**构建持续集成**

**规划节点**

节点规划见表1-1-1。

表1-1-1节点规划

|  |  |  |
| --- | --- | --- |
| **IP** | **主机名** | **节点** |
| 10.24.2.16 | master | master节点、Harbor仓库 |
| 10.24.2.17 | node | node节点 |

**基础准备**

登录OpenStack平台，使用提供的CentOS\_7.5\_x86\_64\_XD.qcow2镜像创建两台云主机，并使用提供的软件包部署好双节点Kubernetes集群。

**1.安装Jenkins环境**

（1）基础环境准备

将提供的离线包CICD\_Offline.tar上传至master节点/root目录下，解压文件：

[root@master ~]# tar -zvxf Jenkins.tar.gz

导入镜像并推送到Harbor仓库：

[root@master Jenkins]# docker load -i images/jenkins\_latest.tar

[root@master Jenkins]# docker tag jenkins/jenkins:latest 10.24.2.16/library/jenkins:latest

[root@master Jenkins]# docker push 10.24.2.16/library/jenkins:latest

[root@master Jenkins]# docker load -i images/gitlab-ce\_latest.tar

[root@master Jenkins]# docker tag gitlab/gitlab-ce:latest 10.24.2.16/library/gitlab-ce:latest

[root@master Jenkins]# docker push 10.24.2.16/library/gitlab-ce:latest

[root@master Jenkins]# docker load -i images/java\_8-jre.tar

[root@master Jenkins]# docker tag java:8-jre 10.24.2.16/library/java:8-jre

[root@master Jenkins]# docker push 10.24.2.16/library/java:8-jre

（2）安装Jenkins

编写Jenkins资源清单文件：

apiVersion: v1

kind: Service

metadata:

name: jenkins

labels:

app: jenkins

spec:

type: NodePort

ports:

- name: http

port: 8080 #服务端口

targetPort: 8080

nodePort: 30880 #NodePort方式暴露 Jenkins 端口

selector:

app: jenkins

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: jenkins

labels:

app: jenkins

spec:

selector:

matchLabels:

app: jenkins

template:

metadata:

labels:

app: jenkins

spec:

nodeName: master

serviceAccountName: jenkins-admin

containers:

- name: jenkins

image: 10.24.2.16/library/jenkins:latest

securityContext:

runAsUser: 0 #设置以ROOT用户运行容器

privileged: true #拥有特权

ports:

- name: http

containerPort: 8080

volumeMounts:

- mountPath: /var/jenkins\_home

name: jenkinshome

- mountPath: /usr/bin/docker

name: docker

- mountPath: /var/run/docker.sock

name: dockersock

- mountPath: /usr/bin/kubectl

name: kubectl

- mountPath: /root/.kube

name: kubeconfig

volumes:

- name: jenkinshome

hostPath:

path: /home/jenkins\_home

- name: docker

hostPath:

path: /usr/bin/docker

- name: dockersock

hostPath:

path: /var/run/docker.sock

- name: kubectl

hostPath:

path: /usr/bin/kubectl

- name: kubeconfig

hostPath:

path: /root/.kube

部署Jenkins需要使用到一个拥有相关权限的serviceAccount，名称为jenkins，可以给jenkins赋予一些必要的权限，也可以直接绑定一个cluster-admin的集群角色权限，此处选择给予集群角色权限。

编写资源清单文件：

[root@master Jenkins]# cat jenkins-rbac.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: jenkins-admin

labels:

name: jenkins

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: jenkins-admin

labels:

name: jenkins

subjects:

- kind: ServiceAccount

name: jenkins-admin

namespace: default

roleRef:

kind: ClusterRole

name: cluster-admin

apiGroup: rbac.authorization.k8s.io

创建资源：

[root@k8s-master-node1 jenkins-slave]# kubectl apply -f jenkins-deployment.yaml -f jenkins-rbac.yaml

service/jenkins created

deployment.apps/jenkins created

serviceaccount/jenkins-admin created

clusterrolebinding.rbac.authorization.k8s.io/jenkins-admin created

查看Pod：

[root@master Jenkins]# kubectl get pods

NAME READY STATUS RESTARTS AGE

jenkins-b89d4c4d8-bc6tn 1/1 Running 0 50s

查看Jenkins Service端口：

[root@master Jenkins]# kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

jenkins NodePort 10.96.53.202 <none> 8080:30880/TCP,50000:30850/TCP 113s

通过http://master\_IP:30880访问Jenkins，如图所示：



获取Jenkins密码：

[root@master Jenkins]# kubectl exec deploy/jenkins -- cat /var/jenkins\_home/secrets/initialAdminPassword

