CICD流程测试

建立日期：2020年9月1日

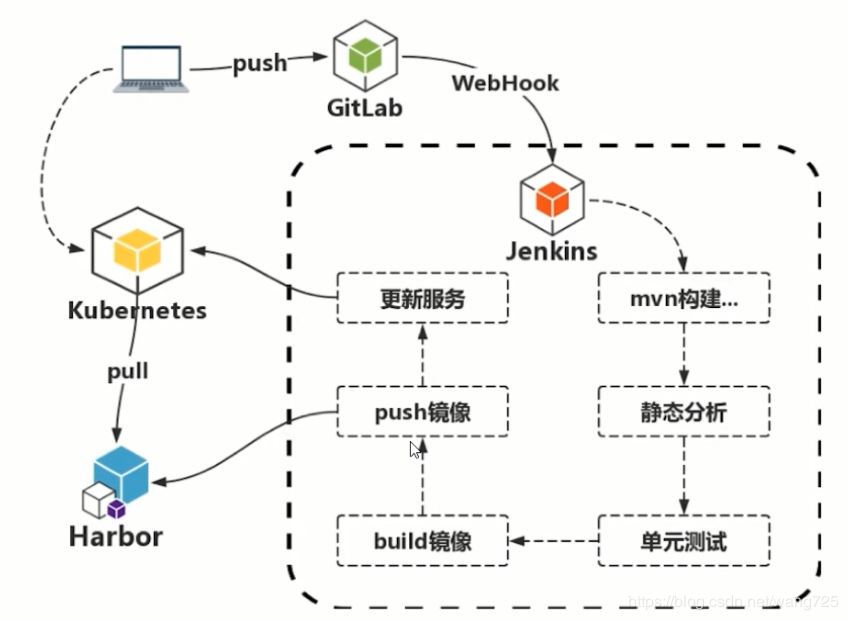
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 变更原因 | 变更内容简述 | 编制/修订者 | 适用环境 | 发布日期 |
| V1.0 | 建立 |  | 陈文华 | 开发、测试 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

CICD（Continuous Integration/Continuous Deployment），持续集成持续部署。CICD流程测试需要Kubernetes集群，Harbor，GitLab和Jenkins等共同完成。

