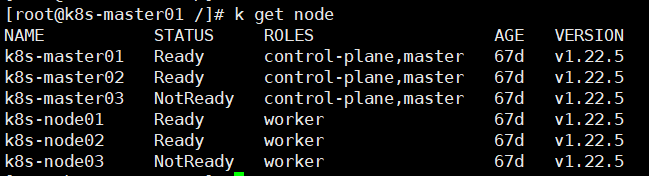
# 基于K8S和Jenkins的CI/CD

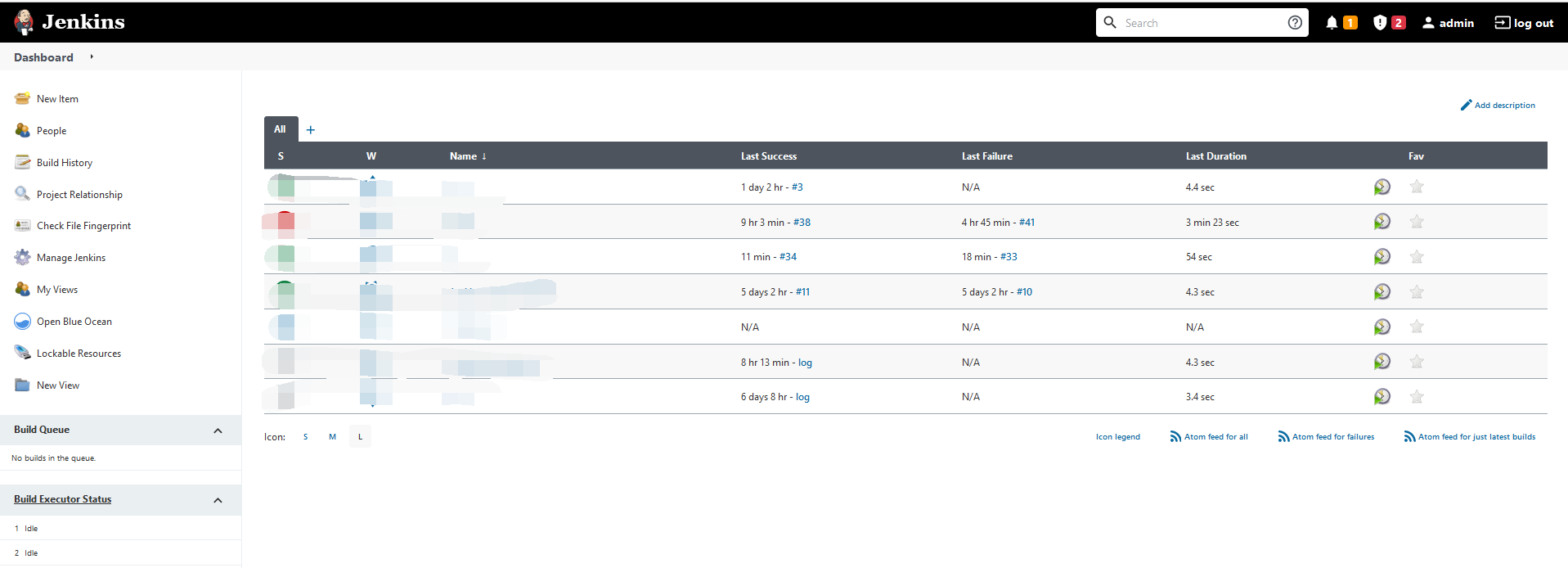
## **测试环境**

测试环境包括：K8S 集群，Jenkins, Harbor, GitLab，均为本地搭建环境，非云环境，提前完成所有环境集成。

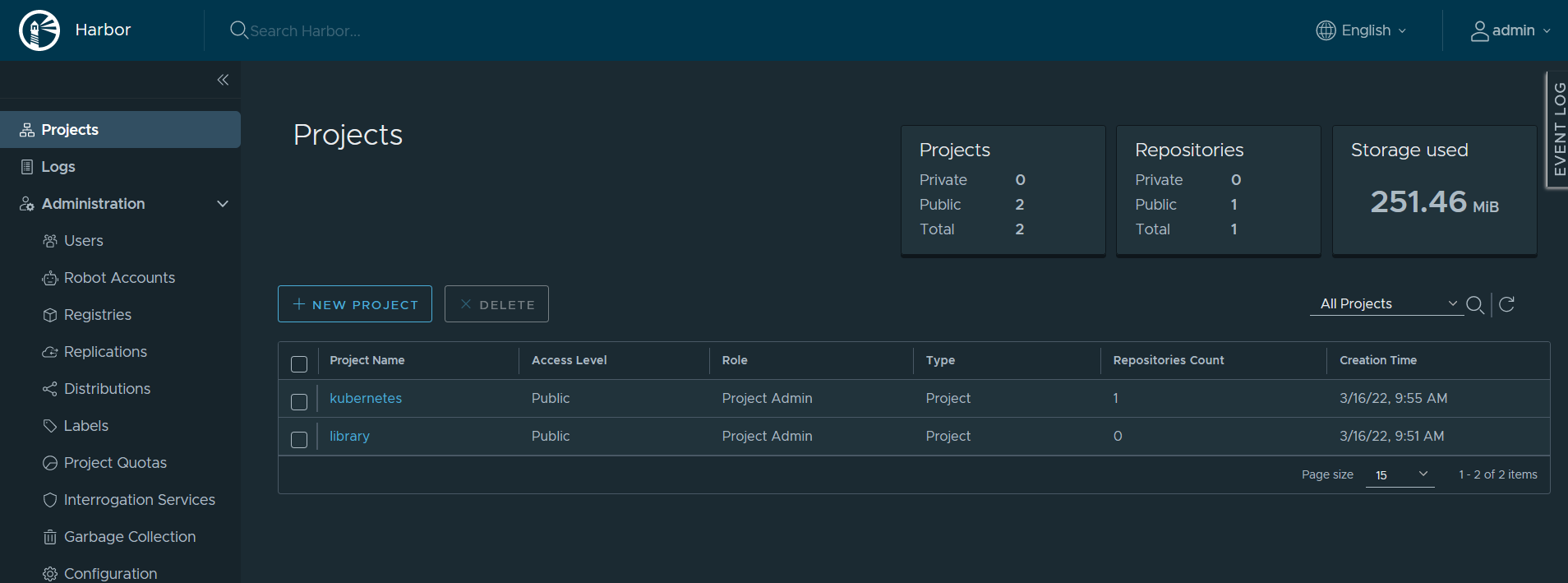
A． K8S集群环境



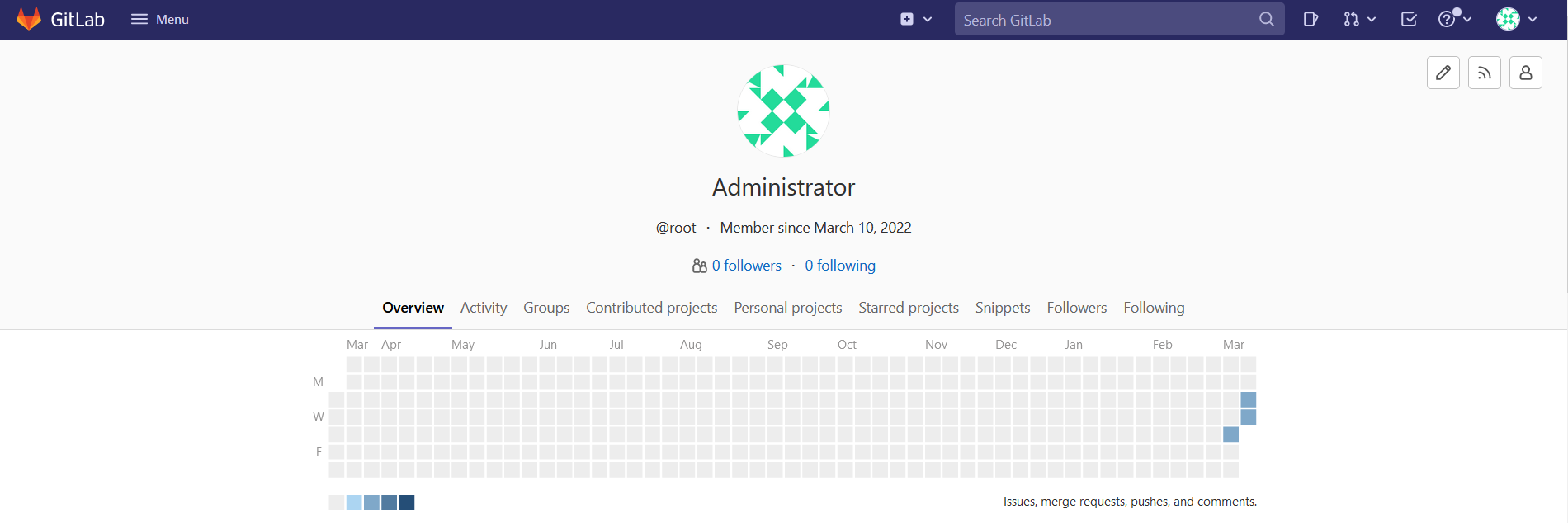
B． Jenkins



C. Harbor



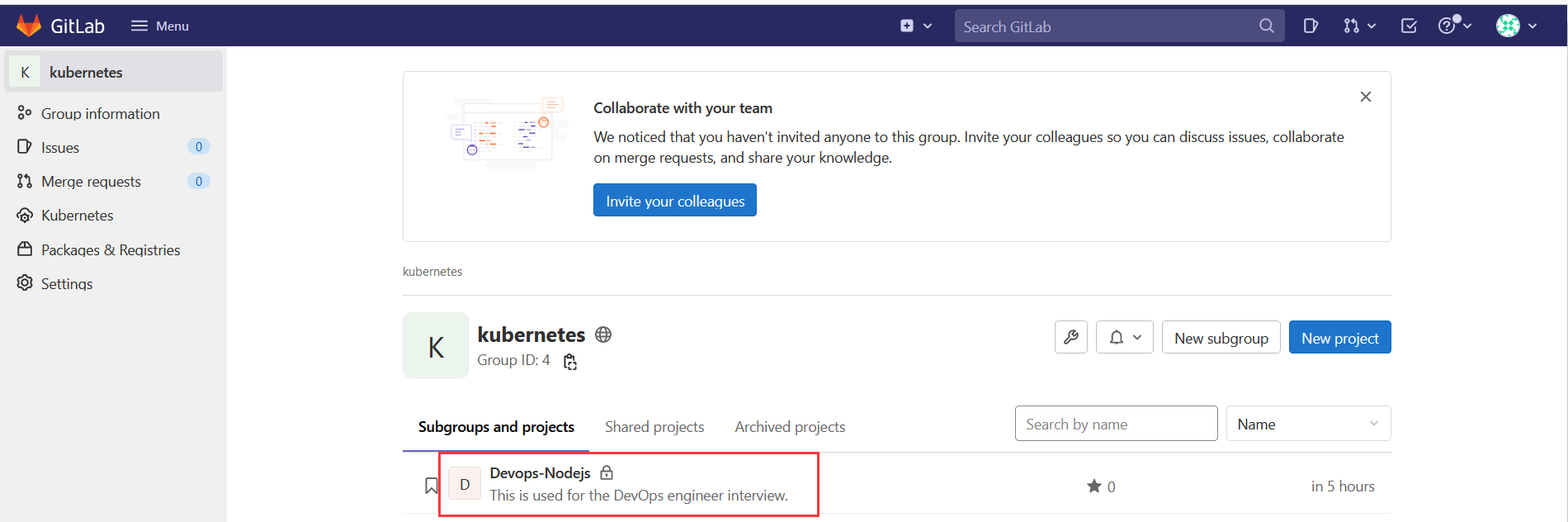
D . GitLab



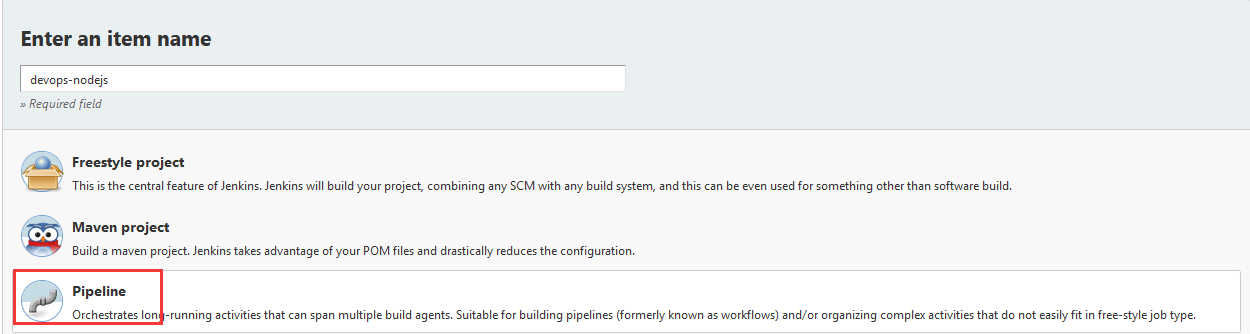
## **2 . CI/CD自动化部署**

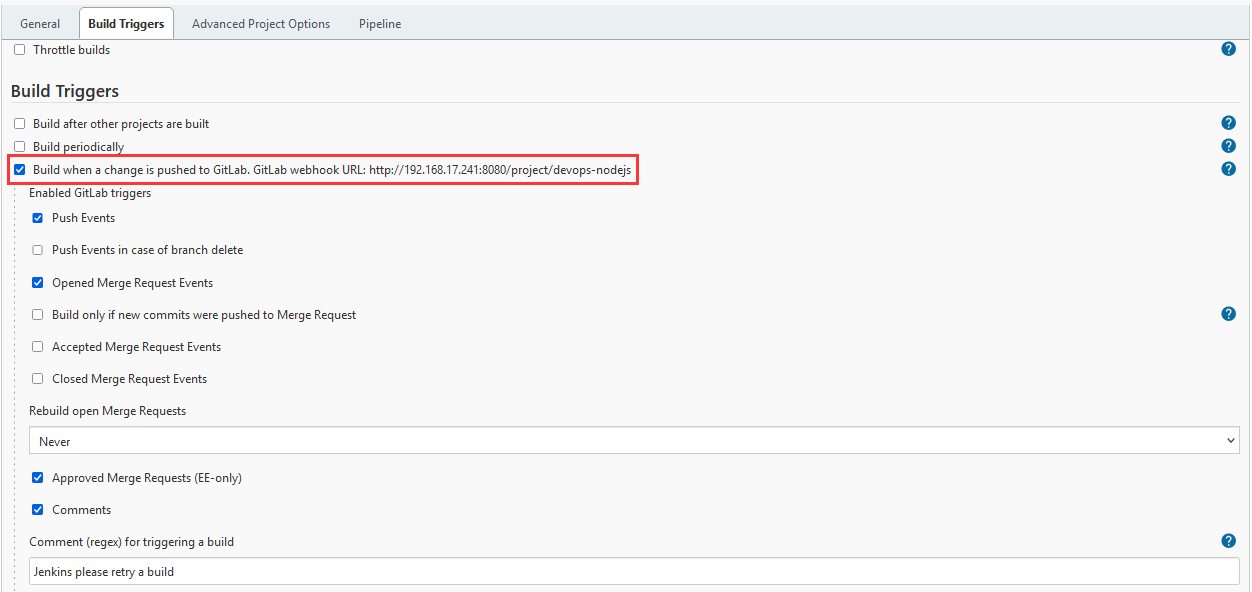
### **2.1 创建项目**

* 在GitLab创建项目，名称为Devops-Nodejs

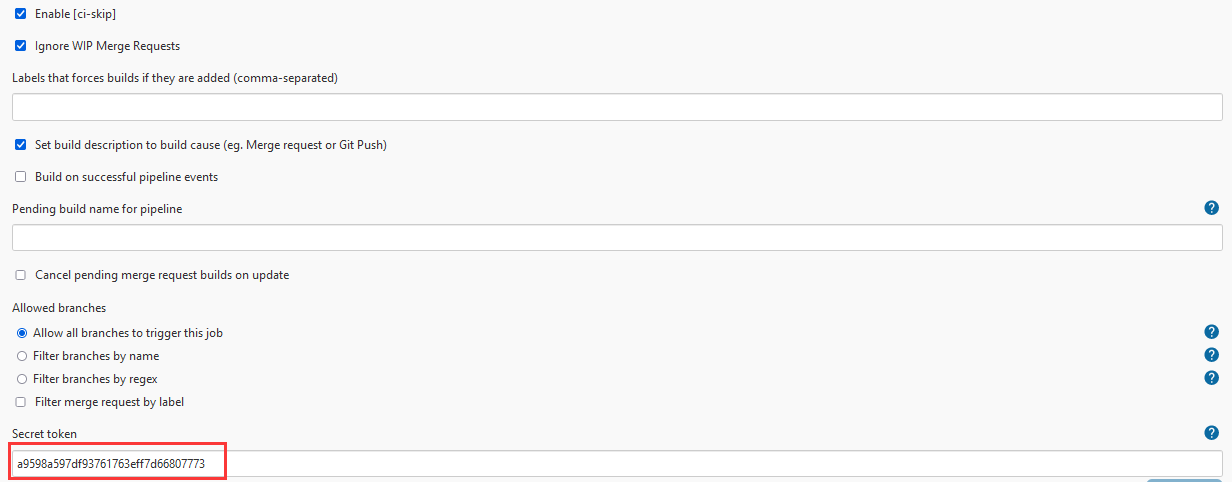


* Jenkins创建Pipeline, Dashboard->New Item, 选择Pipeline类型

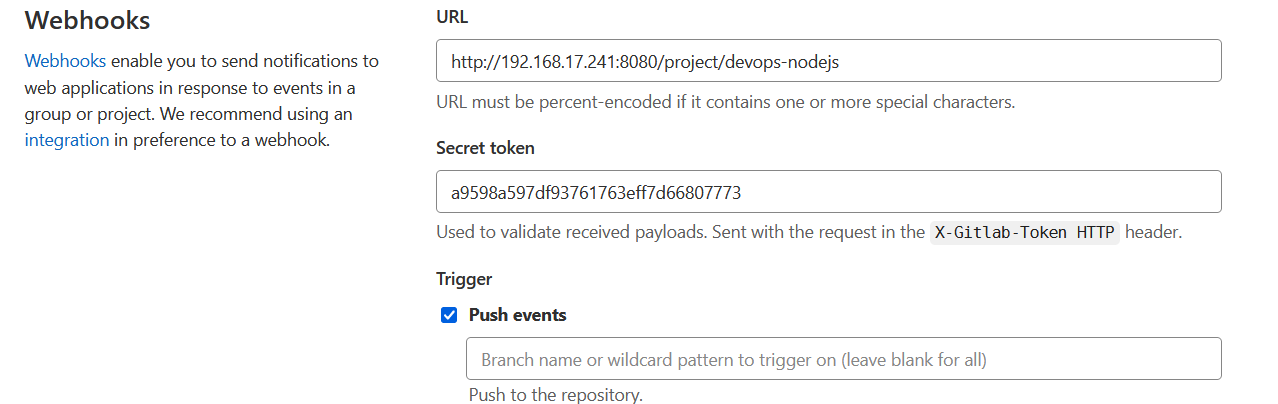




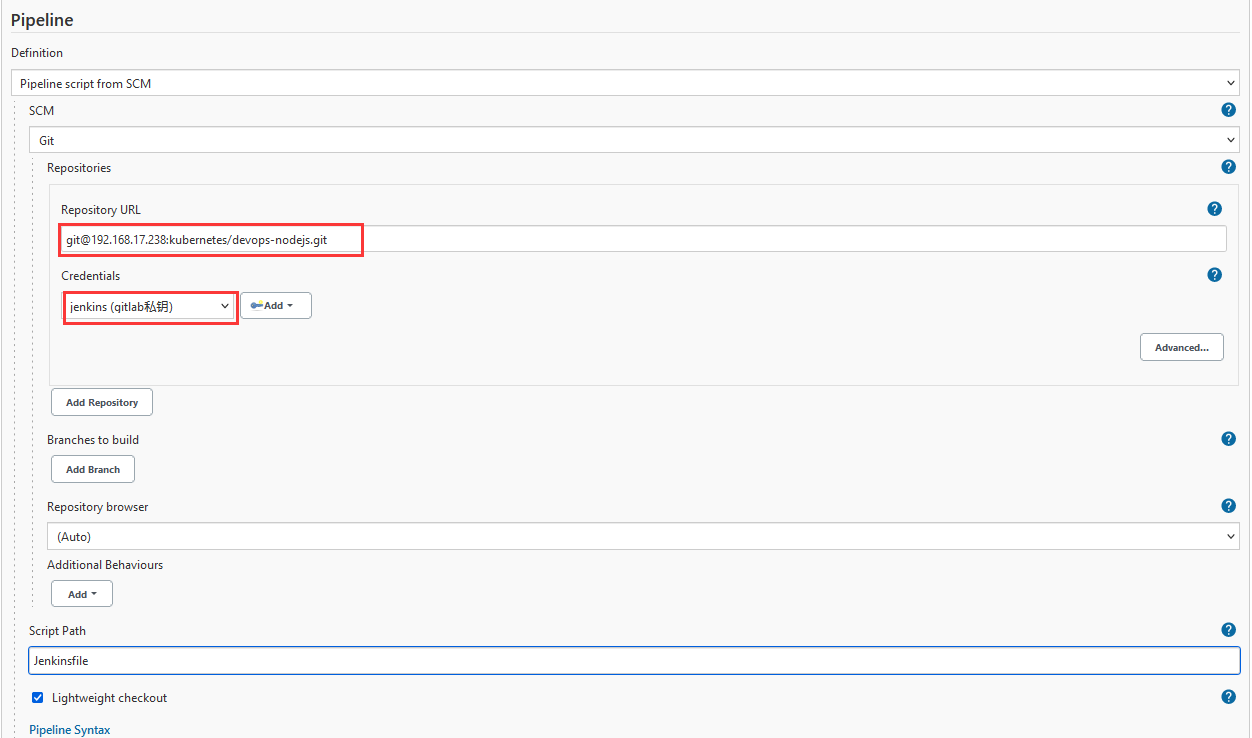
* 创建Secret token,用于与webhooks集成

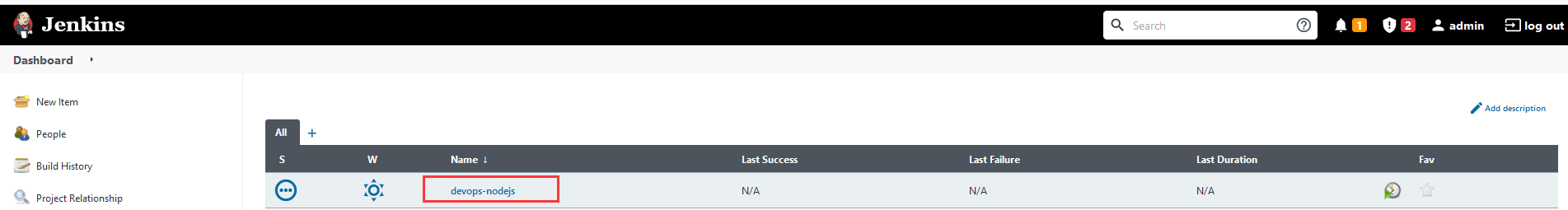


* 将Jenkins生成URL和Secret token添加到对应项目的webhooks



* 通过SCM的方式获取gitlab的项目





### **2.2 编写CI/CD自动化脚本**

* 编写Dockerfile和Jenkinsfile脚本文件，并上传到Devops-Nodejs

Dockerfile和app.js用于创建项目image, Jenkinsfile用于实现项目的自动部署

Dockerfile详解

FROM node:7 //基础镜像