08f73c50281e4026bb2287abd7fa19f8

输入密码后点击“继续”，如图所示：



将离线插件包拷贝到Jenkins：

[root@master Jenkins]# kubectl cp plugins/ jenkins-b89d4c4d8-bc6tn:/var/jenkins\_home

重启Jenkins：

[root@master Jenkins]# kubectl rollout restart deployment jenkins

deployment.apps/jenkins restarted

刷新Jenkins界面，选择“安装推荐的插件”，安装完成后进入用户创建页面，创建一个用户jenkins，密码000000，如图所示：



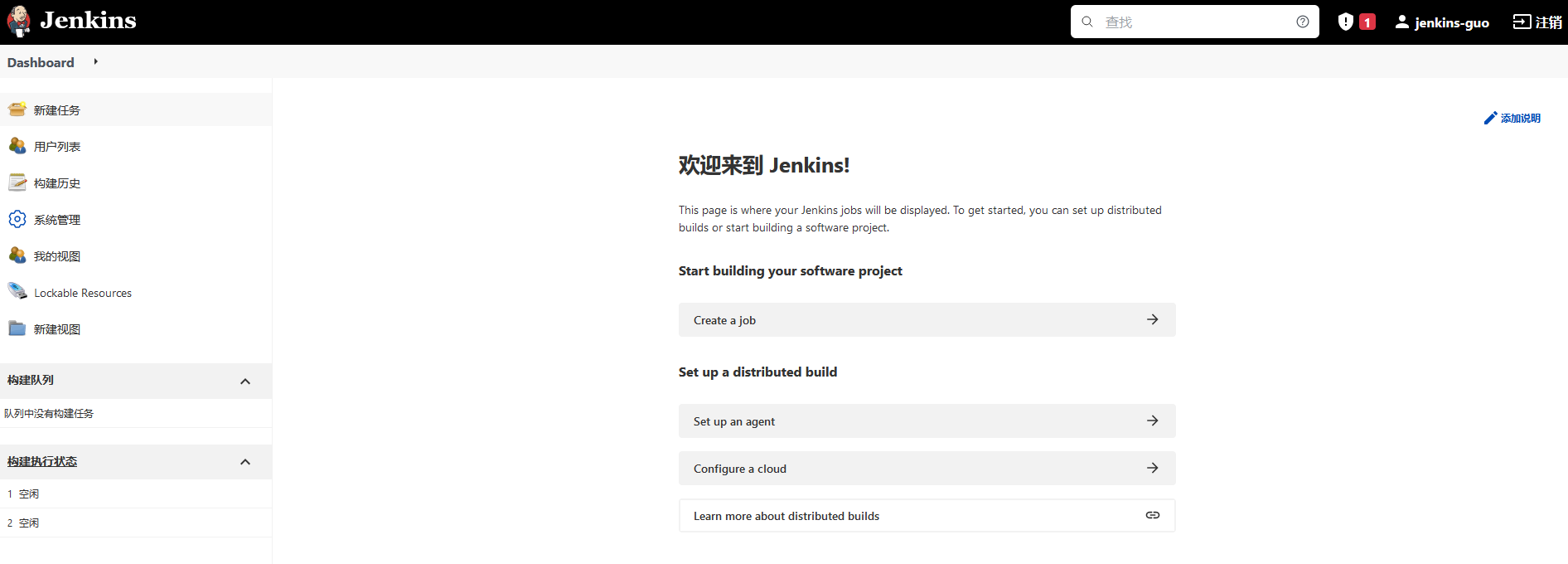
点击“保存并完成”，如图所示：



点击“保存并完成”，如图所示：



点击“开始使用Jenkins”并使用新创建的用户登录Jenkins，如图所示：



**2.部署Gitlab**

GitLab是利用Ruby on Rails一个开源的版本管理系统，实现一个自托管的Git项目仓库，可通过Web界面进行访问公开的或者私人项目。与Github类似，GitLab能够浏览源代码，管理缺陷和注释，可以管理团队对仓库的访问，它非常易于浏览提交过的版本并提供一个文件历史库，团队成员可以利用内置的简单聊天程序(Wall)进行交流。Gitlab还提供一个代码片段收集功能可以轻松实现代码复用，便于日后有需要的时候进行查找。

