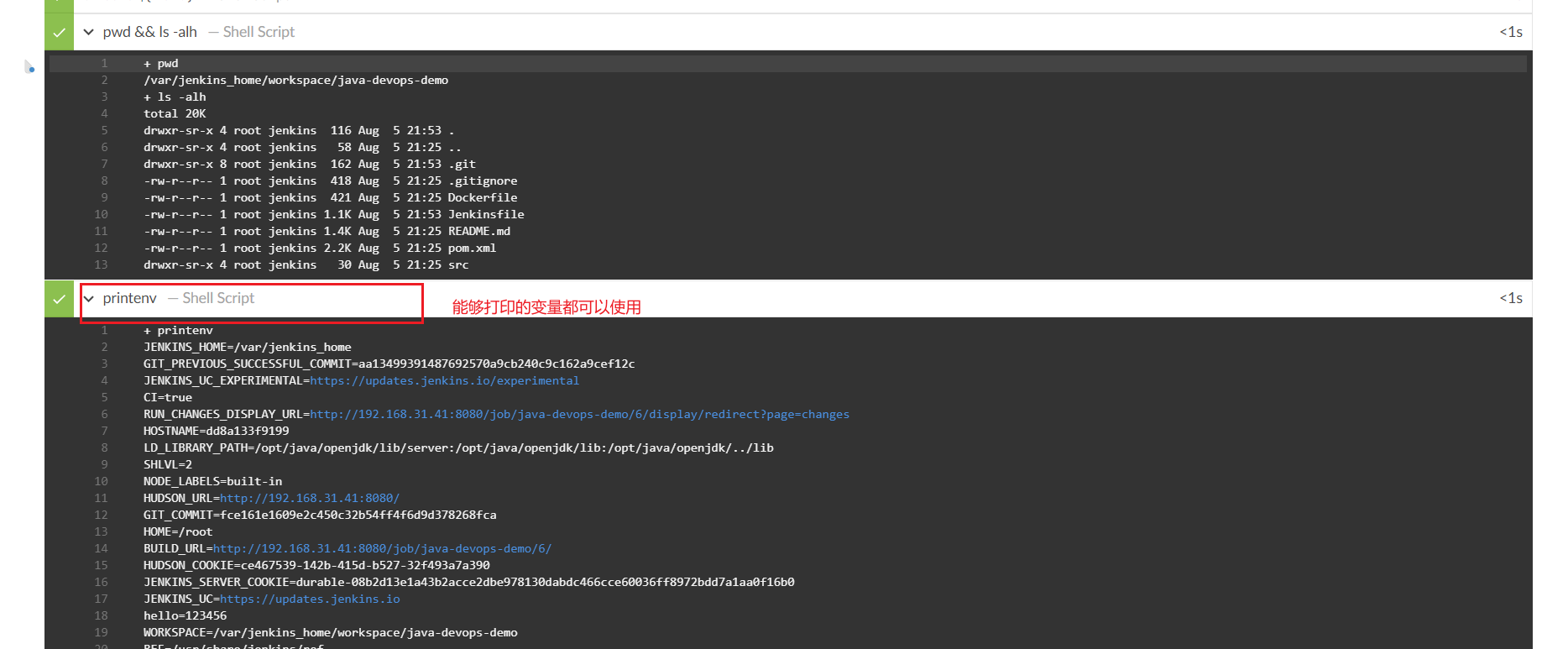
### 3.6.1流水线语法



### 3.6.2 编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  echo "正在检测基本信息"  sh "echo $hello"  sh 'echo ${world}'  sh 'pwd && ls -alh' //当前在那个文件夹  sh 'printenv'  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |

### 3.6.3 结果



如GIT\_BRANCH 变量

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  echo "正在检测基本信息"  sh "echo $hello"  sh 'echo ${world}'  sh 'pwd && ls -alh' //当前在那个文件夹  sh 'printenv'  sh "echo ${GIT\_BRANCH}"  echo "${GIT\_BRANCH}"  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |

sh "echo ${GIT\_BRANCH}"

echo "${GIT\_BRANCH}"

**这两个效果一样**

**结果如下，打印出了git分支**



## 3.7、Jenkins重要的点

1. jenkins的家目录 /var/jenkins\_home 已经被我们docker外部挂载了 ； /var/lib/docker/volumes/jenkins-data/\_data

1、WORKSPACE（工作空间）=/var/jenkins\_home/workspace/java-devops-demo

每一个流水线项目，占用一个文件夹位置

BUILD\_NUMBER=5；当前第几次构建

WORKSPACE\_TMP（临时目录）=/var/jenkins\_home/workspace/java-devops-demo@tmp

1. 常用的环境如果没有，jenkins配置环境一大堆操作
2. jenkins\_url ： <http://139.198.27.103:8080/>

# 4、Jenkins环境检查

## 4.1编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'mvn -v'  }  }    stage('编译'){  steps {  echo "正在检测基本信息"  sh "echo $hello"  sh 'echo ${world}'  sh 'pwd && ls -alh' //当前在那个文件夹  sh 'printenv'  sh "echo ${GIT\_BRANCH}"  echo "${GIT\_BRANCH}"  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |

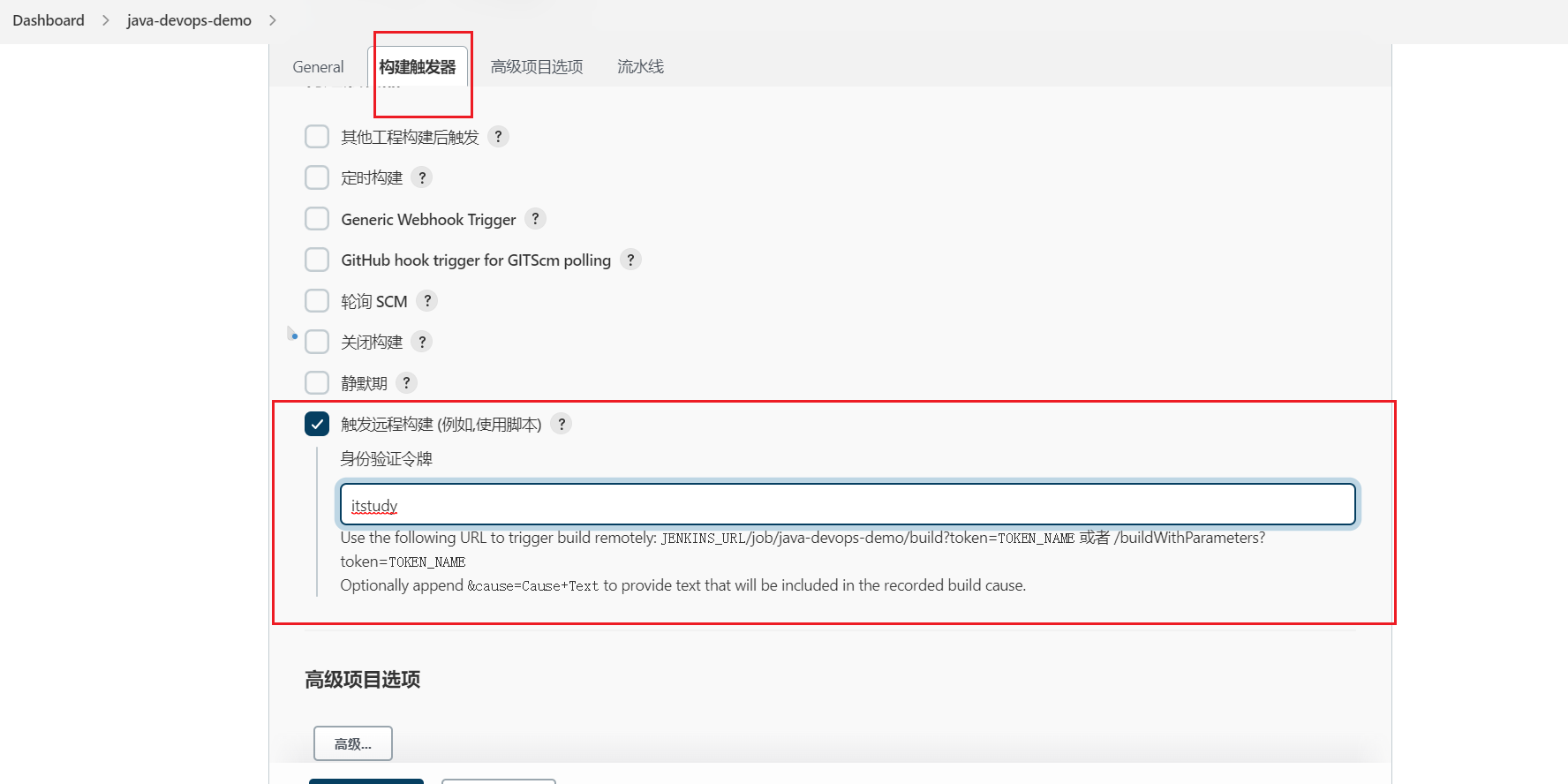
## 4.2 结果



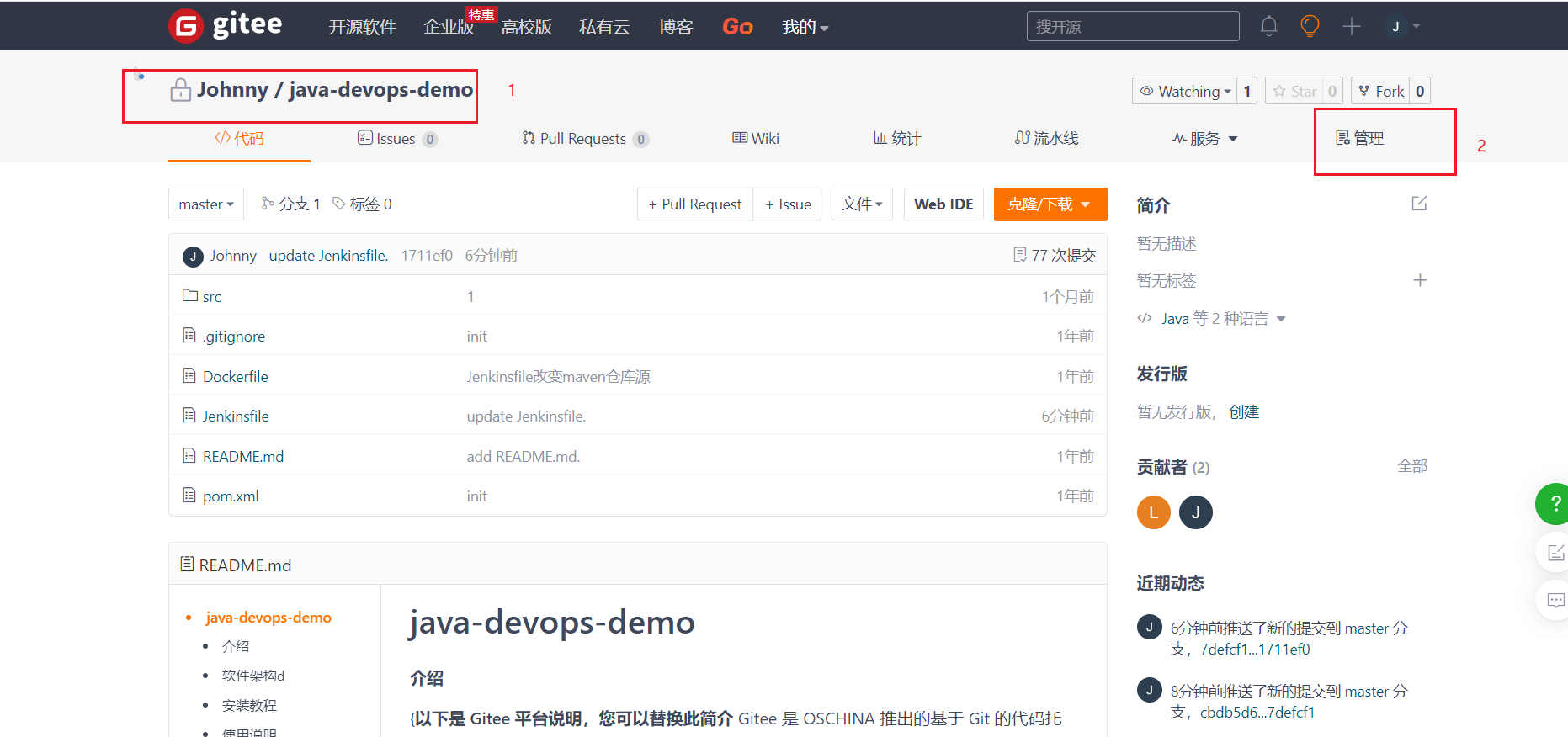
# Gitee远程触发Jenkins构建

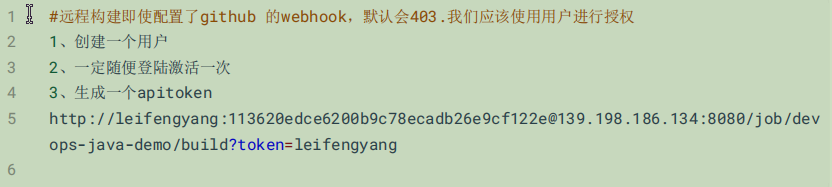
期望效果： 远程的github代码提交了，jenkins流水线自动触发构建

## 5.1Jenkins添加构建触发器



## 5.2 gitee 配置webhooks







<http://zjh:1172258444e160e383b4e963220dcbb8b4@120.77.234.67:8080/job/java-devops-demo/build?token=itstudy>

说明：

zjh： 新建的用户名

1172258444e160e383b4e963220dcbb8b4： apitoken

## 5.2验证





# 6 Jenkins插件安装

**Docker Pipeline && Docker**

安装Docker Pipeline会自动安装docker相关的

这个允许我们自定义agent使用docker环境

**Git Parameter**

解析git参数，允许我们选择分支进行构建

**Active Choices**

可以做到参数的级联选择

**Generic Webhook Trigger**

通用的webhook触发器，构建更强大的webhook功能

**Role-based Authorization Strategy**

RBAC权限指定，给一个用户精确指定权限

**List Git Branches Parameter**

列出分支参数

**Build With Parameters**

基于自定义参数构建

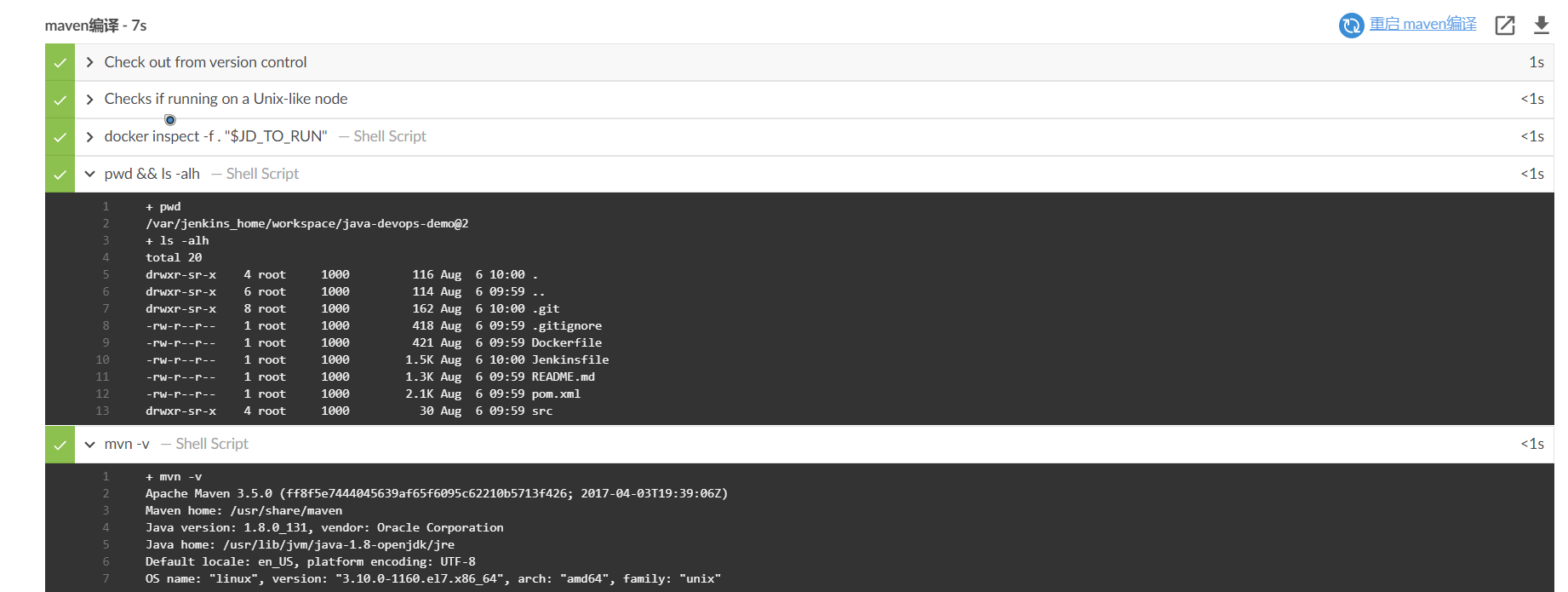
**Email Extension**

邮件通知

# 7 集成maven（docker形式）

## 7.1 简单集成

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  }    stage('maven编译'){  agent {  docker {  image 'maven:3-alpine'  }  }  steps {  sh 'pwd && ls -alh'  sh 'mvn -v'  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |



## 7.2 完全加速

### 7.2.1 配置settings.xml

#cd /var/lib/docker/volumes/jenkins-data/\_data

#mkdir -pv appconfig/maven

#vim settings.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">    <localRepository>/root/m.2</localRepository>  <pluginGroups>  </pluginGroups>  <proxies>  </proxies>  <servers>  </servers>  <mirrors>  <mirror>  <id>aliyunmaven</id>  <mirrorOf>\*</mirrorOf>  <name>阿里云公共仓库</name>  <url>https://maven.aliyun.com/repository/public</url>  </mirror>  <mirror>  <id>aliyunmaven</id>  <mirrorOf>\*</mirrorOf>  <name>阿里云谷歌仓库</name>  <url>https://maven.aliyun.com/repository/google</url>  </mirror>  <mirror>  <id>aliyunmaven</id>  <mirrorOf>\*</mirrorOf>  <name>阿里云阿帕奇仓库</name>  <url>https://maven.aliyun.com/repository/apache-snapshots</url>  </mirror>  <mirror>  <id>aliyunmaven</id>  <mirrorOf>\*</mirrorOf>  <name>阿里云spring仓库</name>  <url>https://maven.aliyun.com/repository/spring</url>  </mirror>  <mirror>  <id>aliyunmaven</id>  <mirrorOf>\*</mirrorOf>  <name>阿里云spring插件仓库</name>  <url>https://maven.aliyun.com/repository/spring-plugin</url>  </mirror>  </mirrors>  <profiles>  </profiles>  </settings> |

用户目录下的.m2是所有jar包的地方

如何自定义使用maven的配置文件

|  |
| --- |
| //流水线这样  stage('编译'){  agent {  docker {  image 'maven:3-alpine' //用完就会杀掉  args '-v maven-repo:/root/.m2'  //docker run -v repo:/root/.m2 这个容器是直接在linux上的,docker  volume也很灵活  }  }  steps {  sh 'pwd && ls -alh'  sh 'mvn -v'  sh 'mvn clean package -s  "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //残留的问题，下一次运行这个命令，已经之前下载的jar还会下载。  }  }  //再给 linux主机挂载的jenkins\_home位置自定义一个 settings.xml 即可  //原理： jenkins在解析流水线期间，可以任意访问jenkins家目录的位置和相关环境信息  //一个配置文件+docker maven agent = maven加速 |

### 7.2.2编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh 'mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  }  }  }  } |

#### 说明1

sh 'mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '

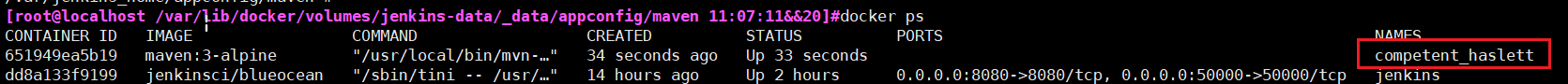
默认所有的可变配置项都推荐放在Jenkins-home的位置，增强移植性

#### **说明2**

settings.xml里面的/root/.m2指的是容器maven运行时容器里的/root/.m2，下载的jar包放在了容器内的/root/.m2位置

<localRepository>/root/.m2</localRepository>

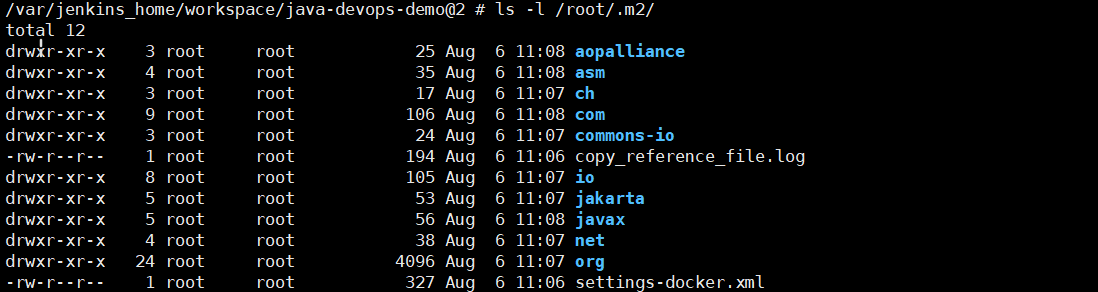
#docker ps



#docker exec -it competent\_haslett sh

/var/jenkins\_home/workspace/java-devops-demo@2 # ls -l /root/.m2/

如下图容器 /root/.m2/路径里的内容。



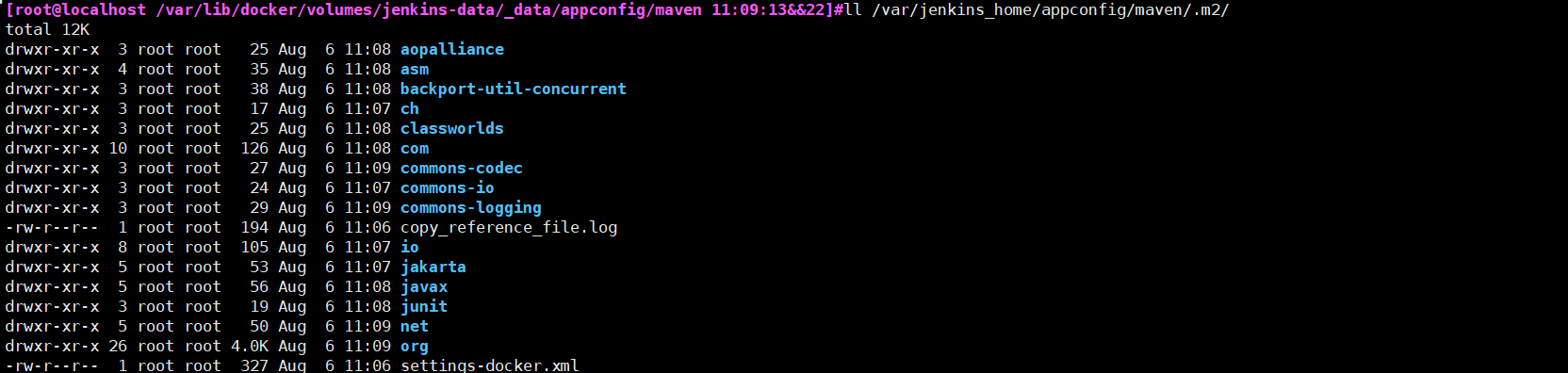
需要把容器里的/root/.m2挂载到本机路径，持久保存，由于maven容器运行完后就会被杀掉，否则下次运行还是会重新下载资源jar包。

args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'

会把本机的/var/jenkins\_home/appconfig/maven/.m2 映射到容器的/root/.m2。下次构建到这个位置继续启动docker maven，容器内/root/.m2下没有东西，可是容器外部有东西，所以就不用下载重复jar包。

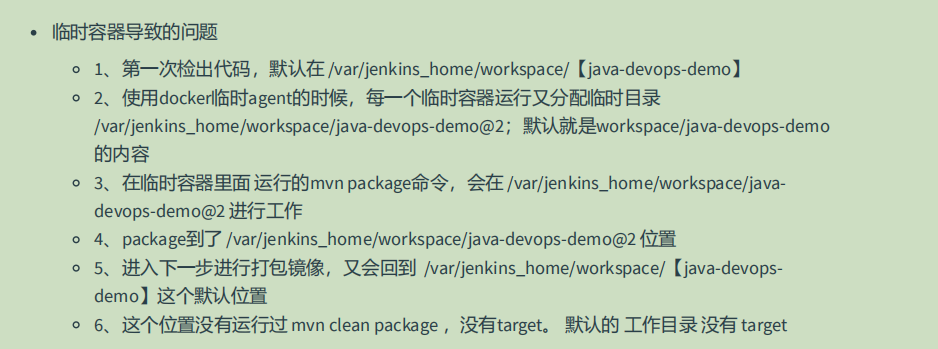
#ll /var/jenkins\_home/appconfig/maven/.m2/