# 一、环境准备

## 1、CICD整体流程

流程图如下：



1、开发者将项目最新的代码提交到GitLab仓库；

2、GitLab的WebHook触发Jenkins构建流程：

（1）拉取项目最新代码进行Maven构建

（2）Maven打包和单元测试

（3）单元测试通过，构建Docker镜像

（4）将构建后的最新镜像推送到镜像仓库Harbor

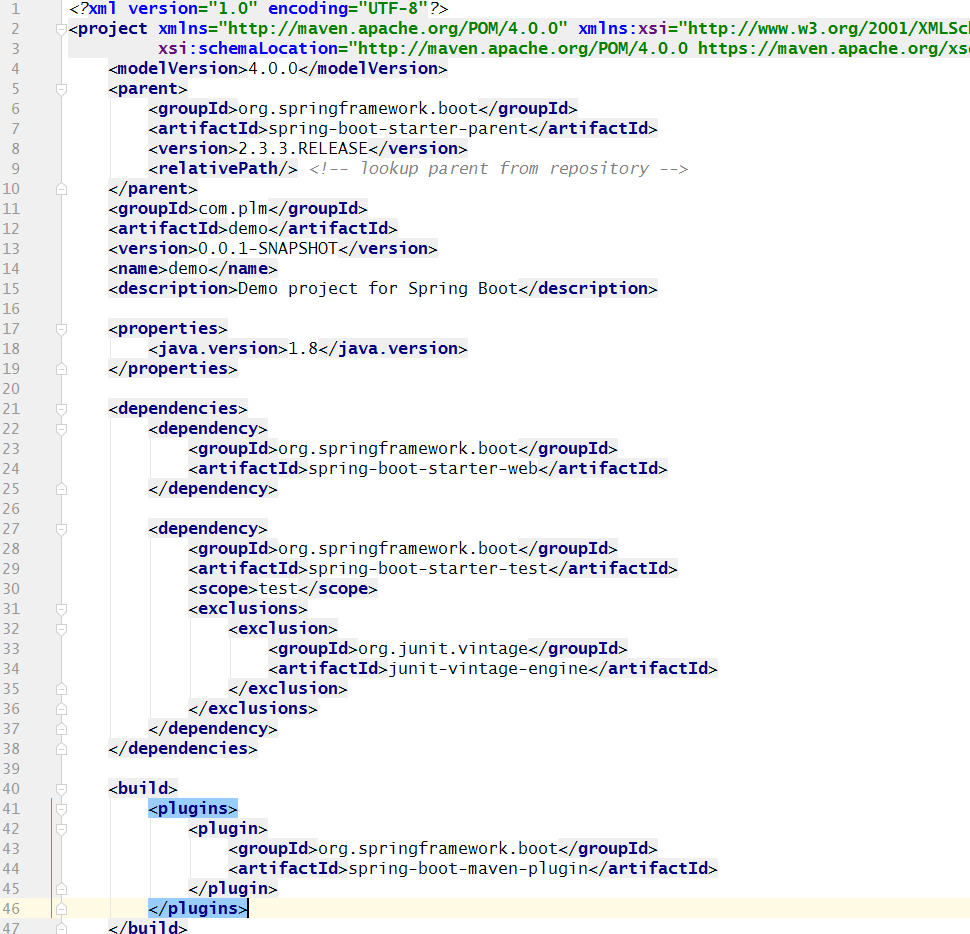
（5）更新Kubernetes镜像版本等相关配置服务

3、 Kubernetes感知到镜像更新，从Harbor拉取最新镜像，进行滚动升级（Rolling Updates）；

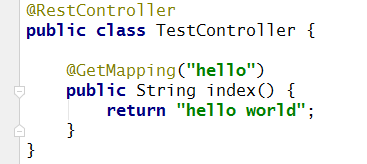
4、项目呈现出最新的代码效果。

## 2、创建测试项目

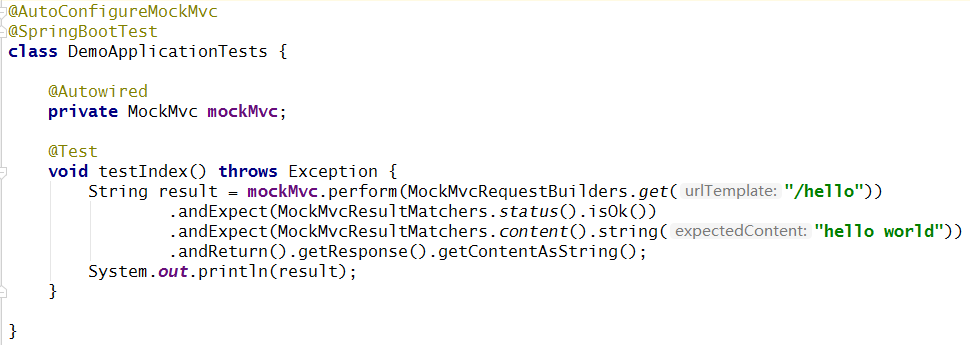
1、创建一个Spring Boot的demo项目用于测试，pom文件内容如下



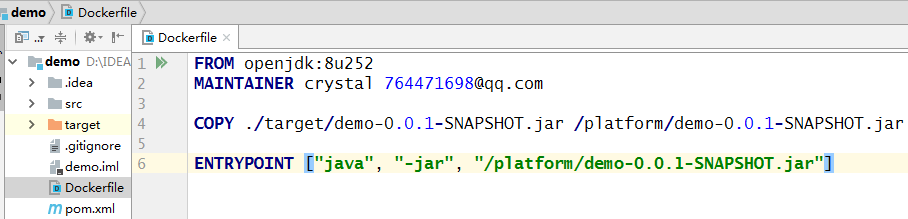
2、新建controller包，包下新建TestController.java并添加一个测试用接口



3、编写一个单元测试

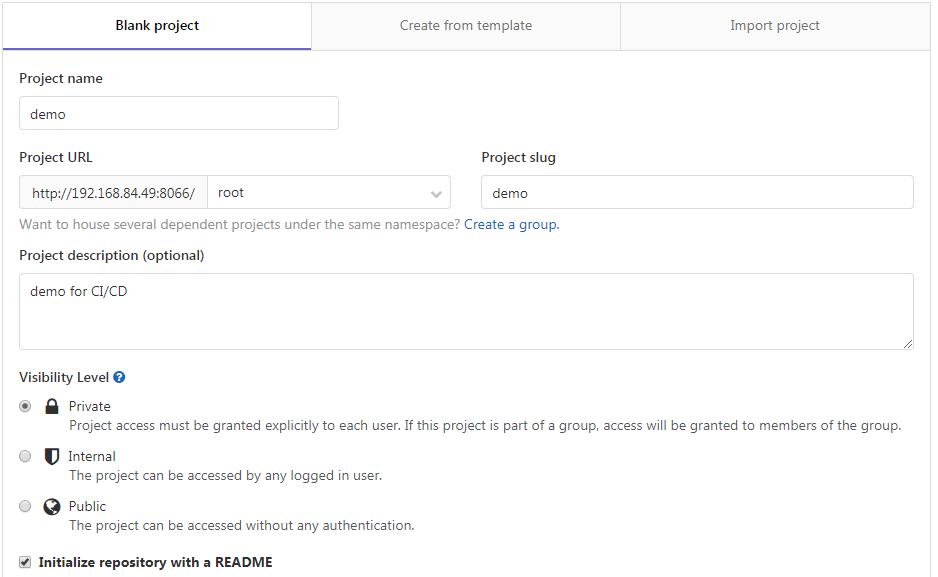


4、创建Dockerfile



## 3、GitLab项目准备

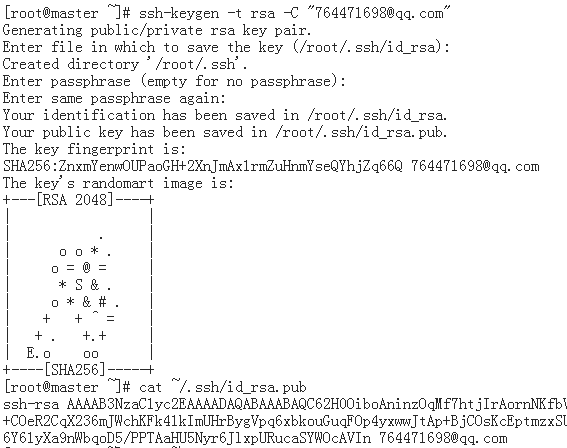
1、登录GitLab，点击Create a project新建项目



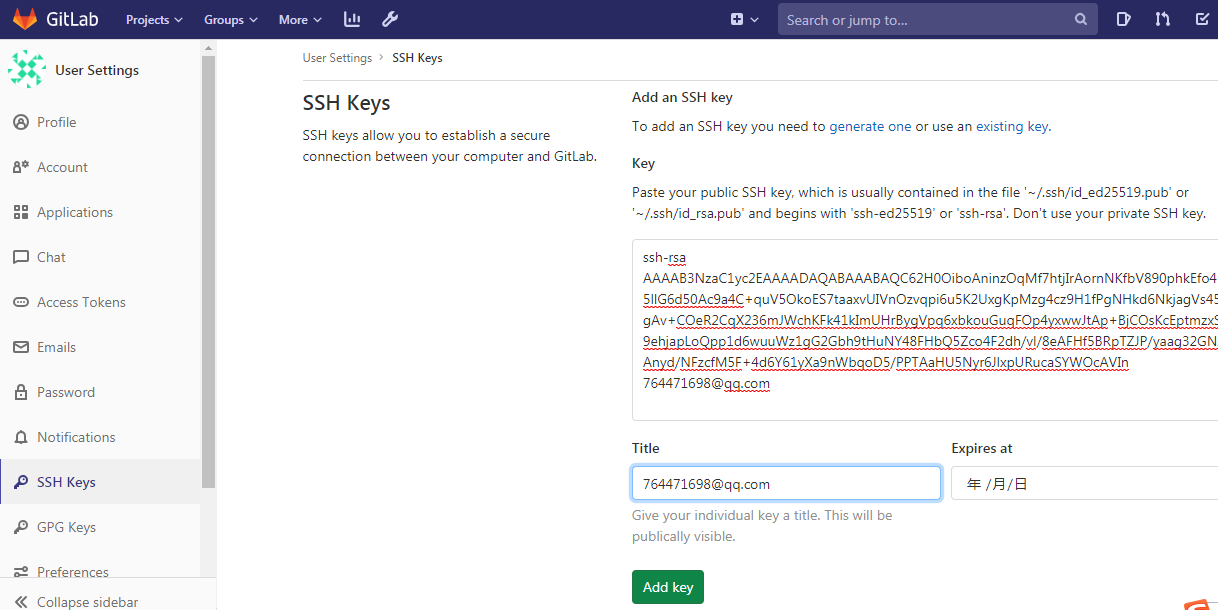
2、在192.168.84.49的master服务器执行以下命令生成SSH Key（敲三次回车）

ssh-keygen -t rsa -C "764471698@qq.com"