ADD app.js /app.js //将app.js复制并解压到/app.js

ENTRYPOINT ["node", "app.js"]//运行

Jenkinsfile详解

pipeline{

//Define global environment variables

environment {

COMMIT\_ID = ""

HARBOR\_ADDRESS = "192.168.17.10"

REGISTRY\_DIR = "kubernetes"

IMAGE\_NAME = "devops-nodejs"

NAMESPACE = "kubernetes"

TAG = ""

}

//Using fitParameter plugin to build with parameter in Jenkins

parameters{

gitParameter(branch:'',branchFilter: 'origin/(.\*)', description:'Branch for build and deployment', defaultValue: 'master', name: 'BRANCH', quickFilterEnabled: false, selectedValue: 'NONE', sortMode: 'NONE', tagFilter: '\*',type: 'PT\_BRANCH')

}

agent {

//K8S as the agent, container jnlp is used for Jenkins agent, container docker is used for creating images, container kubectl is used for deploy project to k8s

kubernetes {

cloud 'kubernetes-POC'

slaveConnectTimeout 1200

yaml '''

apiVersion: v1

kind: Pod

spec:

containers:

-

name: "jnlp"

image: "registry.cn-beijing.aliyuncs.com/citools/jnlp:alpine"

args: [\'$(JENKINS\_SECRET)\',\'$(JENKINS\_NAME)\']

imagePullPolicy: IfNotPresent

tty: true

volumeMounts:

-

mountPath: "/etc/localtime"

name: "localtime"

readOnly: false

-

name: "docker"

image: "docker:latest"

imagePullPolicy: "IfNotPresent"

tty: true

volumeMounts:

-

mountPath: "/etc/localtime"

name: "localtime"

readOnly: false

-

mountPath: "/var/run/docker.sock"

name: "dockersocker"

readOnly: false

command:

- 'cat'

env:

-

name: "LANGUAGE"

value: "en\_US.en"

-

name: "LC\_ALL"

value: "en\_US.UTF-8"

-

name: "LANG"

value: "en\_US.UTF-8"

-

name: "kubectl"

image: "registry.cn-beijing.aliyuncs.com/citools/kubectl:self-1.17"

imagePullPolicy: "IfNotPresent"

tty: true

volumeMounts:

-

mountPath: "/etc/localtime"

name: "localtime"

readOnly: false

command:

- 'cat'

env:

-

name: "LANGUAGE"

value: "en\_US.en"

-

name: "LC\_ALL"

value: "en\_US.UTF-8"

-

name: "LANG"

value: "en\_US.UTF-8"

restartPolicy: "Never"

nodeSelector:

build: "true"

volumes:

-

name: "cachedir"

hostPath:

path: "/opt/m2"

-

name: "dockersocker"

hostPath:

path: "/var/run/docker.sock"

-

name: "localtime"

hostPath:

path: "/usr/share/zoneinfo/Asia/Shanghai"

'''

}

}

stages{

stage("Pulling code"){

//parallel includes 2 stages, you can choose trigger the pipeline in Jenkins or webhooks(gitlab)

parallel{

stage("Pulling Code by Jenkins"){

when {

expression{

//if you use Jenkins to trigger the pipeline, env.gitlabBranch will be null

env.gitlabBranch == null

}

}

steps{

git(url: 'git@192.168.17.238:kubernetes/devops-nodejs.git', branch: "${BRANCH}", changelog: true, poll: true, credentialsId: 'gitlab-key')

script{COMMIT\_ID = sh(returnStdout: true, script: "git log -n 1 --pretty=format:'%h'").trim()

TAG = BUILD\_TAG + '-' + COMMIT\_ID

println "Current branch is ${BRANCH}, Commit ID is ${COMMIT\_ID}, Image TAG is ${TAG}"

}

}

}

stage("Pulling Code by webhooks"){

when{

expression{

//if you use webhooks to trigger the pipeline it will be gitBranchName.