下图本机/var/jenkins\_home/appconfig/maven/.m2路径也同样有下载的内容。





## 7.3打包

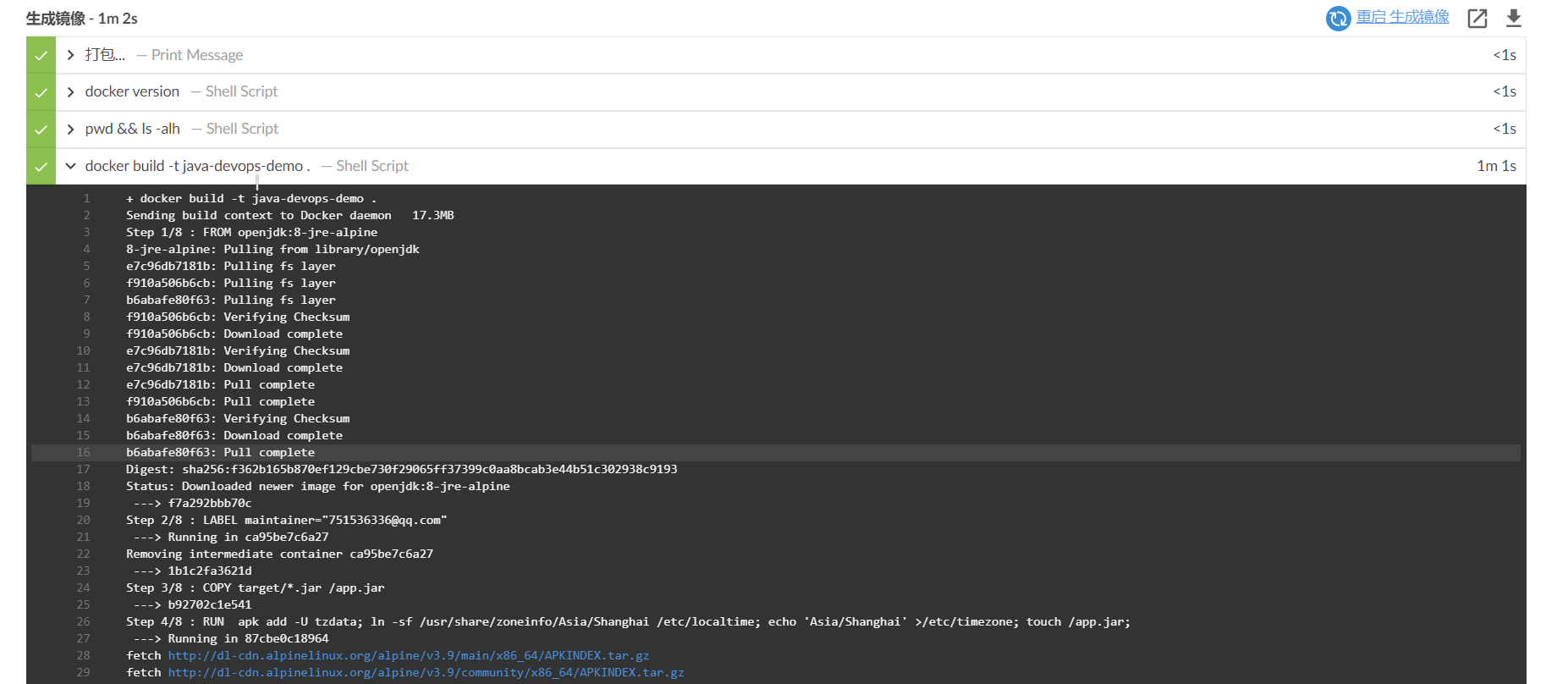


|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }  }  } |

WS = "${WORKSPACE}"

**#cd ${WS} 进入到默认目录以便打包**

验证



## 7.4 post

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }  }  //后置处理过程  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  } |