cat ~/.ssh/id\_rsa.pub

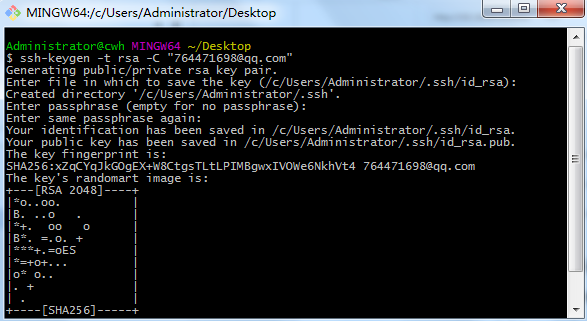


3、在GitLab的User Settings -> SSH Keys添加生成的秘钥

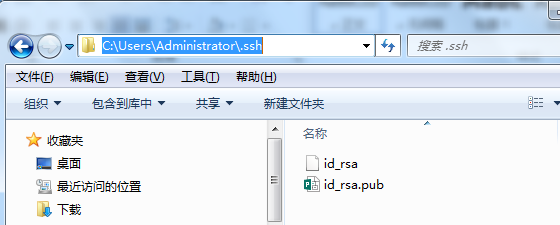


4、windows下载git客户端，地址<https://git-scm.com/downloads>

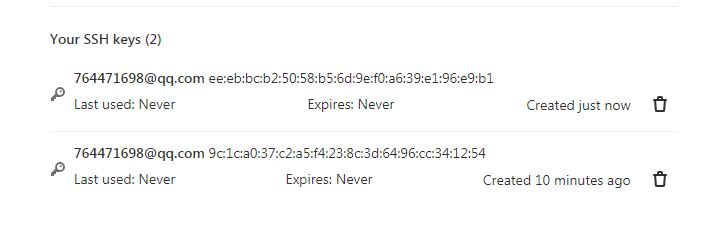
下载安装完成后，右键选择Git Bash Here，执行命令生成SSH Key



在C:\Users\Administrator\.ssh目录下有一对密钥文件



将id\_rsa.pub的内容添加至GitLab的SSH Keys



5、打开C:\Windows\System32\drivers\etc的hosts文件添加本地域名解析

192.168.84.49 gitlab.oumasoft.com

6、在Terminal窗口中执行命令将demo项目推送GitLab

# 配置git，设置你的用户名、邮箱

git config --global user.name "yourname"

git config --global user.email "youremail "

# 初始化git仓库

git init

# 提交代码

git add .

git commit –m init

# 添加远程仓库

git remote add origin ssh://git@gitlab.oumasoft.com:2266/root/demo.git

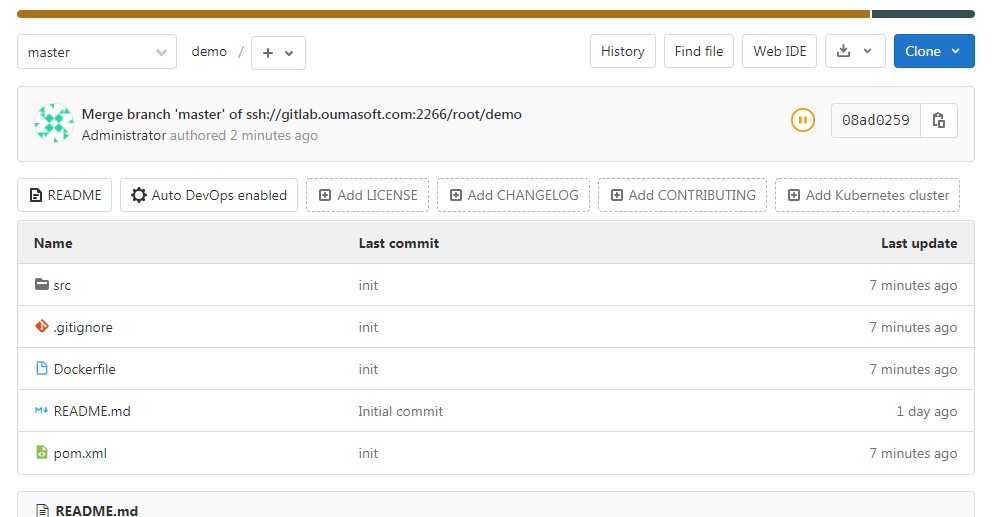
# 先拉取最新代码

git pull origin master --allow-unrelated-histories

# 推送到GitLab

git push origin master

7、在GitLab上查看项目推送结果



如果git add . 出现The file will have its original line endings in your working directory.的提示。这是因为文件中换行符的差别导致的。这个提示的意思是说：会把windows格式（CRLF（也就是回车换行））转换成Unix格式（LF），这些是转换文件格式的警告，不影响使用。git默认支持LF。windows commit代码时git会把CRLF转LF，update代码时LF换CRLF。可以执行以下命令：

git rm -r --cached .

git config core.autocrlf false

git add .

## 4、Kubernetes部署

1、克隆项目

cd /home/

git clone ssh://git@gitlab.oumasoft.com:2266/root/demo.git

2、下载并配置maven

# 下载安装包

wget http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz

# 解压缩，conf/settings.xml添加阿里云镜像

tar -xzvf apache-maven-3.6.3-bin.tar.gz

# 修改环境变量

vim /etc/profile

添加如下内容：

M2\_HOME=/home/apache-maven-3.6.3

export PATH=$PATH:$JAVA\_HOME/bin:$M2\_HOME/bin

# 使配置生效

source /etc/profile

# 验证安装



3、构建镜像，推送至Harbor

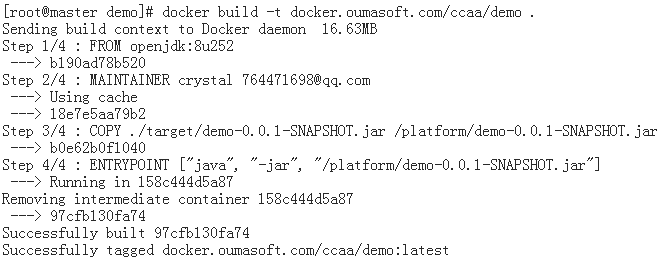
# 项目打包

cd demo/

mvn clean package

# 构建Docker镜像

docker build -t docker.oumasoft.com/ccaa/demo .