env.gitlabBranch !=null

}

}

steps{

git(url: 'git@192.168.17.238:kubernetes/devops-nodejs.git', branch: env.gitlabBranch, changelog: true, poll: true, credentialsId: 'gitlab-key')

script{COMMIT\_ID = sh(returnStdout: true, script: "git log -n 1 --pretty=format:'%h'").trim()

TAG = BUILD\_TAG + '-' + COMMIT\_ID

println "Current branch is ${env.gitlabBranch}, Commit ID is ${COMMIT\_ID}, Image TAG is ${TAG}"

}

}

}

}

}

//Create image and push to Harbor

stage("Docker build for creating image"){

environment{

HARBOR\_USER = credentials('Harbor')

}

steps{

//Using DOCKER container to create the images and push into to Harbor.

container(name: "docker"){

sh """

echo ${HARBOR\_USER\_USR} ${HARBOR\_USER\_PSW} ${TAG}

echo ${HARBOR\_USER\_PSW} >> ~/HARBOR\_USER\_PSW.txt

docker build -t ${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${TAG} .

docker login --username ${HARBOR\_USER\_USR} --password-stdin < ~/HARBOR\_USER\_PSW.txt ${HARBOR\_ADDRESS}

docker push ${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${TAG}

"""

}

}

}

//Deploy application to K8S

stage("Deploy to K8S"){

environment{

MY\_KUBECONFIG = credentials('kuberbetes-POC-admin')

}

steps{

container(name: 'kubectl'){

sh """

/usr/local/bin/kubectl set image deployment -l app=${IMAGE\_NAME} ${IMAGE\_NAME}=${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${TAG} -n $NAMESPACE --kubeconfig $MY\_KUBECONFIG

"""

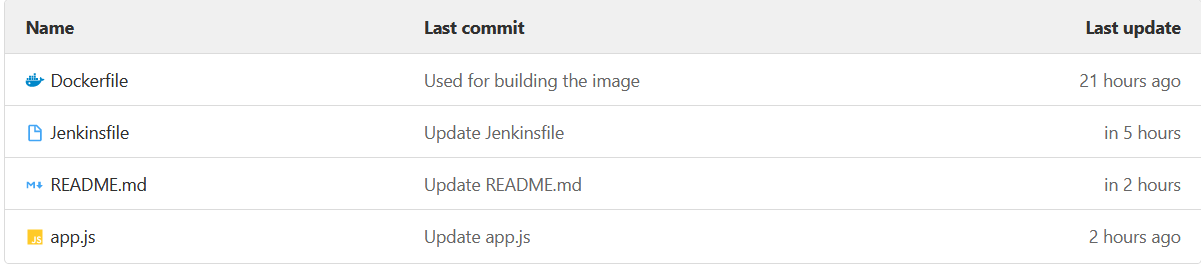
}

}

}

}

}



### **2.3 部署和测试**

* 前置环境

在通过CI/CD完成项目自动化部署前，在k8s提前将部署了应用，image是使用nodejs:v2，我们通过CI/CD完成部署后，image会随之改变，那我们就完成了自动化的部署。

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: devops-nodejs

name: devops-nodejs

namespace: kubernetes

spec:

replicas: 1

selector:

matchLabels:

app: devops-nodejs

strategy: {}

template:

metadata:

labels:

app: devops-nodejs

spec:

imagePullSecrets:

-

name: harborkey

containers:

-

image: nodejs:v2

name: devops-nodejs

imagePullPolicy: IfNotPresent

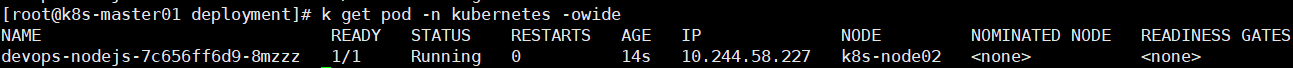
ports:

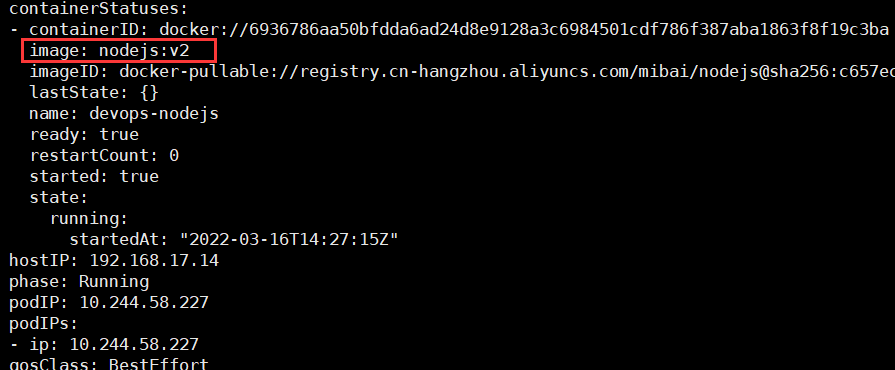
-

containerPort: 8080

name: web

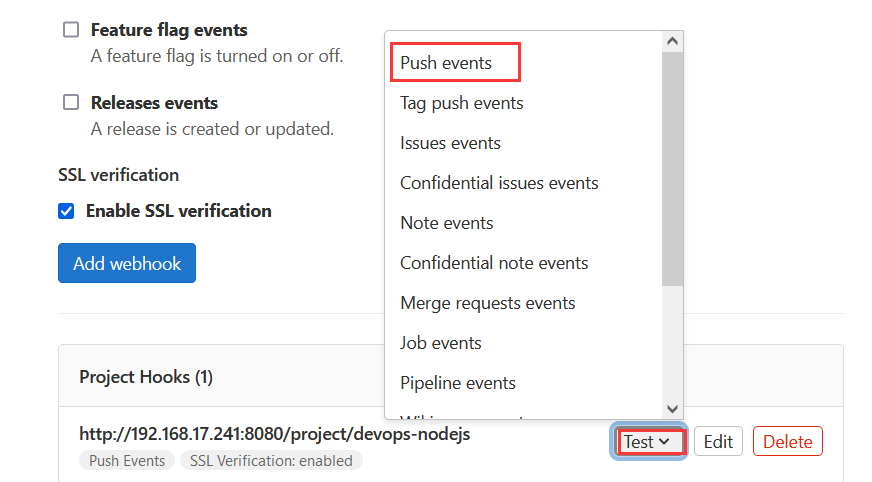
当前应用已经正常运行

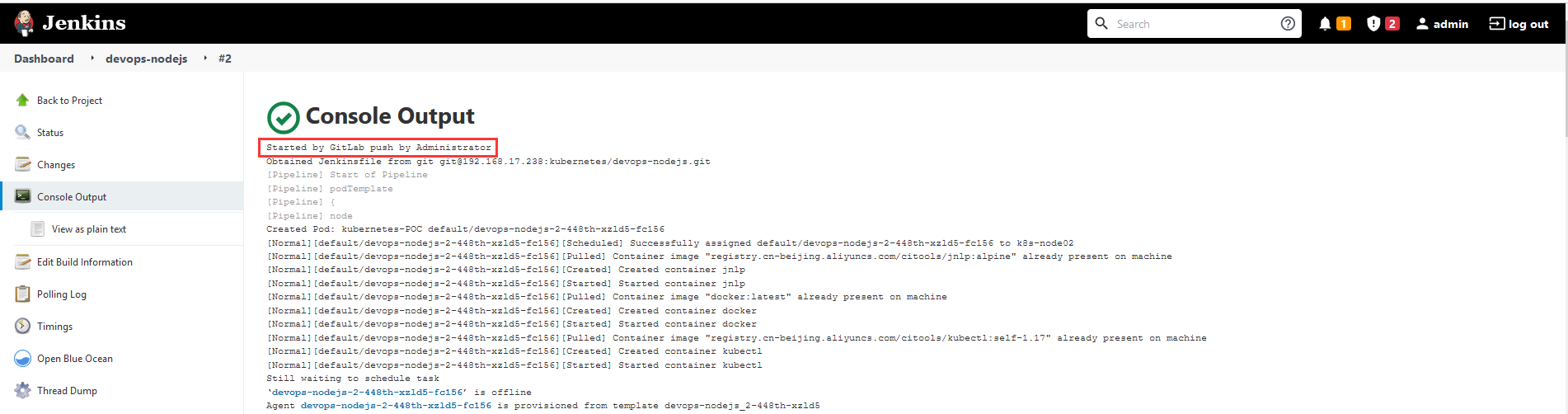


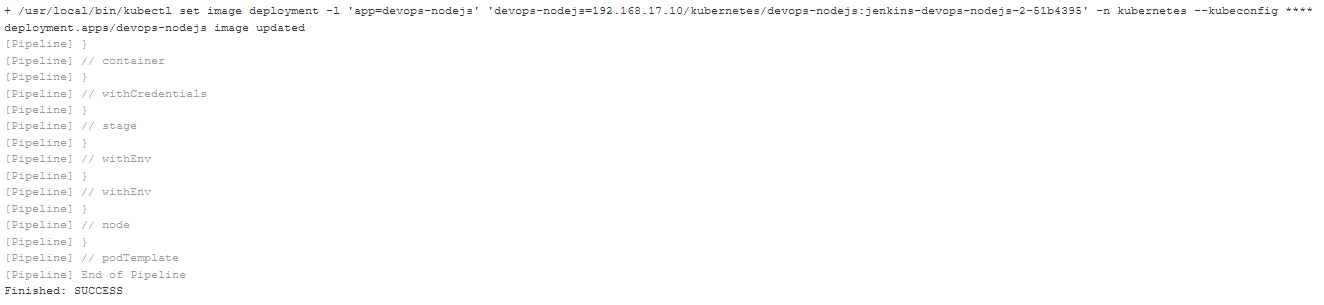


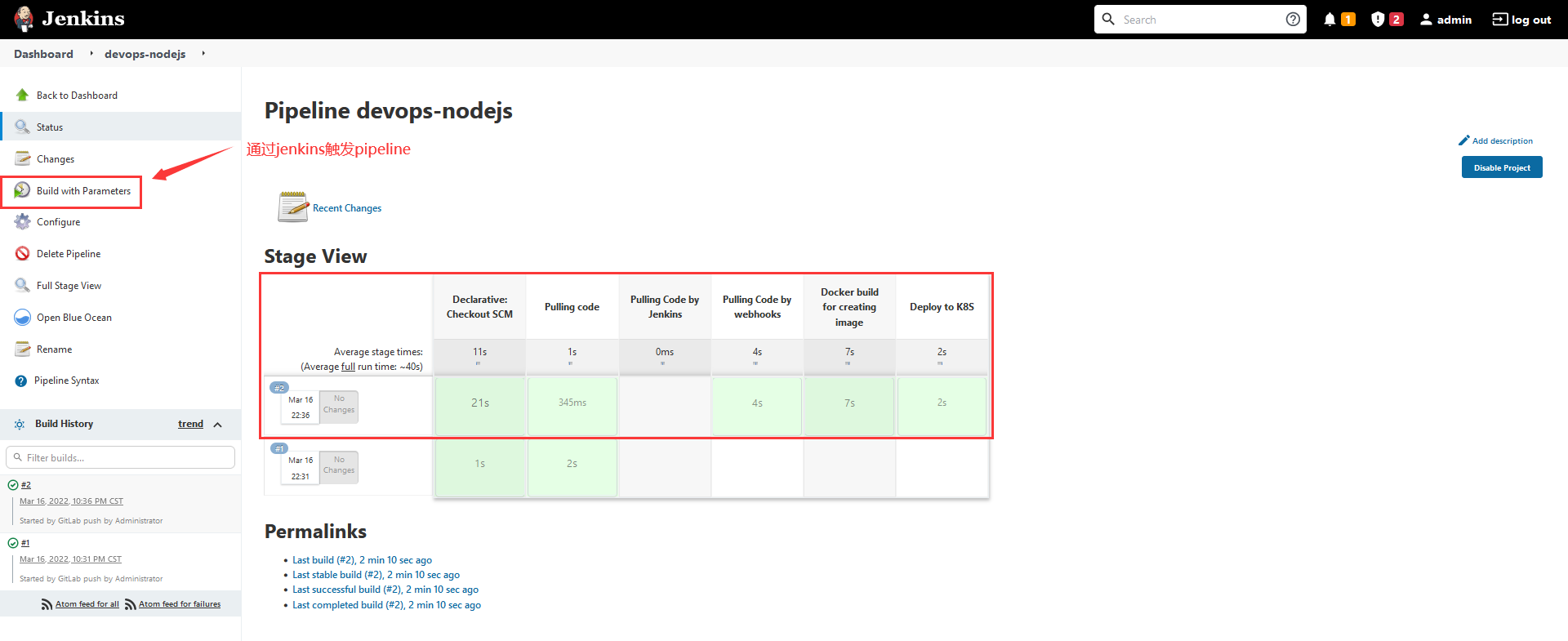
#### 2.3.1 通过Webhooks自动化完成容器镜像构建和项目部署

登录GitLab->Devops-Nodejs->setting->webhooks,通过push events来完成自动化部署