post可以在stages外部，作为全局，也可以写在每一stage里面

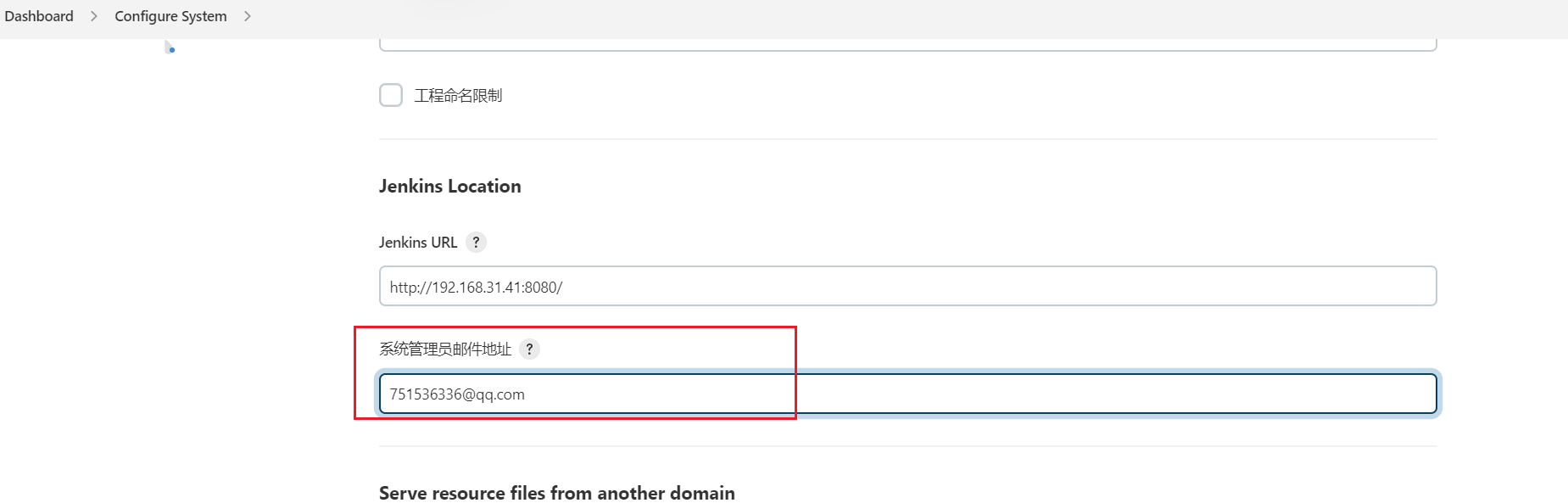
## 7.5 推送报告、邮件

Jenkins设置

系统管理-》系统设置

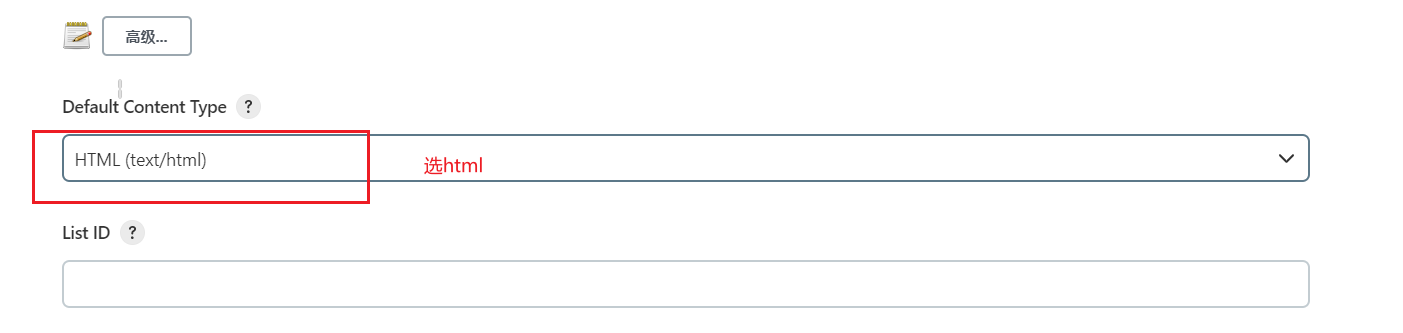
系统管理员邮箱地址

### 7.5.1 Jenkins Location

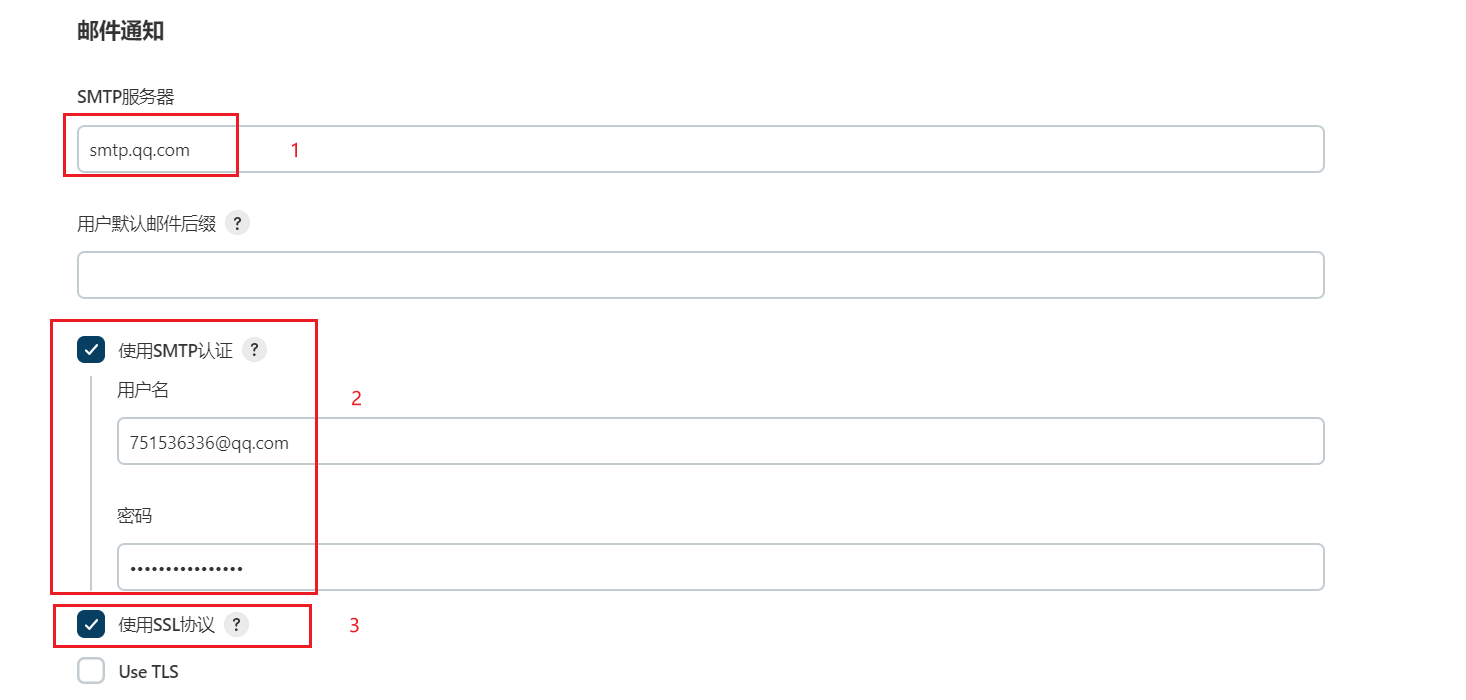


### 7.5.2 Extended E-mail Notification

确保插件Email Extension已安装



### 7.5.3 邮件通知





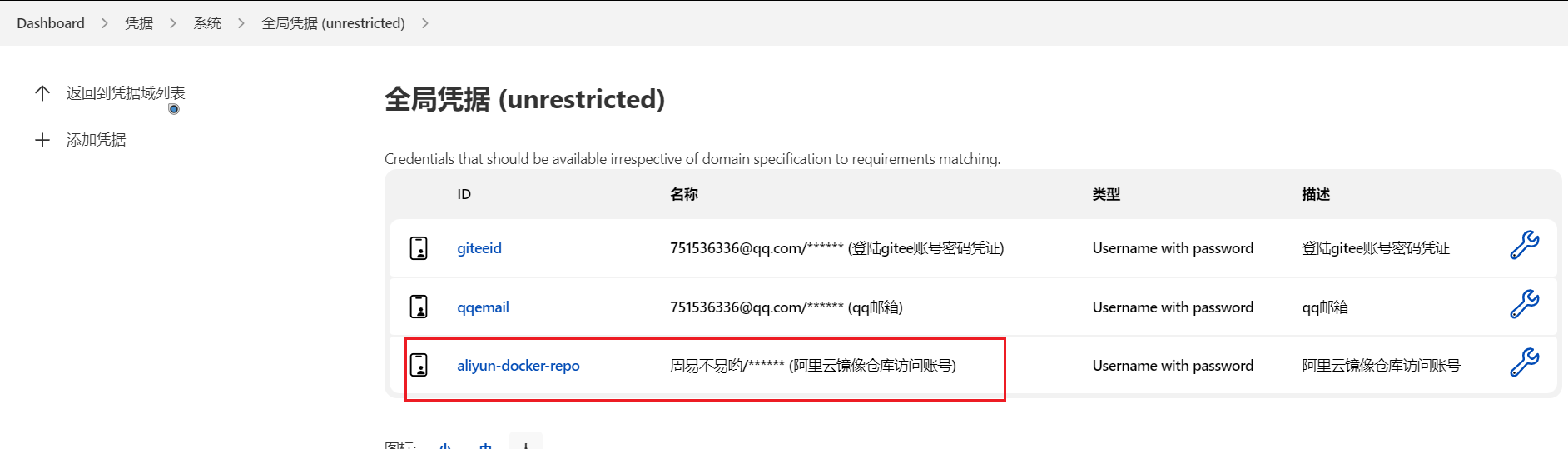
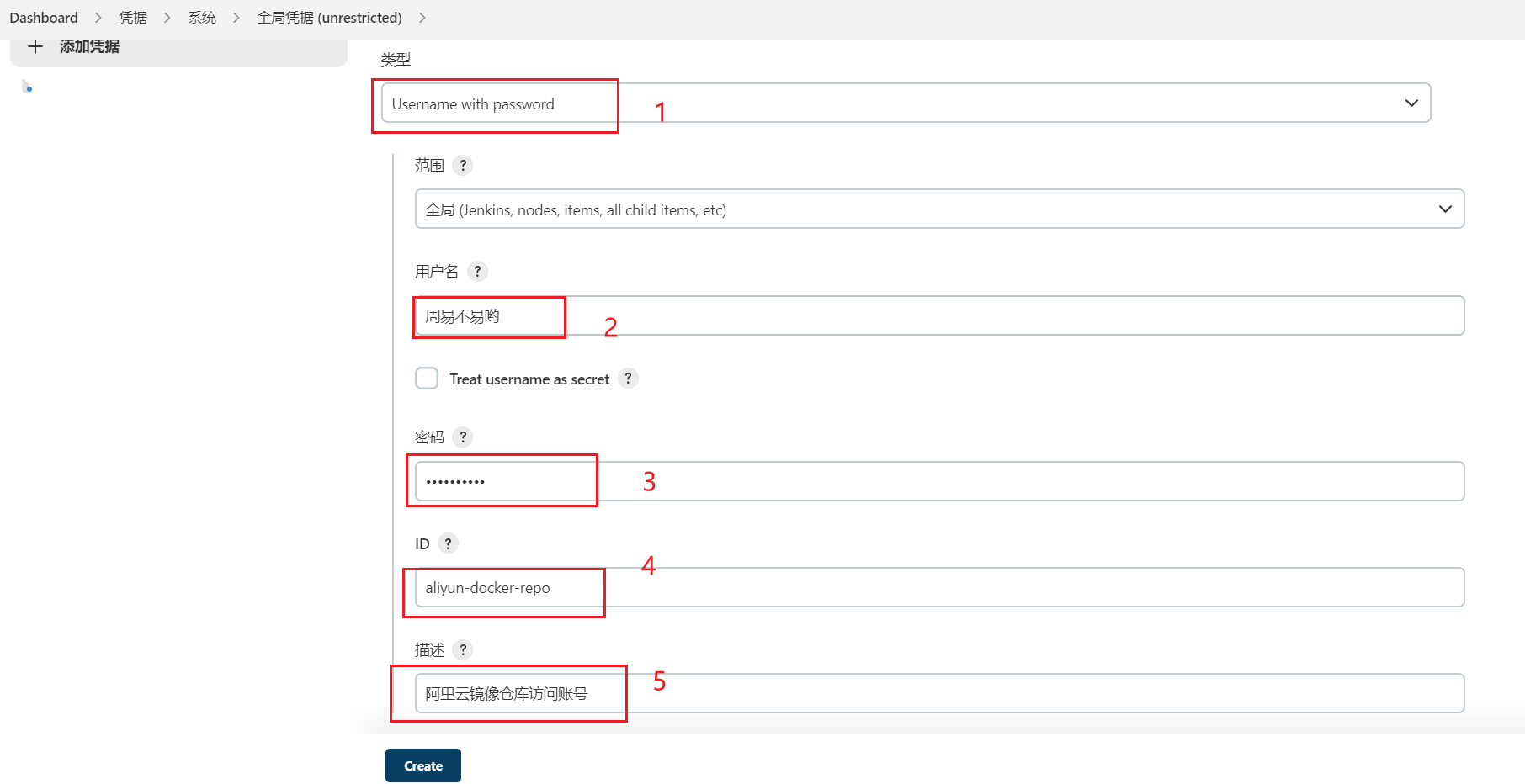
编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }    //5、推送报告  stage("发送报告"){  steps {  //短信通知，购买api接口即可  // sh 'curl -i -k -X POST 'https://gyytz.market.alicloudapi.com/sms/smsSend?mobile=mobile&param=\*\*code\*\*%3A12345%2C\*\*minute\*\*%3A5&smsSignId=2e65b1bb3d054466b82f0c9d125465e2&templateId=908e94ccf08b4476ba6c876d13f084ad' -H 'Authorization:APPCODE dddddddd''  //REST API 所有都行  // sh 'curl '  echo '准备发送报告'  emailext body: '''<!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志</title>  </head>  <body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4"  offset="0">  <table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">  <h3>本邮件由系统自动发出，请勿回复！</h3>  <tr>  <br/>  各位同事，大家好，以下为${PROJECT\_NAME }项目构建信息</br>  <td><font color="#CC0000">构建结果 - ${BUILD\_STATUS}</font></td>  </tr>  <tr>  <td><br />  <b><font color="#0B610B">构建信息</font></b>  <hr size="2" width="100%" align="center" /></td>  </tr>  <tr>  <td>  <ul>  <li>项目名称 ： ${PROJECT\_NAME}</li>  <li>构建编号 ： 第${BUILD\_NUMBER}次构建</li>  <li>触发原因： ${CAUSE}</li>  <li>构建状态： ${BUILD\_STATUS}</li>  <li>构建日志： <a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>  <li>构建 Url ： <a href="${BUILD\_URL}">${BUILD\_URL}</a></li>  <li>工作目录 ： <a href="${PROJECT\_URL}ws">${PROJECT\_URL}ws</a></li>  <li>项目 Url ： <a href="${PROJECT\_URL}">${PROJECT\_URL}</a></li>  </ul>  <h4><font color="#0B610B">最近提交</font></h4>  <ul>  <hr size="2" width="100%" />  ${CHANGES\_SINCE\_LAST\_SUCCESS, reverse=true, format="%c", changesFormat="<li>%d [%a] %m</li>"}  </ul>  详细提交: <a href="${PROJECT\_URL}changes">${PROJECT\_URL}changes</a><br/>  </td>  </tr>  </table>  </body>  </html>''', subject: '${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志', to: '751536336@qq.com'  }  }  }  //后置处理过程  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  } |