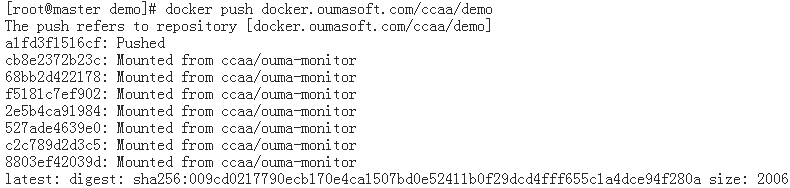


# 登录Harbor

docker login docker.oumasoft.com

# 推送镜像

docker push docker.oumasoft.com/ccaa/demo



到Harbor中查看镜像：



4、创建kubernetes资源

vim demo.yml

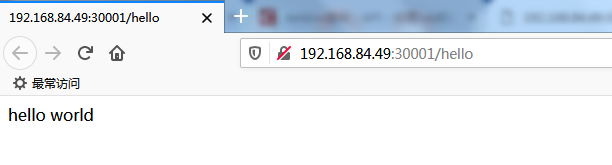
内容如下:

**apiVersion:** apps/v1  
**kind:** Deployment  
**metadata:  
 name:** demo-deployment  
**spec:  
 selector:  
 matchLabels:  
 app:** demo  
 **replicas:** 2  
 **template:  
 metadata:  
 labels:  
 app:** demo  
 **spec:  
 containers:** - **name:** demo  
 **image:** docker.oumasoft.com/ccaa/demo  
 **ports:** - **containerPort:** 8080  
 **resources:  
 requests:  
 cpu:** 200m  
 **memory:** 200Mi  
 **limits:  
 cpu:** 500m  
 **memory:** 500Mi  
---  
**apiVersion:** v1  
**kind:** Service  
**metadata:  
 name:** demo-service  
**spec:  
 type:** NodePort  
 **ports:** - **port:** 8080  
 **targetPort:** 8080  
 **nodePort:** 30001  
 **selector:  
 app:** demo

创建该资源

kubectl apply -f demo.yml

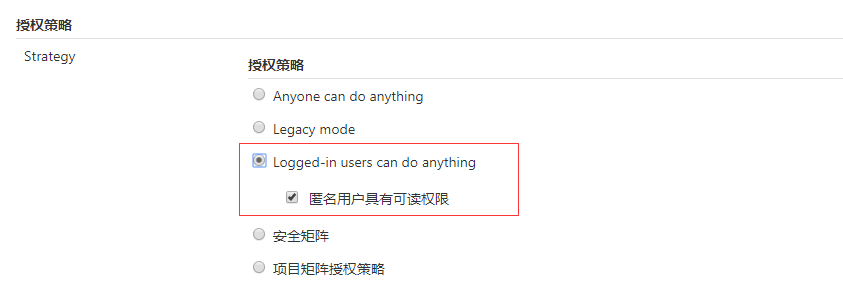
浏览器访问 <http://192.168.84.49:30001/hello>



# 二、CI/CD流程测试

## 1、配置WebHook

1、登录Jenkins，选择[Manage Jenkins](http://192.168.84.49:8067/manage) -> Configure Global Security

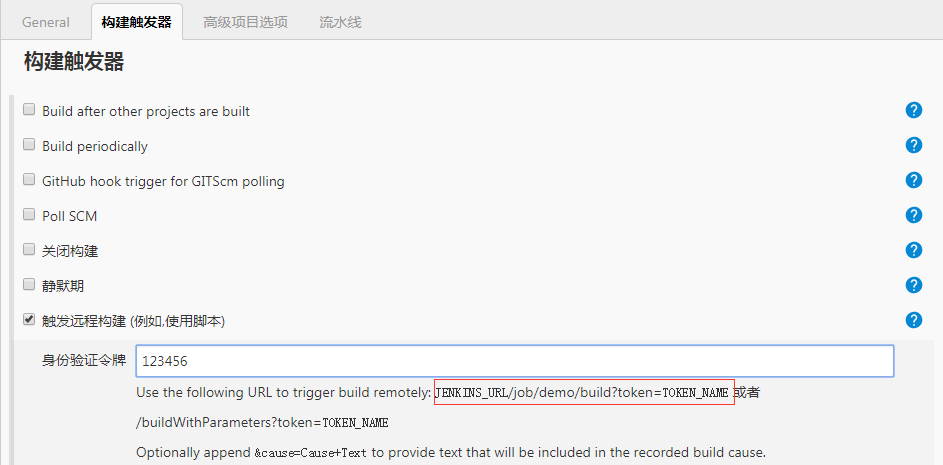


开启匿名用户具有可读权限。

2、新建Item，输入任务名称demo，选择流水线点击确定



构建触发器勾选“触发远程构建”