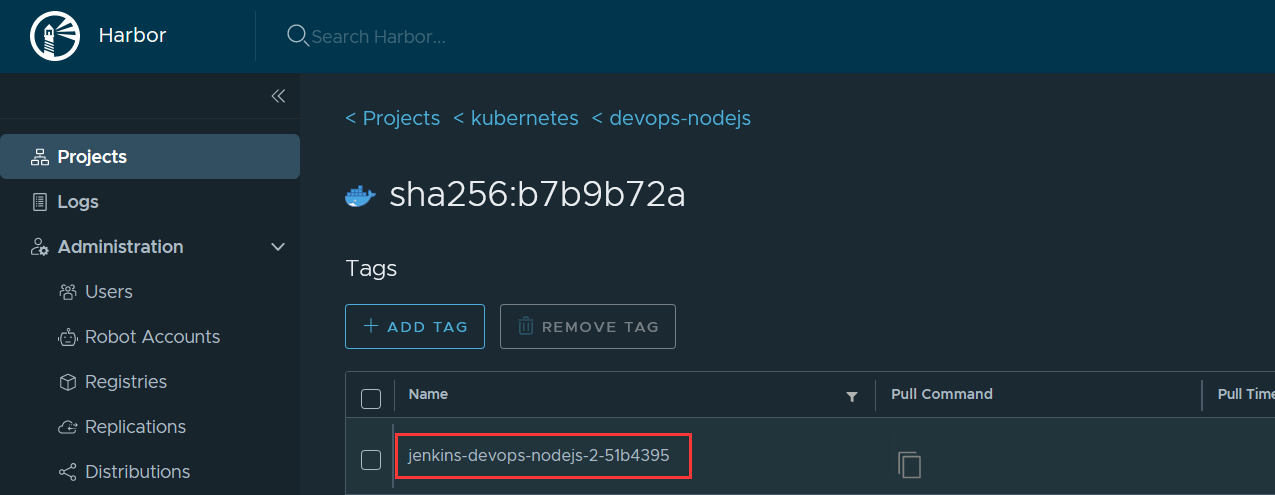




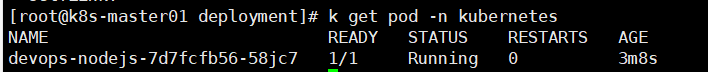


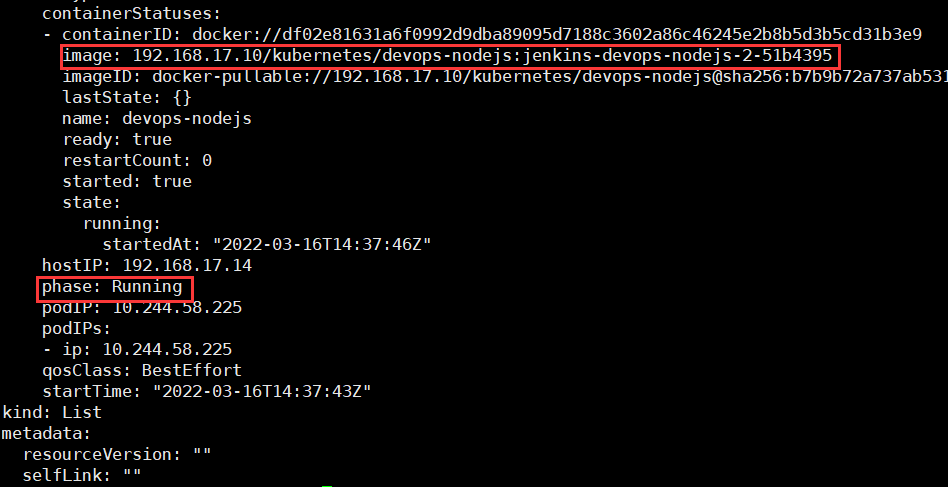


登录Harbor查看，镜像已经上传



登录k8s查看应用是否正常运行，image是否已经更换

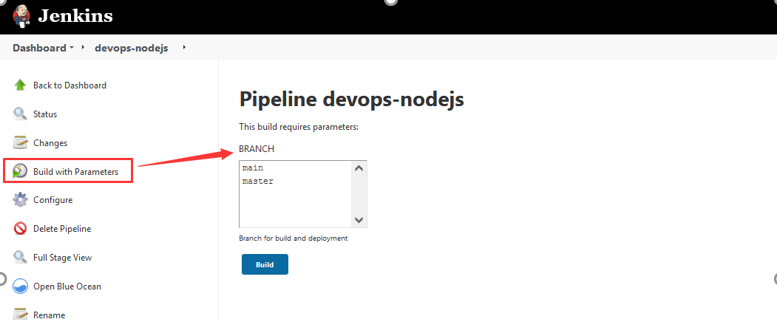


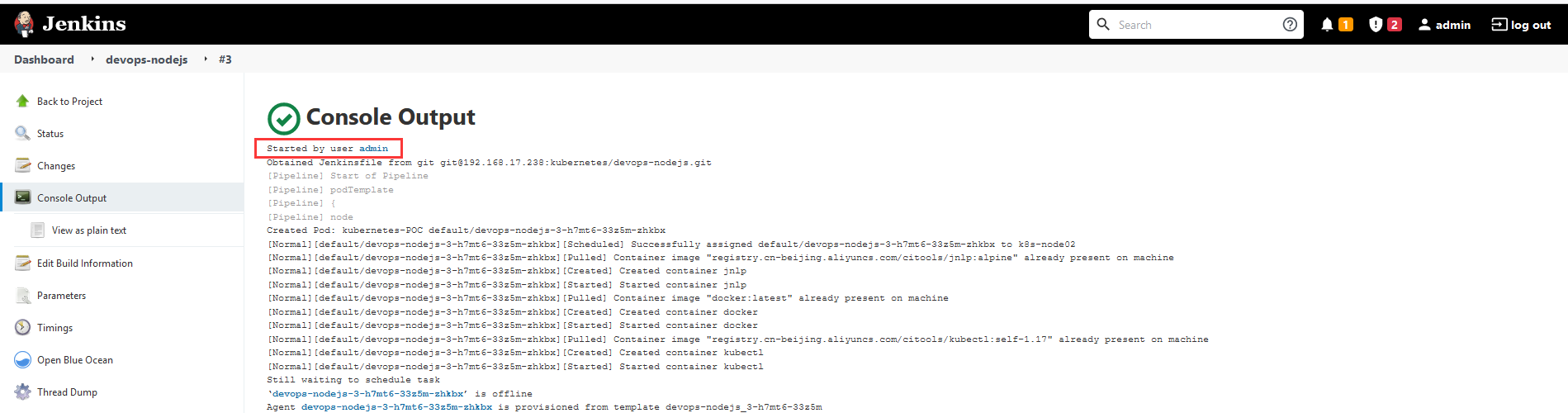


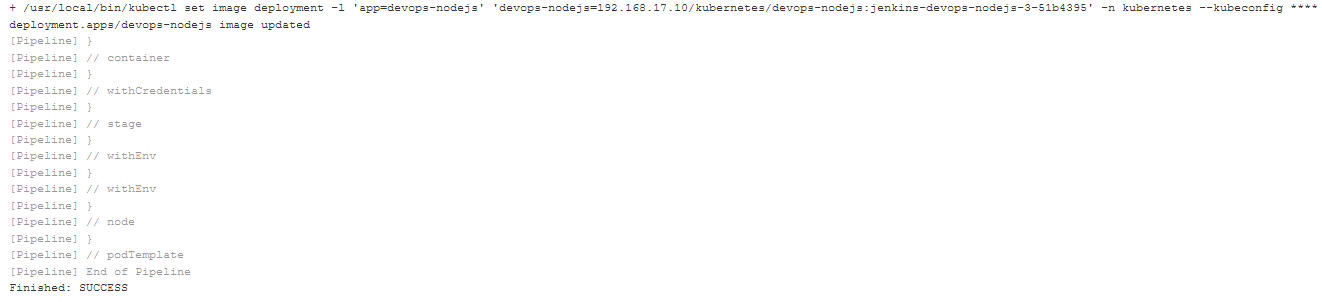
到此，通过webhooks自动化构建应用已经完成。