本项目Gitlab与Harbor共用一台服务器。

（1）部署GitLab

编写GitLab资源清单文件：

[root@master Jenkins]# cat gitlab-deployment.yaml

apiVersion: v1

kind: Service

metadata:

name: gitlab

spec:

type: NodePort

ports:

- port: 443

nodePort: 30443

targetPort: 443

name: gitlab-443

- port: 80

nodePort: 30888

targetPort: 80

name: gitlab-80

selector:

app: gitlab

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: gitlab

spec:

selector:

matchLabels:

app: gitlab

revisionHistoryLimit: 2

template:

metadata:

labels:

app: gitlab

spec:

containers:

- image: 10.24.2.16/library/gitlab-ce:latest

name: gitlab

imagePullPolicy: IfNotPresent

env:

- name: GITLAB\_ROOT\_PASSWORD

value: admin123

- name: GITLAB\_ROOT\_EMAIL

value: guorui@example.com

- name: GITLAB\_PORT

value: "80"

ports:

- containerPort: 443

name: gitlab-443

- containerPort: 80

name: gitlab-80

启动Gitlab：

[root@master Jenkins]# kubectl apply -f gitlab-deployment.yaml

service/gitlab created

deployment.apps/gitlab created

查看Pod：

[root@master Jenkins]# kubectl get pods

NAME READY STATUS RESTARTS AGE

gitlab-6778c45f9-xx5gs 1/1 Running 0 30s

jenkins-b89d4c4d8-bc6tn 1/1 Running 0 5m36s

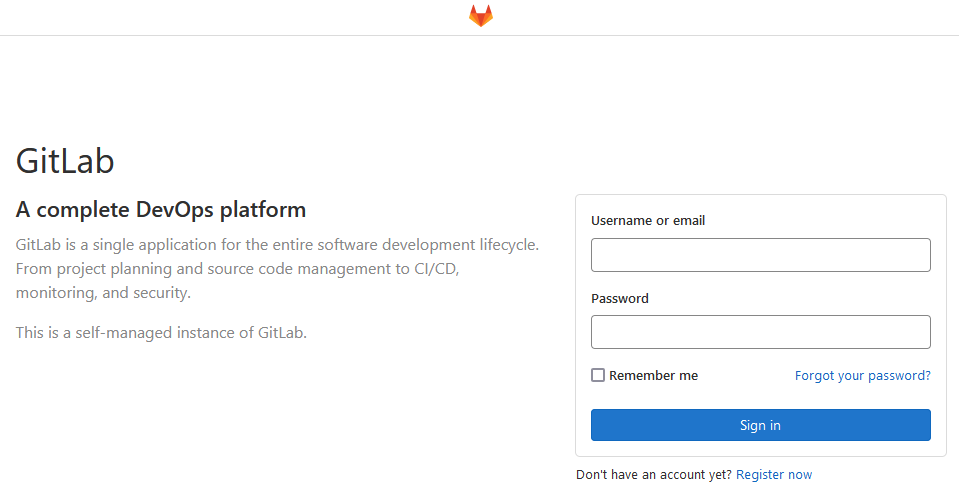
查看GitLab Service：

[root@master Jenkins]# kubectl get svc

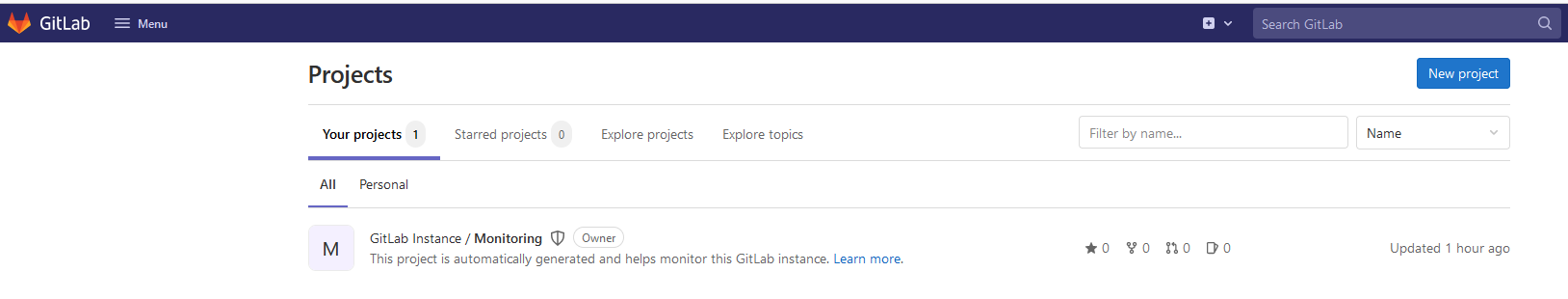
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

gitlab NodePort 10.107.127.252 <none> 443:30443/TCP,80:30888/TCP 51s

Gitlab启动较慢，可以通过docker logs查看启动状态。启动完成后，在web端访问Gitlab（http://master\_IP:30888），如图所示：