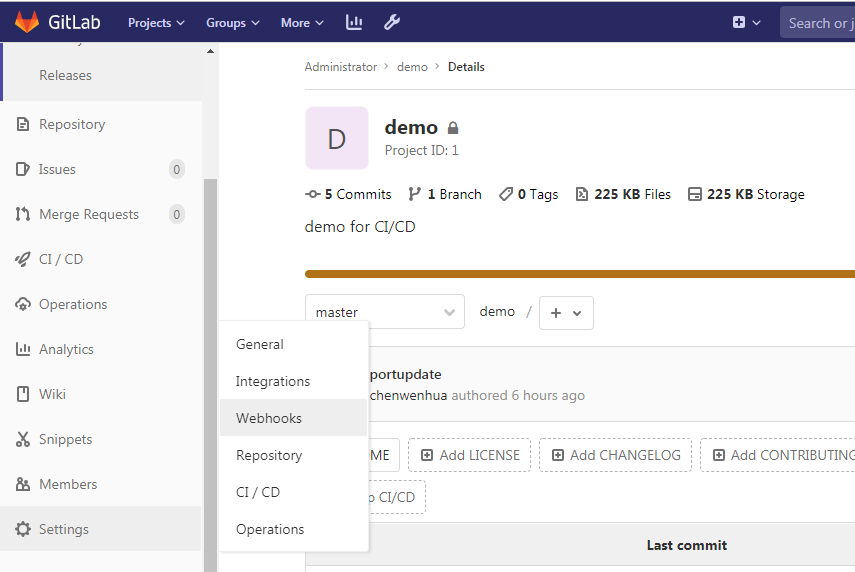


设置身份验证令牌为123456，将触发地址

JENKINS\_URL/job/demo/build?token=TOKEN\_NAME

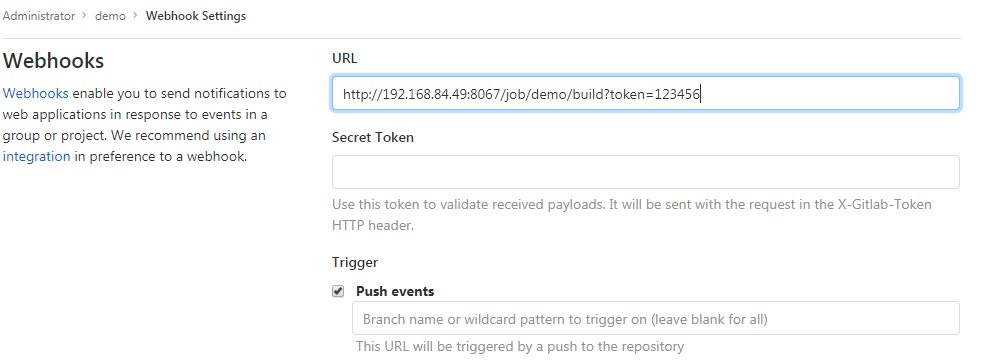
配置到GitLab Webhooks

3、打开demo项目，选择Settings -> Webhooks



URL填写http://192.168.84.49:8067/job/demo/build?token=123456

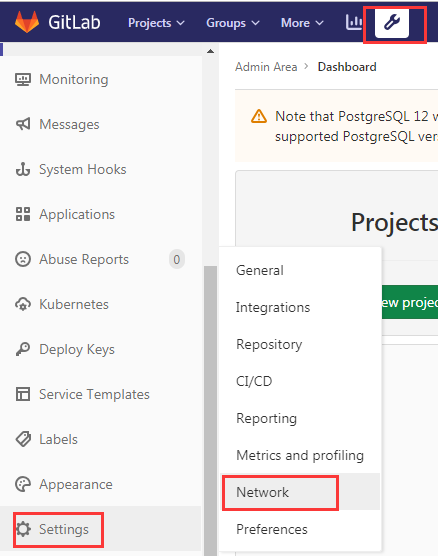
Trigger选择Push events



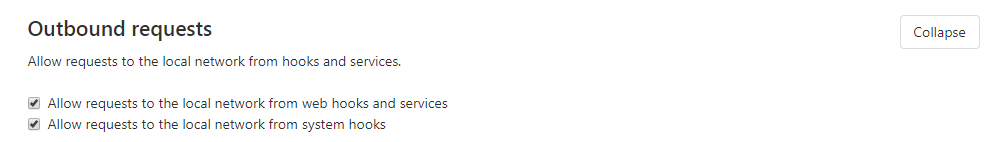
然后点击Add webhook，出错了，错误信息如下



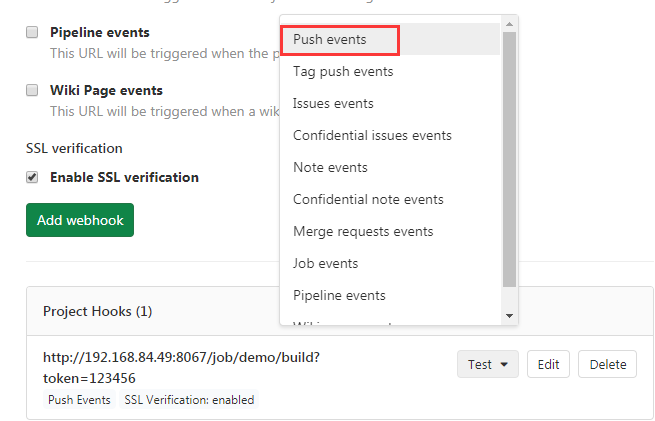
使用admin账号登录Gitlab，选择Admin Area->Settings->Network



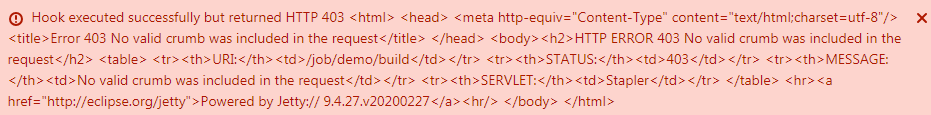
勾选Allow requests to the local network from web hooks and services



重新配置Webhooks，测试



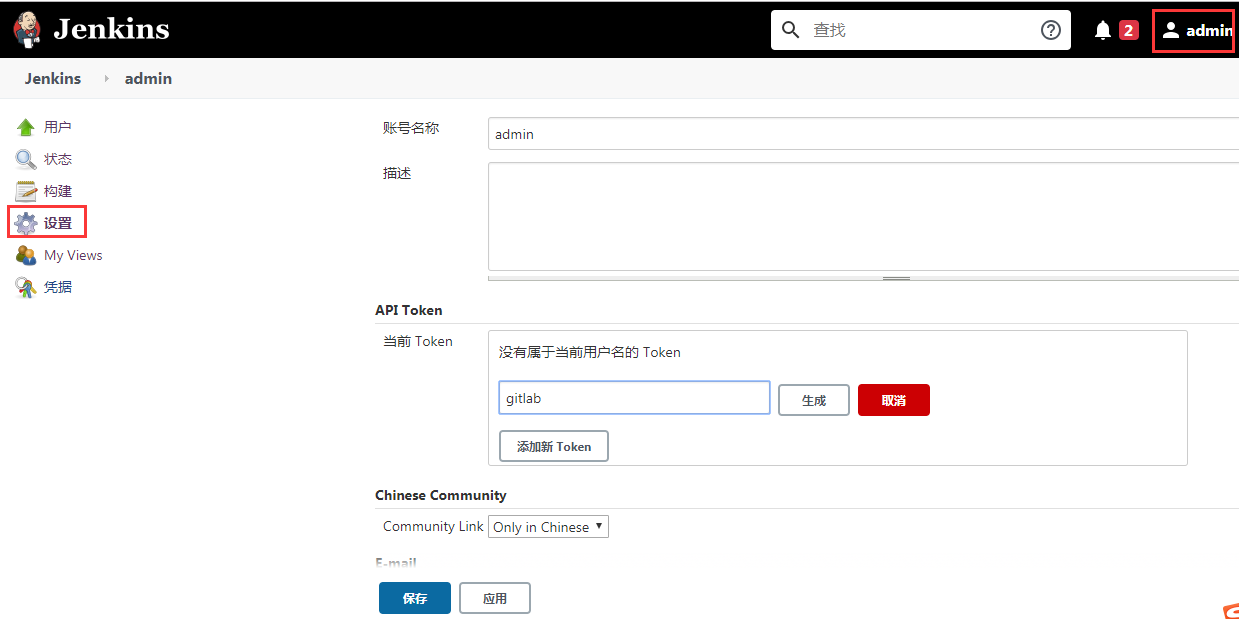
报错信息：



解决方法：（1）以下命令获取Jenkins crumb

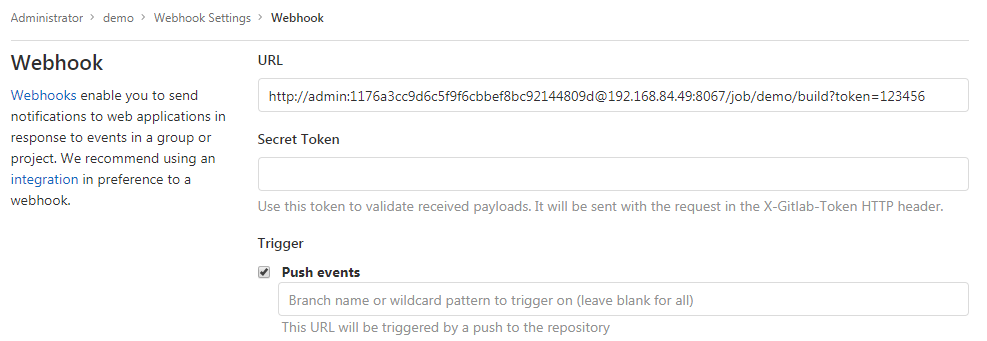
curl -u yourname:password 'http://192.168.84.49:8067/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'