# 8高级使用

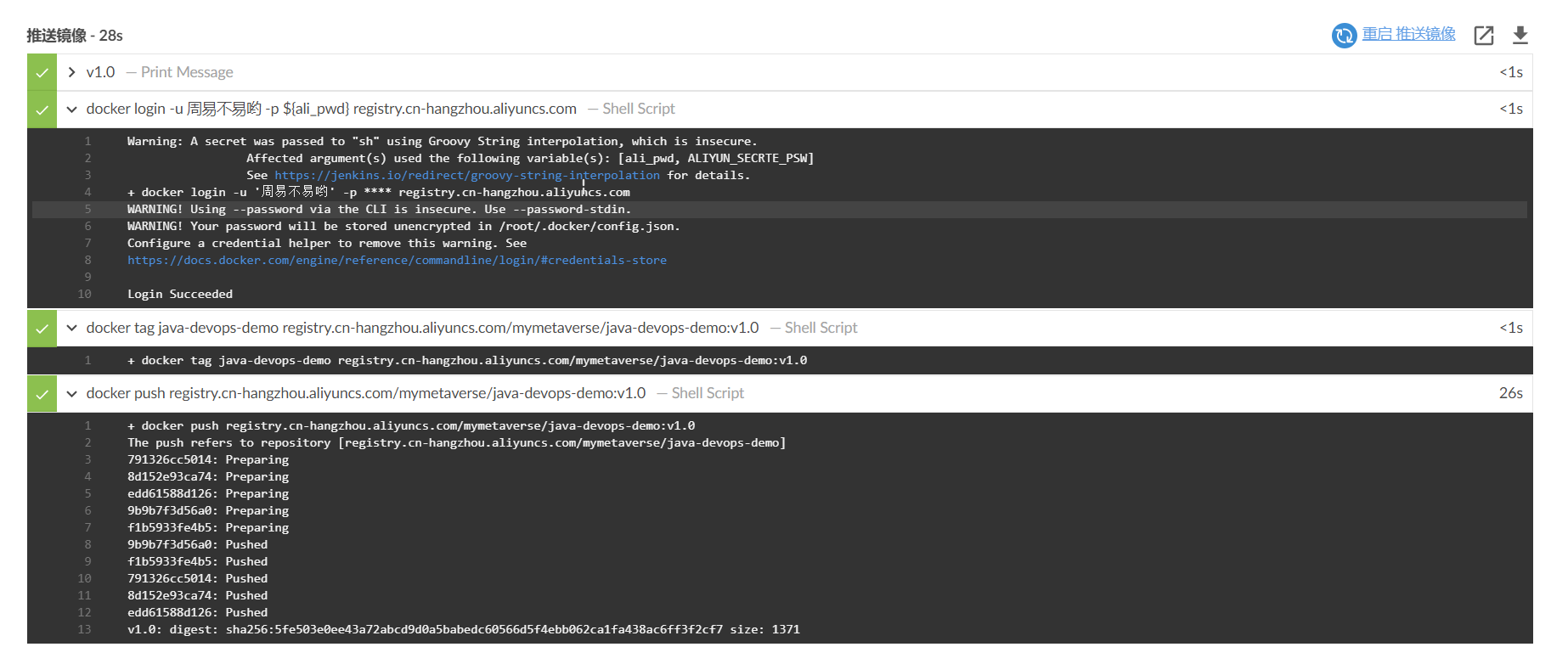
## 8.1推送镜像到阿里云镜像仓库

### 8.1.1 Jenkins创建凭证



### 编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  IMAGE\_VERSION = "v1.0"    //引用Jenkins配置的全局秘钥信息  ALIYUN\_SECRTE=credentials("aliyun-docker-repo")  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }        stage('推送镜像'){  steps {  echo "$IMAGE\_VERSION"  //Generic Webhook Trigger + script脚本 + 其他已讲过的内容 + 其他自己看看官网 = 搞定  //别人提交一个issue，jenkins触发 自动把issue做成 fix分支，让程序员自己去修改  //所有东西都是模板  script {  withCredentials([usernamePassword(credentialsId: 'aliyun-docker-repo', passwordVariable: 'ali\_pwd', usernameVariable: 'ali\_user')]) {  sh "docker login -u ${ali\_user} -p ${ali\_pwd} registry.cn-hangzhou.aliyuncs.com"  sh "docker tag java-devops-demo registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${IMAGE\_VERSION}"  sh "docker push registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${IMAGE\_VERSION}"  }  }  }  }    //5、推送报告  stage("发送报告"){  steps {  //短信通知，购买api接口即可  // sh 'curl -i -k -X POST 'https://gyytz.market.alicloudapi.com/sms/smsSend?mobile=mobile&param=\*\*code\*\*%3A12345%2C\*\*minute\*\*%3A5&smsSignId=2e65b1bb3d054466b82f0c9d125465e2&templateId=908e94ccf08b4476ba6c876d13f084ad' -H 'Authorization:APPCODE dddddddd''  //REST API 所有都行  // sh 'curl '  echo '准备发送报告'  emailext body: '''<!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志</title>  </head>  <body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4"  offset="0">  <table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">  <h3>本邮件由系统自动发出，请勿回复！</h3>  <tr>  <br/>  各位同事，大家好，以下为${PROJECT\_NAME }项目构建信息</br>  <td><font color="#CC0000">构建结果 - ${BUILD\_STATUS}</font></td>  </tr>  <tr>  <td><br />  <b><font color="#0B610B">构建信息</font></b>  <hr size="2" width="100%" align="center" /></td>  </tr>  <tr>  <td>  <ul>  <li>项目名称 ： ${PROJECT\_NAME}</li>  <li>构建编号 ： 第${BUILD\_NUMBER}次构建</li>  <li>触发原因： ${CAUSE}</li>  <li>构建状态： ${BUILD\_STATUS}</li>  <li>构建日志： <a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>  <li>构建 Url ： <a href="${BUILD\_URL}">${BUILD\_URL}</a></li>  <li>工作目录 ： <a href="${PROJECT\_URL}ws">${PROJECT\_URL}ws</a></li>  <li>项目 Url ： <a href="${PROJECT\_URL}">${PROJECT\_URL}</a></li>  </ul>  <h4><font color="#0B610B">最近提交</font></h4>  <ul>  <hr size="2" width="100%" />  ${CHANGES\_SINCE\_LAST\_SUCCESS, reverse=true, format="%c", changesFormat="<li>%d [%a] %m</li>"}  </ul>  详细提交: <a href="${PROJECT\_URL}changes">${PROJECT\_URL}changes</a><br/>  </td>  </tr>  </table>  </body>  </html>''', subject: '${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志', to: '15280808129@139.com'  }  }  }  //后置处理过程  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  } |