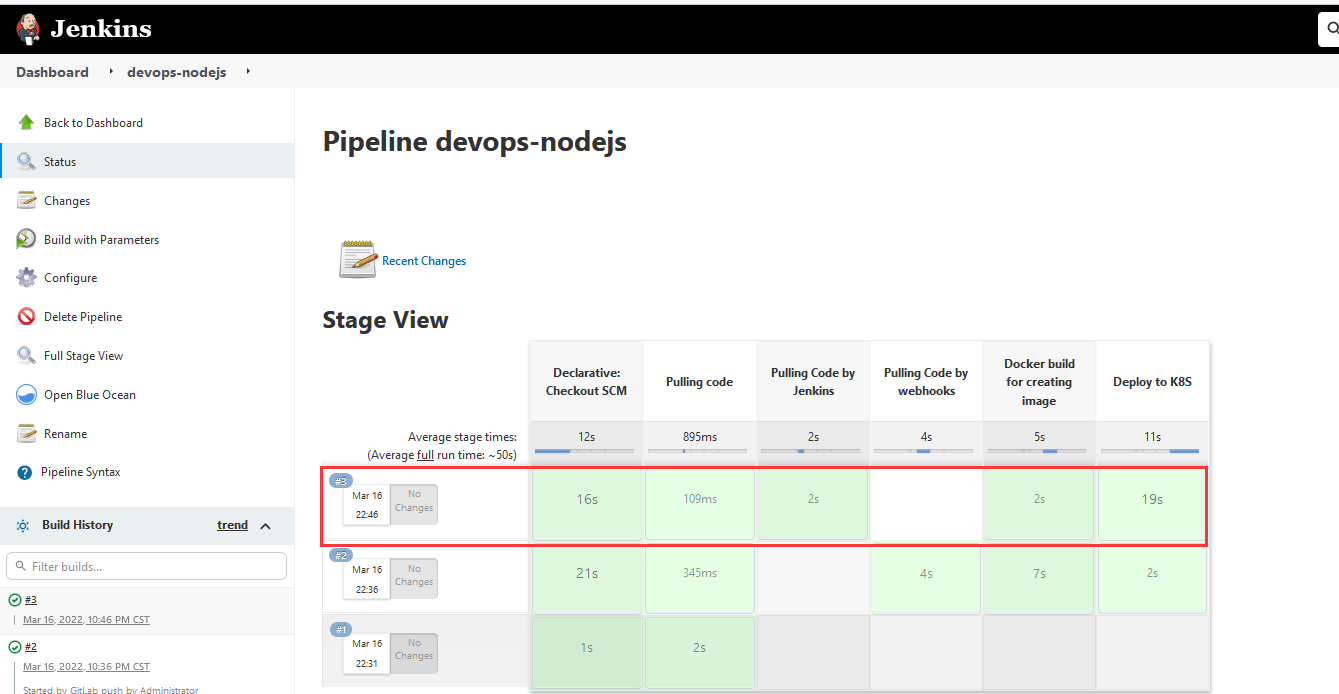
#### 2.3.2通过Jenkins触发pipeline完成容器镜像构建和项目部署

选择项目分支，点击Build即可

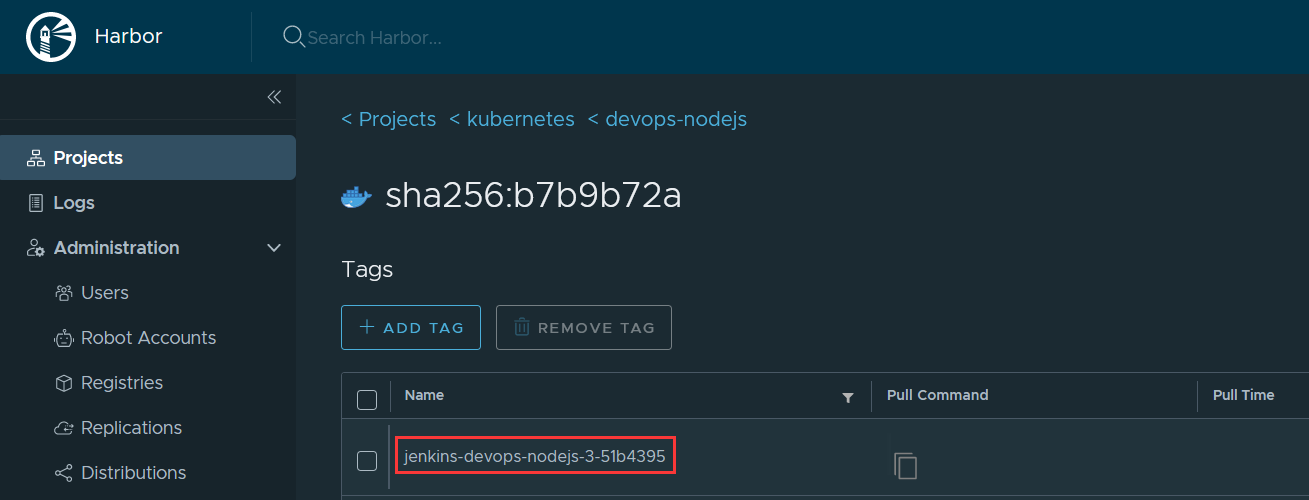




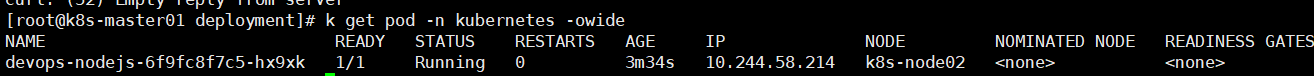


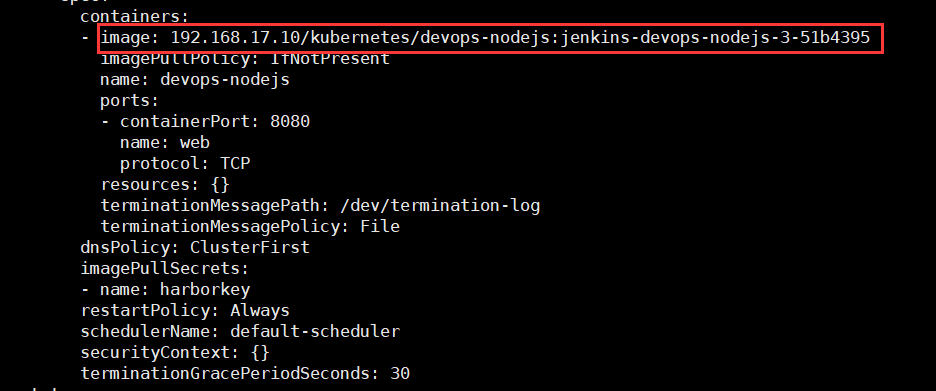


登录Harbor，通过Jenkins触发pipeline生成的新的image已经上传到Harbor



登录K8S，查看pod或deployment运行状态和image





至此，通过Jenkins手动触发pipeline完成项目部署