（2）点击用户名->设置->API Token，添加一个Token



点击“生成”按钮，复制生成的Token，点击“保存”。回到Gitlab Webhook添加

http://yourname:APIToken@192.168.84.49:8067/job/demo/build?token=123456



再次测试，出现以下信息则配置完成。



## 2、Jenkins拉取代码

1、点击在Jenkins上创建的demo流水线，点击“配置”，编辑Pipeline script，添加如下脚本：



pipeline {

agent any

environment {

REPOSITORY="ssh://git@gitlab.oumasoft.com:2266/root/demo.git"

}

stages {

stage('代码拉取') {

steps {

echo "从${REPOSITORY}地址拉取代码"

deleteDir()

git "${REPOSITORY}"

}

}

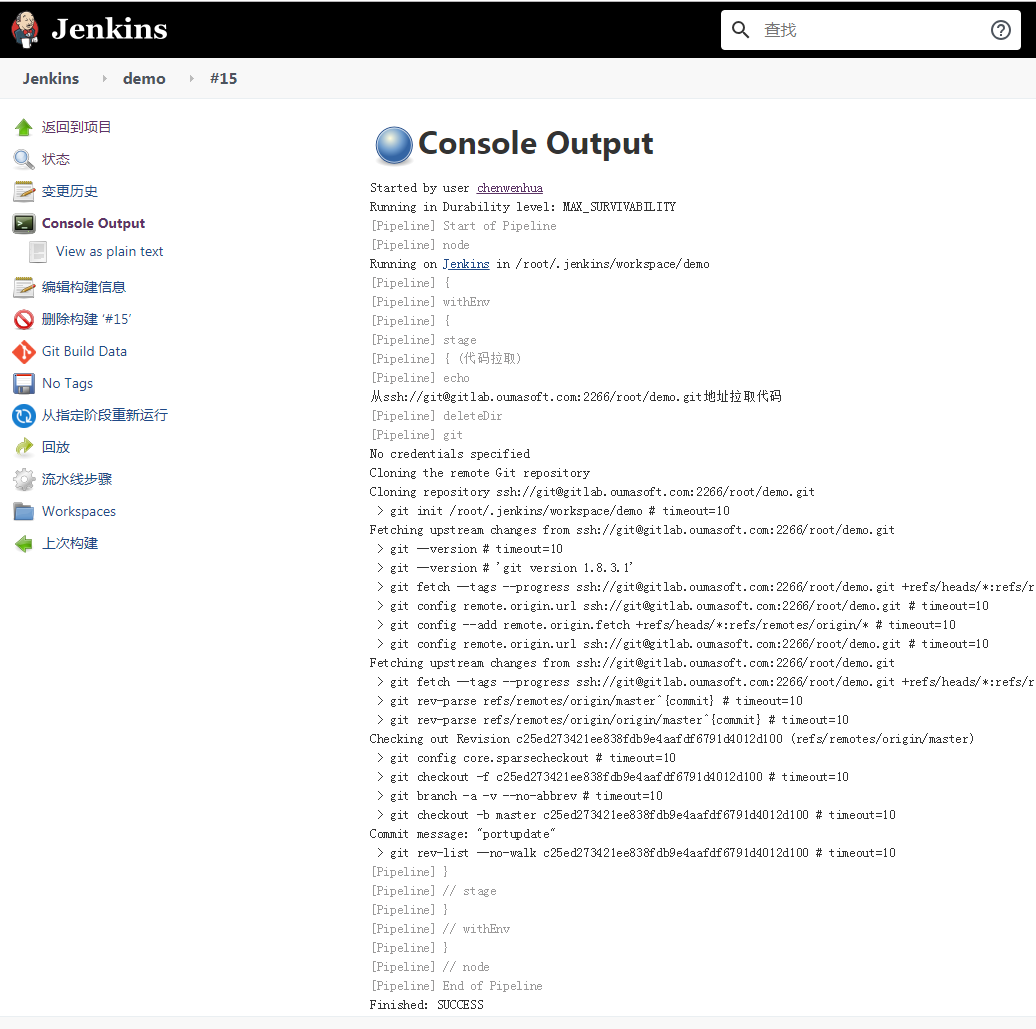
}

}

environment定义地址变量，deleteDir()清空demo文件夹目录，git "${REPOSITORY}"拉取代码。具体的模板语法可点击“流水线语法”来查看。例如从git拉取代码：



2、保存执行脚本后，点击“Build Now”可以查看阶段视图，点开Build History链接查看Console Output



控制台输出成功，查看代码是否已经拉取：



## 3、项目maven打包

1、Jenkins从/usr/bin文件夹中调用maven命令，执行以下命令创建软连接

ln -s /home/apache-maven-3.6.3/bin/mvn /usr/bin/mvn

或者在Global Tool Configuration中配置maven添加MAVEN\_HOME的路径

/home/apache-maven-3.6.3

2、修改流水线Pipeline script脚本，增加maven打包的阶段



脚本：

pipeline {

agent any

environment {

REPOSITORY="ssh://git@gitlab.oumasoft.com:2266/root/demo.git"

}

stages {

stage('拉取代码') {

steps {

echo "从GitLab地址${REPOSITORY}拉取代码"

deleteDir()

git "${REPOSITORY}"

}

}

stage('项目打包 单元测试') {

steps {

echo "开始打包"

sh "mvn -U -am clean package"

}

}

}

}

-am表示同时处理选定模块所依赖的模块

-U参数： 该参数能强制让Maven检查所有SNAPSHOT依赖更新，确保集成基于最新的状态。

3、保存后点击“Build Now”，查看日志。

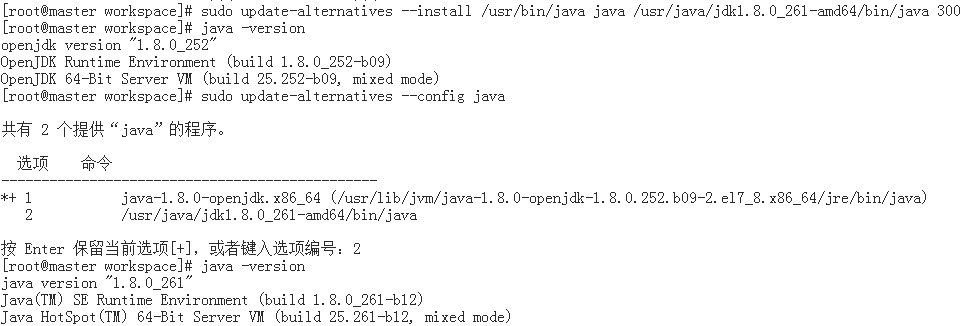
如果打包过程出错

No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK

因为docker运行了一个JDK环境没有JRE，切换成服务器安装的JDK

sudo update-alternatives --install /usr/bin/java java /usr/java/jdk1.8.0\_261-amd64/bin/java 300

sudo update-alternatives --config java



## 4、构建、推送镜像

1、在/home/demo文件夹下创建执行脚本

vim build\_push.sh

内容如下

#!/bin/bash

TIME=`date "+%Y%m%d%H%M"`

GIT\_REVISION=`git log -1 --pretty=format:"%h"`

IMAGE\_NAME=docker.oumasoft.com/ccaa/demo:${TIME}\_${GIT\_REVISION}

docker build -t ${IMAGE\_NAME} .