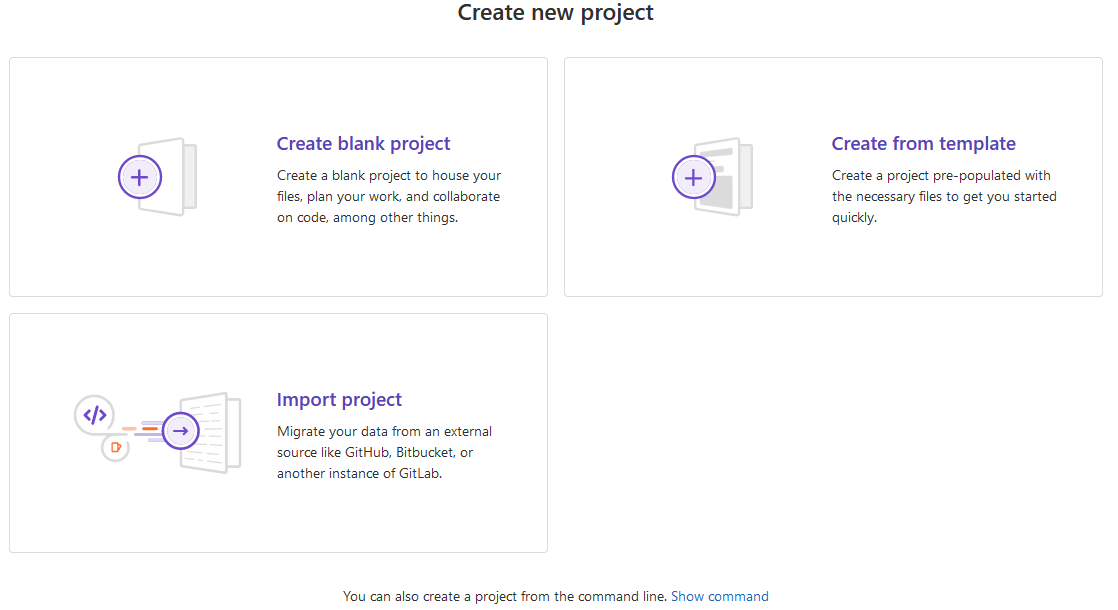


登录Gitlab，如图所示：

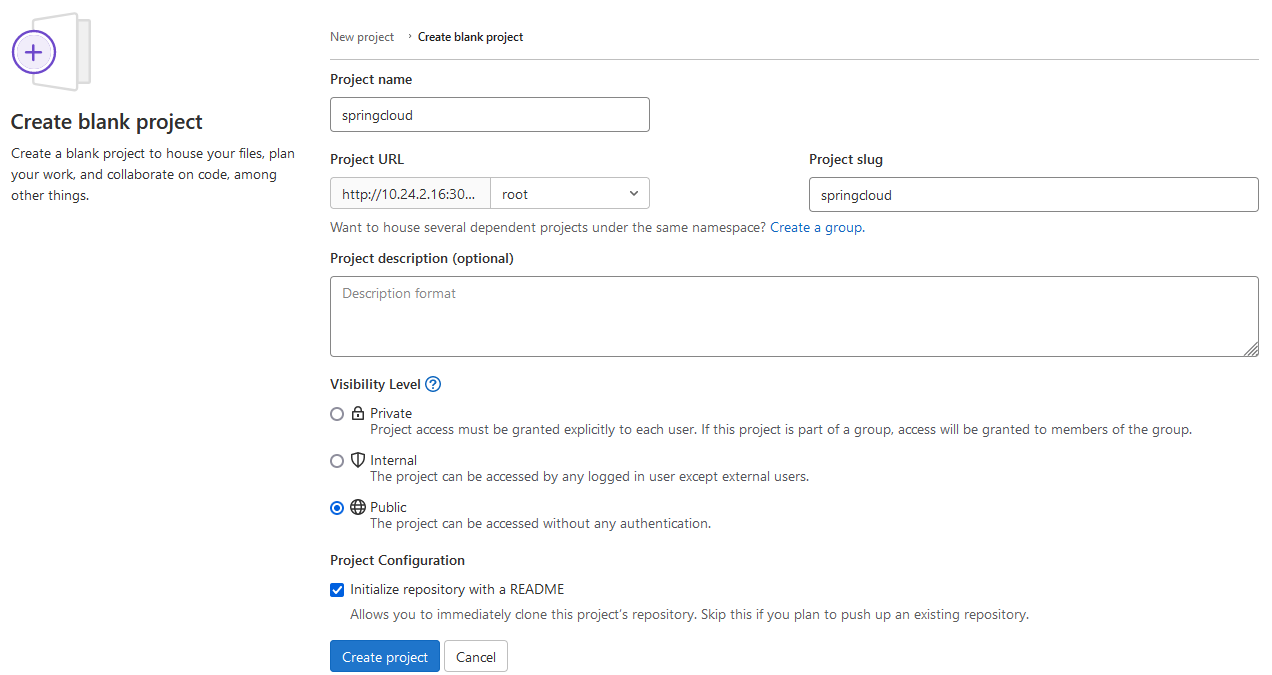


（2）创建项目

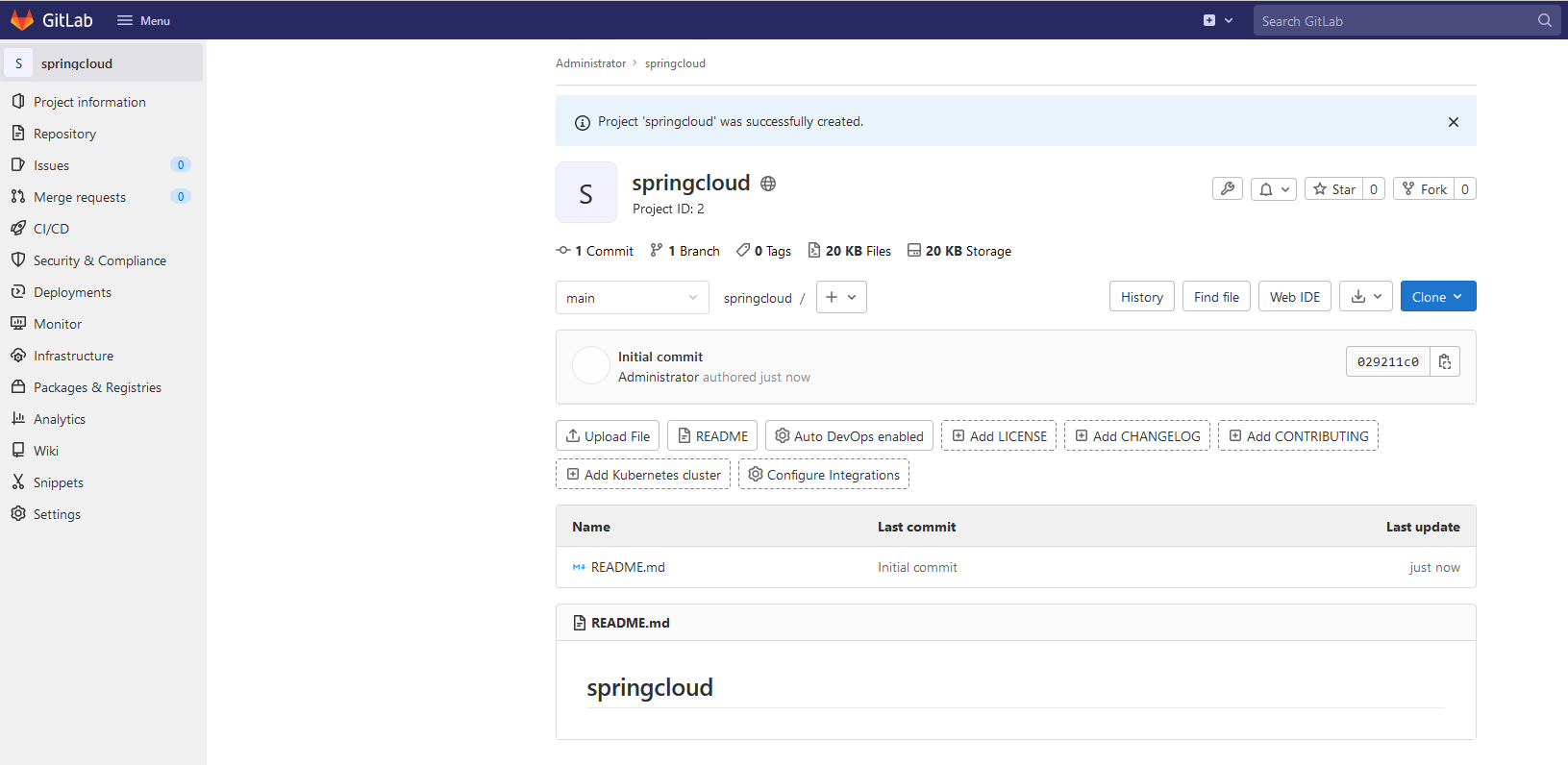
点击“New project”，如图所示：



点击“Create blank project”创建项目springcloud，可见等级选择“Public”，如图所示：



点击“Create project”，进入项目，如图所示：



push源代码到gitlab的springcloud项目：

[root@master Jenkins]# yum install -y git

[root@master Jenkins]# cd springcloud/

[root@master springcloud]# git config --global user.name "administrator"