## 8.2 交互

### 编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  IMAGE\_VERSION = "v1.0"    //引用Jenkins配置的全局秘钥信息  ALIYUN\_SECRTE=credentials("aliyun-docker-repo")  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }    stage('推送镜像'){  //step外面这么写  input {  message "需要推送远程镜像吗?"  ok "需要"  parameters {  string(name: 'APP\_VER', defaultValue: 'v1.0', description: '生产环境需要部署的版本')  choice choices: ['bj-01', 'sh-02', 'wuhan-01'], description: '部署的大区', name: 'DEPLOY\_WHERE'  }  }  steps{  script {  withCredentials([usernamePassword(credentialsId: 'aliyun-docker-repo', passwordVariable: 'ali\_pwd', usernameVariable: 'ali\_user')]) {  sh "docker login -u ${ali\_user} -p ${ali\_pwd} registry.cn-hangzhou.aliyuncs.com"  sh "docker tag java-devops-demo registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${APP\_VER}"  sh "docker push registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${APP\_VER}"  }  }  }  }    //5、推送报告  stage("发送报告"){  steps {  //短信通知，购买api接口即可  // sh 'curl -i -k -X POST 'https://gyytz.market.alicloudapi.com/sms/smsSend?mobile=mobile&param=\*\*code\*\*%3A12345%2C\*\*minute\*\*%3A5&smsSignId=2e65b1bb3d054466b82f0c9d125465e2&templateId=908e94ccf08b4476ba6c876d13f084ad' -H 'Authorization:APPCODE dddddddd''  //REST API 所有都行  // sh 'curl '  echo '准备发送报告'  emailext body: '''<!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志</title>  </head>  <body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4"  offset="0">  <table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">  <h3>本邮件由系统自动发出，请勿回复！</h3>  <tr>  <br/>  各位同事，大家好，以下为${PROJECT\_NAME }项目构建信息</br>  <td><font color="#CC0000">构建结果 - ${BUILD\_STATUS}</font></td>  </tr>  <tr>  <td><br />  <b><font color="#0B610B">构建信息</font></b>  <hr size="2" width="100%" align="center" /></td>  </tr>  <tr>  <td>  <ul>  <li>项目名称 ： ${PROJECT\_NAME}</li>  <li>构建编号 ： 第${BUILD\_NUMBER}次构建</li>  <li>触发原因： ${CAUSE}</li>  <li>构建状态： ${BUILD\_STATUS}</li>  <li>构建日志： <a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>  <li>构建 Url ： <a href="${BUILD\_URL}">${BUILD\_URL}</a></li>  <li>工作目录 ： <a href="${PROJECT\_URL}ws">${PROJECT\_URL}ws</a></li>  <li>项目 Url ： <a href="${PROJECT\_URL}">${PROJECT\_URL}</a></li>  </ul>  <h4><font color="#0B610B">最近提交</font></h4>  <ul>  <hr size="2" width="100%" />  ${CHANGES\_SINCE\_LAST\_SUCCESS, reverse=true, format="%c", changesFormat="<li>%d [%a] %m</li>"}  </ul>  详细提交: <a href="${PROJECT\_URL}changes">${PROJECT\_URL}changes</a><br/>  </td>  </tr>  </table>  </body>  </html>''', subject: '${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志', to: '15280808129@139.com'  }  }  }  //后置处理过程  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  } |

### 验证



## 8.3 判断部署到哪个环境

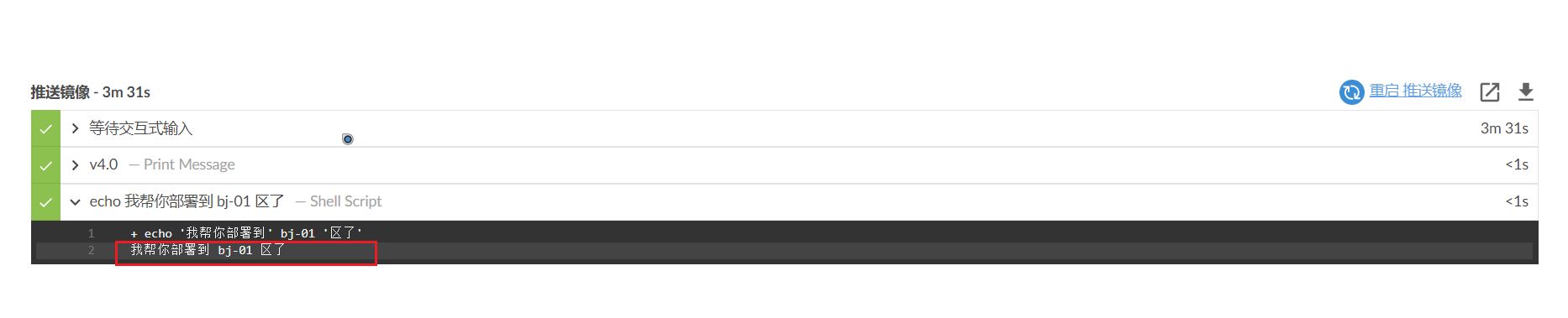
### 编辑Jenkinsfile

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  IMAGE\_VERSION = "v1.0"    //引用Jenkins配置的全局秘钥信息  ALIYUN\_SECRTE=credentials("aliyun-docker-repo")  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }    stage('推送镜像'){  //没有起容器代理，默认就是jenkins环境  //step里面卡点这么写  // input message: '需要推送远程镜像吗？', ok: '需要', parameters: [text(defaultValue: 'v1.0', description: '生产环境需要部署的版本', name: 'APP\_VER')]  //step外面这么写  input {  message "需要推送远程镜像吗?"  ok "需要"  parameters {  string(name: 'APP\_VER', defaultValue: 'v1.0', description: '生产环境需要部署的版本')  choice choices: ['bj-01', 'sh-02', 'wuhan-01'], description: '部署的大区', name: 'DEPLOY\_WHERE'  }  }  steps {  //false就直接结束  echo "$APP\_VER"  //脚本方式进行判断，和具体逻辑  // 远程触发，自动分析是哪个分支，如果是master就部署生产，dev就集成测试等  // gulimall  // mall-order  // mall-user  // xx 100个微服务  // 哪一个微服务代码提交了部署哪一个微服务，不用手工干预，只需要一个jenkinsfile  //Generic Webhook Trigger 远程触发jenkins，jenkins获取当前提交的分支，和所有分支  // gitee会告诉我们这次 add了哪些，modify了哪些，delete了哪些，  // add了哪些，modify了哪些，delete了哪些， 自己判断这些修改的文件在哪个模块下  //对这个模块进行持续集成  //Generic Webhook Trigger + script脚本 + 其他已讲过的内容 + 其他自己看看官网 = 搞定  //别人提交一个issue，jenkins触发 自动把issue做成 fix分支，让程序员自己去修改  //所有东西都是模板  script {  //groovy  def where = "${DEPLOY\_WHERE}"  if (where == "bj-01"){  sh "echo 我帮你部署到 bj-01 区了"  }else if(where == "sh-02"){  sh "echo 我帮你部署到 sh-02 区了"  }else{  sh "echo 没人要的，我帮你部署到 wuhan-01 区了"  withCredentials([usernamePassword(credentialsId: 'aliyun-docker-repo', passwordVariable: 'ali\_pwd', usernameVariable: 'ali\_user')]) {  sh "docker login -u ${ali\_user} -p ${ali\_pwd} registry.cn-hangzhou.aliyuncs.com"  sh "docker tag java-devops-demo registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${IMAGE\_VERSION}"  sh "docker push registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${IMAGE\_VERSION}"  }  //ssh 秘钥文件配置到 jenkins 全局秘钥中  withCredentials(ssh){  //ansible 没有  sh "ssh root@xxxx "  //不应该的操作。  sh "远程操作其他机器。。。。"  //k8s集群  //动态切换k8s集群  }  }  }  // sh "docker push registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${APP\_VER}"  }  }    //5、推送报告  stage("发送报告"){  steps {  //短信通知，购买api接口即可  // sh 'curl -i -k -X POST 'https://gyytz.market.alicloudapi.com/sms/smsSend?mobile=mobile&param=\*\*code\*\*%3A12345%2C\*\*minute\*\*%3A5&smsSignId=2e65b1bb3d054466b82f0c9d125465e2&templateId=908e94ccf08b4476ba6c876d13f084ad' -H 'Authorization:APPCODE dddddddd''  //REST API 所有都行  // sh 'curl '  echo '准备发送报告'  emailext body: '''<!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志</title>  </head>  <body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4"  offset="0">  <table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">  <h3>本邮件由系统自动发出，请勿回复！</h3>  <tr>  <br/>  各位同事，大家好，以下为${PROJECT\_NAME }项目构建信息</br>  <td><font color="#CC0000">构建结果 - ${BUILD\_STATUS}</font></td>  </tr>  <tr>  <td><br />  <b><font color="#0B610B">构建信息</font></b>  <hr size="2" width="100%" align="center" /></td>  </tr>  <tr>  <td>  <ul>  <li>项目名称 ： ${PROJECT\_NAME}</li>  <li>构建编号 ： 第${BUILD\_NUMBER}次构建</li>  <li>触发原因： ${CAUSE}</li>  <li>构建状态： ${BUILD\_STATUS}</li>  <li>构建日志： <a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>  <li>构建 Url ： <a href="${BUILD\_URL}">${BUILD\_URL}</a></li>  <li>工作目录 ： <a href="${PROJECT\_URL}ws">${PROJECT\_URL}ws</a></li>  <li>项目 Url ： <a href="${PROJECT\_URL}">${PROJECT\_URL}</a></li>  </ul>  <h4><font color="#0B610B">最近提交</font></h4>  <ul>  <hr size="2" width="100%" />  ${CHANGES\_SINCE\_LAST\_SUCCESS, reverse=true, format="%c", changesFormat="<li>%d [%a] %m</li>"}  </ul>  详细提交: <a href="${PROJECT\_URL}changes">${PROJECT\_URL}changes</a><br/>  </td>  </tr>  </table>  </body>  </html>''', subject: '${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志', to: '15280808129@139.com'  }  }  }  //后置处理过程  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  } |