docker push ${IMAGE\_NAME}

echo "${IMAGE\_NAME}" > image\_name

打包推送后的镜像名称输出到了当前目录下的image\_name文件中

2、添加执行权限：

chmod +x build\_push.sh

格式化sh文件防止执行出错

dos2unix \*\*.sh

3、添加构建并推送Docker镜像的stage

pipeline {

agent any

environment {

REPOSITORY="ssh://git@gitlab.oumasoft.com:2266/root/demo.git"

SCRIPT\_PATH="/home/demo"

}

stages {

stage('拉取代码') {

steps {

echo "从GitLab地址${REPOSITORY}拉取代码"

deleteDir()

git "${REPOSITORY}"

}

}

stage('项目打包') {

steps {

echo "开始打包"

sh "mvn -U -am clean package"

}

}

stage('构建并推送Docker镜像') {

steps {

echo "开始构建Docker镜像并推送到Harbor"

sh "${SCRIPT\_PATH}/build\_push.sh"

}

}

}

}

4、查看镜像是否已经推送到Harbor



## 5、Kubernetes升级镜像

1、在/home/demo文件夹下创建执行脚本

vim deployment.sh

内容如下：

#!/bin/bash

IMAGE=`cat image\_name`

kubectl set image deployment/demo-deployment demo=${IMAGE}

2、添加执行权限：

chmod +x deployment.sh

格式化sh文件防止执行出错

dos2unix \*\*.sh

3、修改Pipeline script，添加stage

pipeline {

agent any

environment {

REPOSITORY="ssh://git@gitlab.oumasoft.com:2266/root/demo.git"

SCRIPT\_PATH="/home/demo"

}

stages {

stage('拉取代码') {

steps {

echo "从GitLab地址${REPOSITORY}拉取代码"

deleteDir()

git "${REPOSITORY}"

}

}

stage('项目打包') {

steps {

echo "开始打包"

sh "mvn -U -am clean package"

}

}

stage('构建并推送Docker镜像') {

steps {

echo "开始构建Docker镜像并推送到Harbor"

sh "${SCRIPT\_PATH}/build\_push.sh"

}

}

stage('Kubernetes升级Deployment') {

steps {

echo "开始升级Deployment"

sh "${SCRIPT\_PATH}/deployment.sh"

}

}

}

}

4、保存后点击“Build Now”，查看deployment，Image已经更新

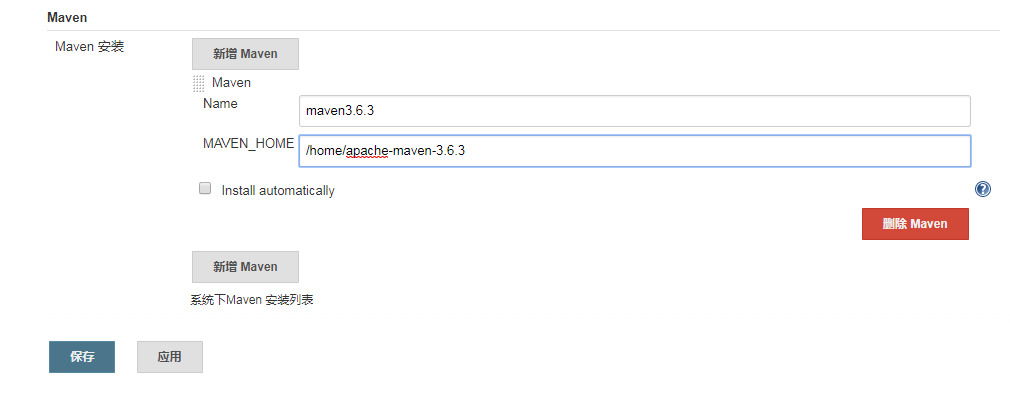


PS：

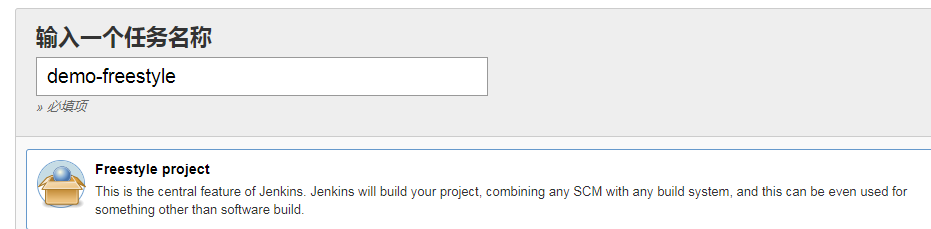
实测发现Jenkins经常在maven打包和执行sh命令的步骤中卡住，控制台没有报错信息，但是会一直转圈无法停止。尝试了卸载重装、安装不同的稳定版本、修改系统设置等方式都没有解决。我在阿里云服务器上安装配置的Jenkins则没有这个问题。可能是公司的网络环境导致某些插件下载出错造成的。如果出现这个问题，我们来创建一个Freestyle project或者maven项目。

# 三、Jenkins创建Freestyle project

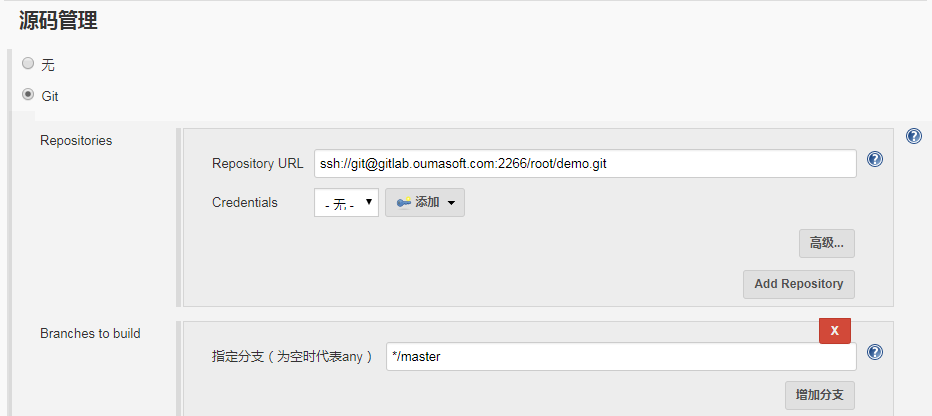
如果构建maven项目，需要在Global Tool Configuration中先添加maven配置