[root@master springcloud]# git config --global user.email "admin@example.com"

[root@master springcloud]# git remote remove origin

[root@master springcloud]# git remote add origin http://10.24.2.16:30888/root/springcloud.git

[root@master springcloud]# git add .

[root@master springcloud]# git commit -m "initial commit"

# On branch master

nothing to commit, working directory clean

[root@master springcloud]# git push -u origin master

Username for 'http://10.24.2.16:30888': root

Password for 'http://root@10.24.2.16:30888':

Counting objects: 3192, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (1428/1428), done.

Writing objects: 100% (3192/3192), 1.40 MiB | 1.70 MiB/s, done.

Total 3192 (delta 1233), reused 3010 (delta 1207)

remote: Resolving deltas: 100% (1233/1233), done.

remote:

remote: To create a merge request for master, visit:

remote: http://gitlab-6778c45f9-xx5gs/root/springcloud/-/merge\_requests/new?merge\_request%5Bsource\_branch%5D=master

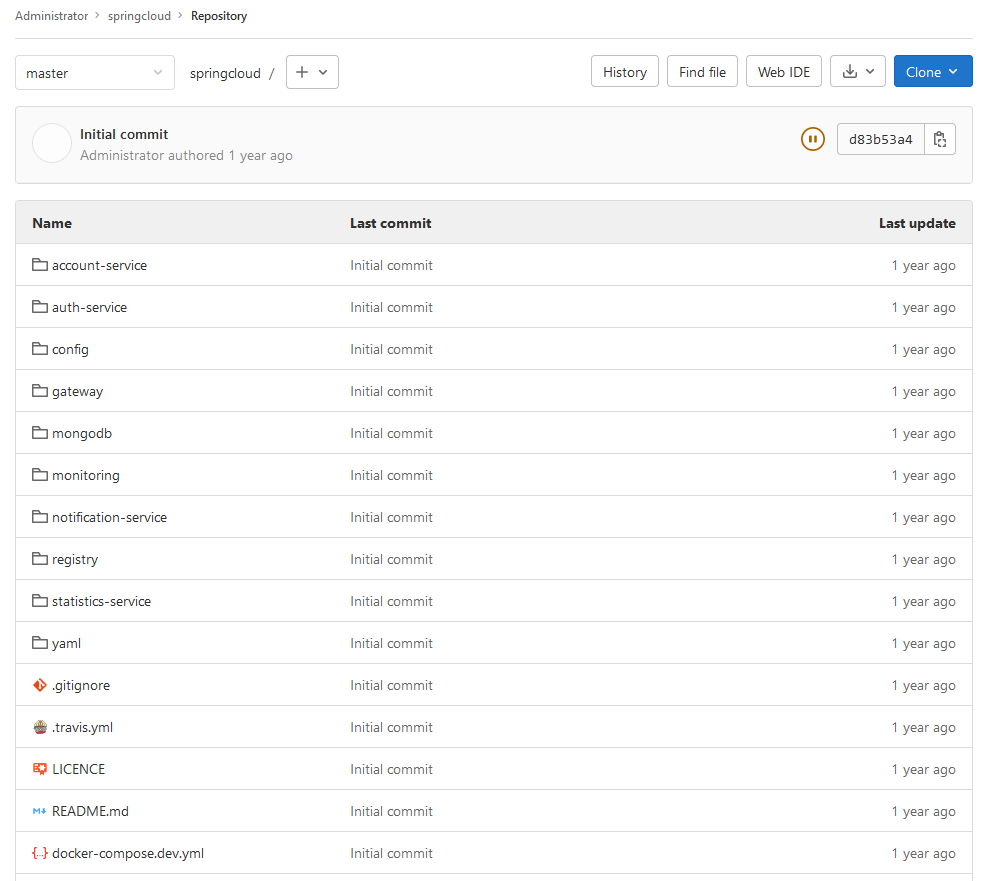
remote:

To http://10.24.2.16:30888/root/springcloud.git

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

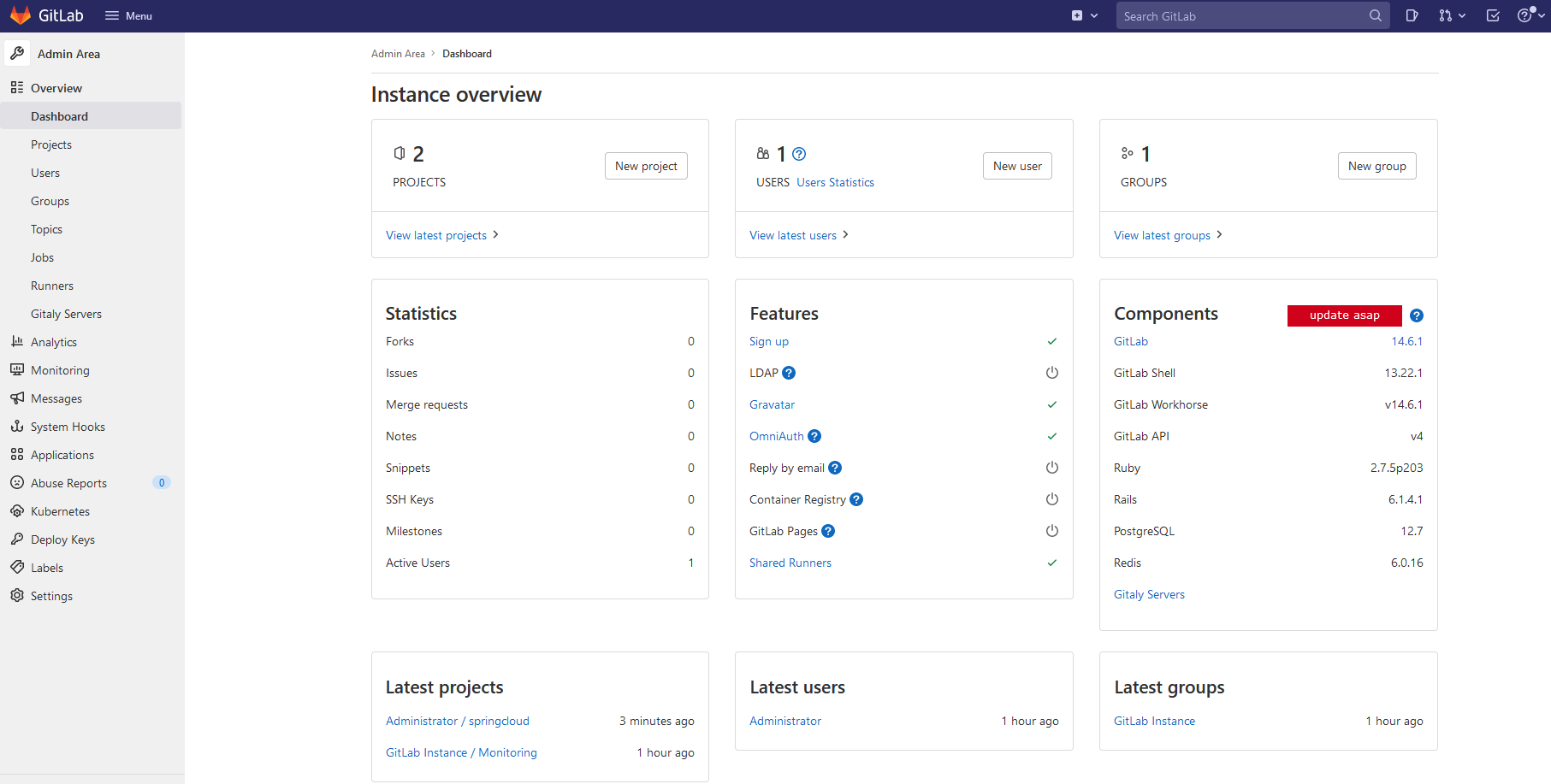
刷新网页，springcloud项目master分支中的文件已经更新了，如图所示：