bj-01 sh-02 只是简单的echo，并没有实际动作，wuhan-01会推送镜像

如选择bj-01





## 8.4 交互优化

### 将推送镜像和选择发布环境分开

|  |
| --- |
| //写流水线的脚本（声明式、脚本式）  pipeline{  //全部的CICD流程都需要在这里定义  //任何一个代理可用就可以执行  // agent none //以后所有stage都必须指定自己的  agent any  //定义一些环境信息  environment {  hello = "123456"  world = "456789"  WS = "${WORKSPACE}"  IMAGE\_VERSION = "v1.0"    //引用Jenkins配置的全局秘钥信息  ALIYUN\_SECRTE=credentials("aliyun-docker-repo")  }  //定义流水线的加工流程  stages {  //流水线的所有阶段  stage('环境检查'){  steps {  sh 'printenv'  echo "正在检测基本信息"  sh 'java -version'  sh 'git --version'  sh 'docker version'  sh 'pwd && ls -alh'  sh "echo $hello"  //未来，凡是需要取变量值的时候，都用双引号  sh 'echo ${world}'  //sh "ssh --help"  }  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  }    stage('maven编译'){  //jenkins不配置任何环境的情况下， 仅适用docker 兼容所有场景  agent {  docker {  image 'maven:3-alpine'  args '-v /var/jenkins\_home/appconfig/maven/.m2:/root/.m2'  // args '-v /a/settings.xml:/app/settings.xml'  //docker run -v /a/settings.xml:/app/settings.xml  }  }  steps {  //git下载来的代码目录下  sh 'pwd && ls -alh'  sh 'mvn -v'  //打包，jar.。默认是从maven中央仓库下载。 jenkins目录+容器目录；-s指定容器内位置  //只要jenkins迁移，不会对我们产生任何影响  sh "echo 默认的工作目录：${WS}"  // sh 'cd ${WS}'  //workdir  //每一行指令都是基于当前环境信息。和上下指令无关  sh 'cd ${WS} && mvn clean package -s "/var/jenkins\_home/appconfig/maven/settings.xml" -Dmaven.test.skip=true '  //jar包推送给maven repo ，nexus  //如何让他适用阿里云镜像源  }  }    //2、测试，每一个 stage的开始，都会重置到默认的WORKSPACE位置  stage('测试'){  steps {  sh 'pwd && ls -alh'  echo "测试..."  }  }  //3、打包  stage('生成镜像'){  steps {  echo "打包..."  //检查Jenkins的docker命令是否能运行  sh 'docker version'  sh 'pwd && ls -alh'  sh 'docker build -t java-devops-demo .'  //镜像就可以进行保存  }  }    stage('推送镜像'){  //没有起容器代理，默认就是jenkins环境  //step里面卡点这么写  // input message: '需要推送远程镜像吗？', ok: '需要', parameters: [text(defaultValue: 'v1.0', description: '生产环境需要部署的版本', name: 'APP\_VER')]  //step外面这么写  input {  message "需要推送远程镜像吗?"  ok "需要"  parameters {  string(name: 'APP\_VER', defaultValue: 'v1.0', description: '生产环境需要部署的版本')  }  }  steps {  echo "$APP\_VER"  script {  withCredentials([usernamePassword(credentialsId: 'aliyun-docker-repo', passwordVariable: 'ali\_pwd', usernameVariable: 'ali\_user')]) {  sh "docker login -u ${ali\_user} -p ${ali\_pwd} registry.cn-hangzhou.aliyuncs.com"  sh "docker tag java-devops-demo registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${APP\_VER}"  sh "docker push registry.cn-hangzhou.aliyuncs.com/mymetaverse/java-devops-demo:${APP\_VER}"  }  }  }  }    stage('选择部署环境'){  //step外面这么写  input {  message "请选择部署环境?"  ok "确定"  parameters {  choice choices: ['bj-01', 'sh-02', 'wuhan-01'], description: '部署的大区', name: 'DEPLOY\_WHERE'  }  }  steps {  script {  //groovy  def where = "${DEPLOY\_WHERE}"  if (where == "bj-01"){  sh "echo 我帮你部署到 bj-01 区了"  }else if(where == "sh-02"){  sh "echo 我帮你部署到 sh-02 区了"  }else{  sh "echo 没人要的，我帮你部署到 wuhan-01 区了"  }  }  }  }    //5、推送报告  stage("发送报告"){  steps {  //短信通知，购买api接口即可  // sh 'curl -i -k -X POST 'https://gyytz.market.alicloudapi.com/sms/smsSend?mobile=mobile&param=\*\*code\*\*%3A12345%2C\*\*minute\*\*%3A5&smsSignId=2e65b1bb3d054466b82f0c9d125465e2&templateId=908e94ccf08b4476ba6c876d13f084ad' -H 'Authorization:APPCODE dddddddd''  //REST API 所有都行  // sh 'curl '  echo '准备发送报告'  emailext body: '''<!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志</title>  </head>  <body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4"  offset="0">  <table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">  <h3>本邮件由系统自动发出，请勿回复！</h3>  <tr>  <br/>  各位同事，大家好，以下为${PROJECT\_NAME }项目构建信息</br>  <td><font color="#CC0000">构建结果 - ${BUILD\_STATUS}</font></td>  </tr>  <tr>  <td><br />  <b><font color="#0B610B">构建信息</font></b>  <hr size="2" width="100%" align="center" /></td>  </tr>  <tr>  <td>  <ul>  <li>项目名称 ： ${PROJECT\_NAME}</li>  <li>构建编号 ： 第${BUILD\_NUMBER}次构建</li>  <li>触发原因： ${CAUSE}</li>  <li>构建状态： ${BUILD\_STATUS}</li>  <li>构建日志： <a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>  <li>构建 Url ： <a href="${BUILD\_URL}">${BUILD\_URL}</a></li>  <li>工作目录 ： <a href="${PROJECT\_URL}ws">${PROJECT\_URL}ws</a></li>  <li>项目 Url ： <a href="${PROJECT\_URL}">${PROJECT\_URL}</a></li>  </ul>  <h4><font color="#0B610B">最近提交</font></h4>  <ul>  <hr size="2" width="100%" />  ${CHANGES\_SINCE\_LAST\_SUCCESS, reverse=true, format="%c", changesFormat="<li>%d [%a] %m</li>"}  </ul>  详细提交: <a href="${PROJECT\_URL}changes">${PROJECT\_URL}changes</a><br/>  </td>  </tr>  </table>  </body>  </html>''', subject: '${ENV, var="JOB\_NAME"}-第${BUILD\_NUMBER}次构建日志', to: '15280808129@139.com'  }  }  }  //后置处理过程  post {  failure {  echo "这个阶段 完蛋了.... $currentBuild.result"  }  success {  echo "这个阶段 成了.... $currentBuild.result"  }  }  } |

### 验证