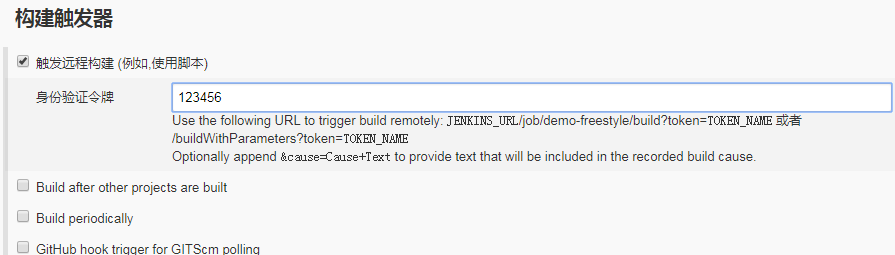
1. 以创建Freestyle project类型的任务为例



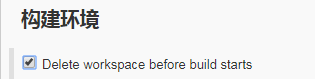
1. 源码管理选择git，填写demo项目的gitlab地址



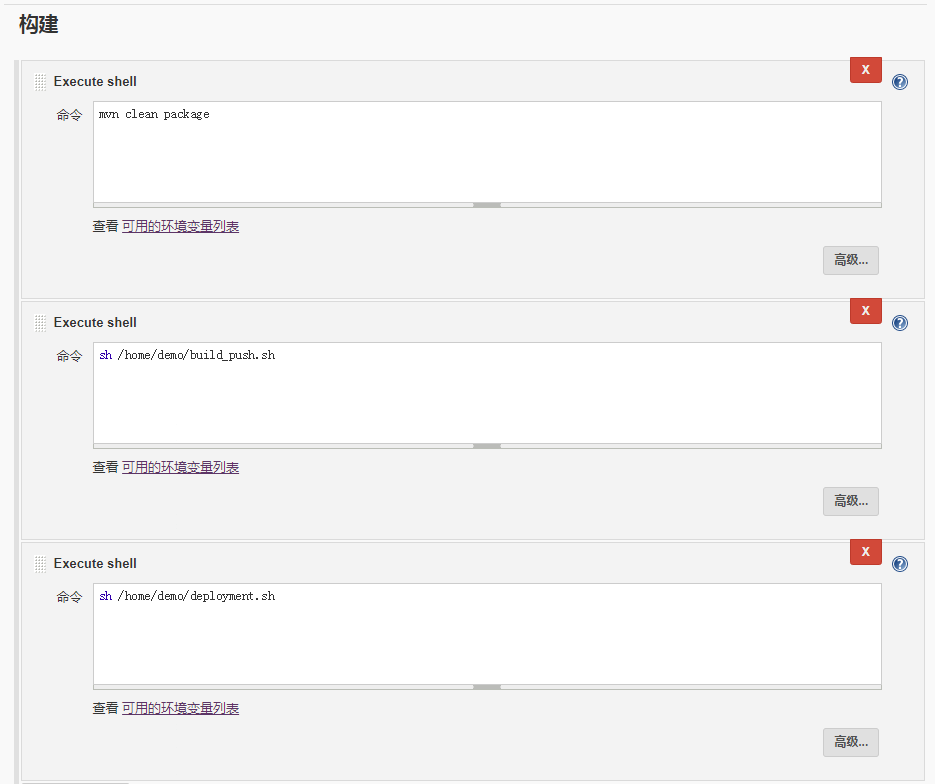
1. 构建触发器添加身份令牌



4、构建环境勾选Delete workspace before build starts

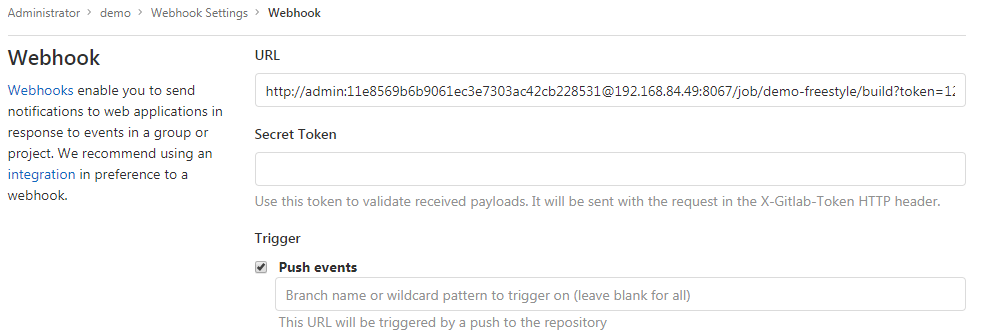


5、增加构建步骤，选择Execute shell



即执行maven打包命令，构建推送镜像脚本，kubernetes更新deployment脚本

6、到gitlab中删除原来的webhook，添加刚刚创建的触发器

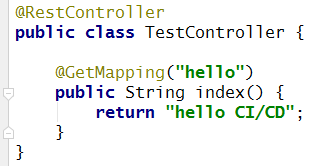


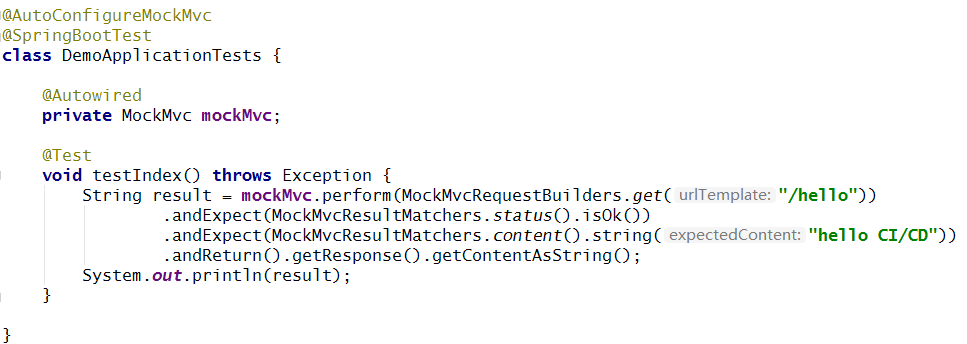
7、测试Push events，到Jenkins控制台查看日志，所有步骤正常运行，出现

Finished: SUCCESS提示信息

# 四、测试CI/CD整体效果

1、修改项目的TestController和单元测试





2、提交代码到gitlab

git add .

git commit -m test

git push origin master

1. 等待一会，访问 <http://192.168.84.49:30001/hello>