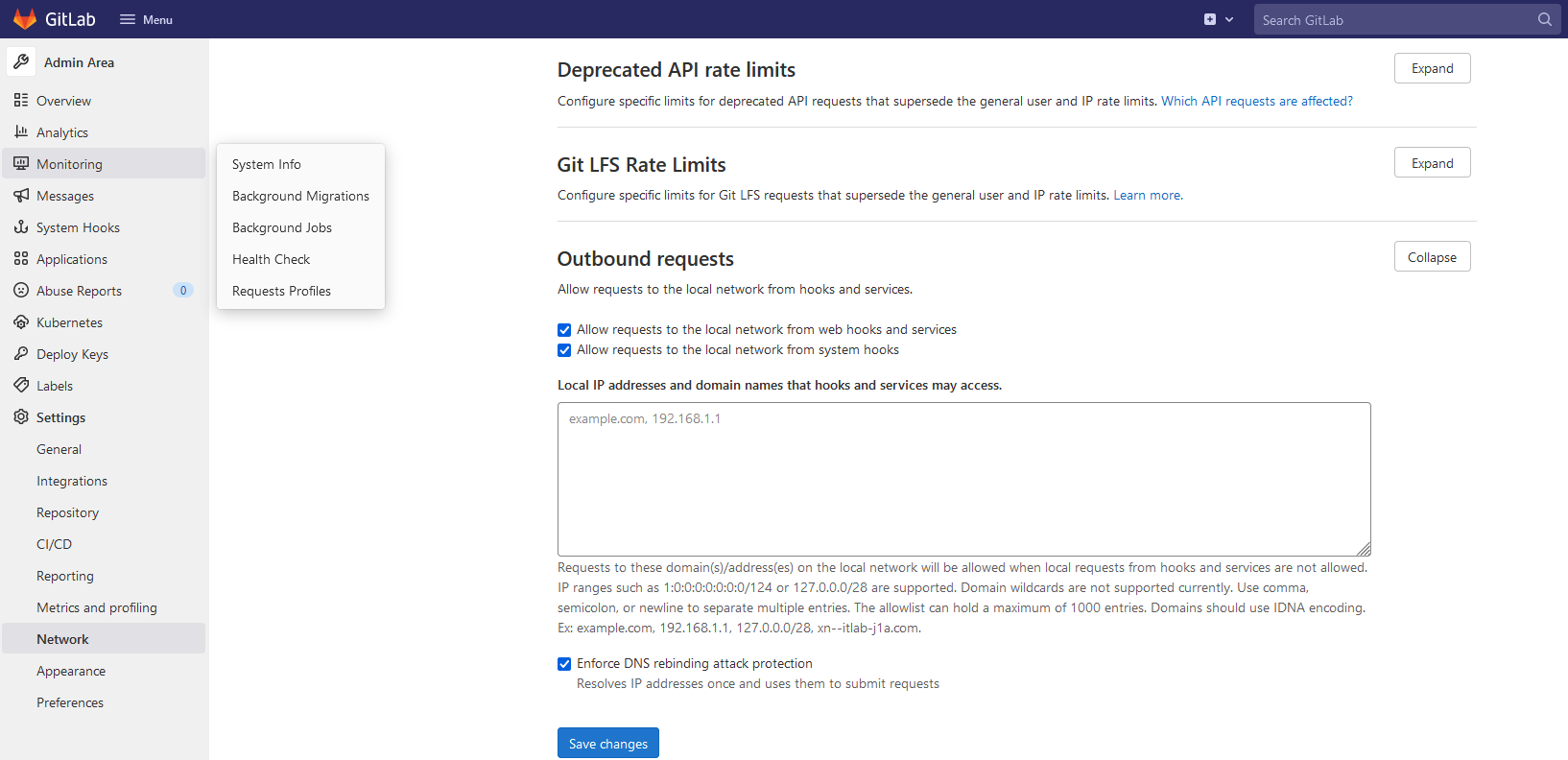
**3.配置Jenkins连接Gitlab**

（1）设置Outbound requests

登录Gitlab管理员界面（http://master\_IP:30888/admin），如图所示：



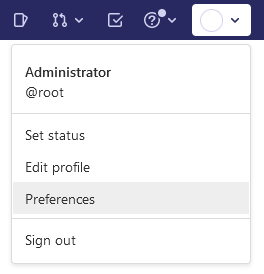
点击左侧导航栏的“Settings”→“Network”，设置“Outbound requests”，勾选“Allow requests to the local network from web hooks and services”，如图所示：



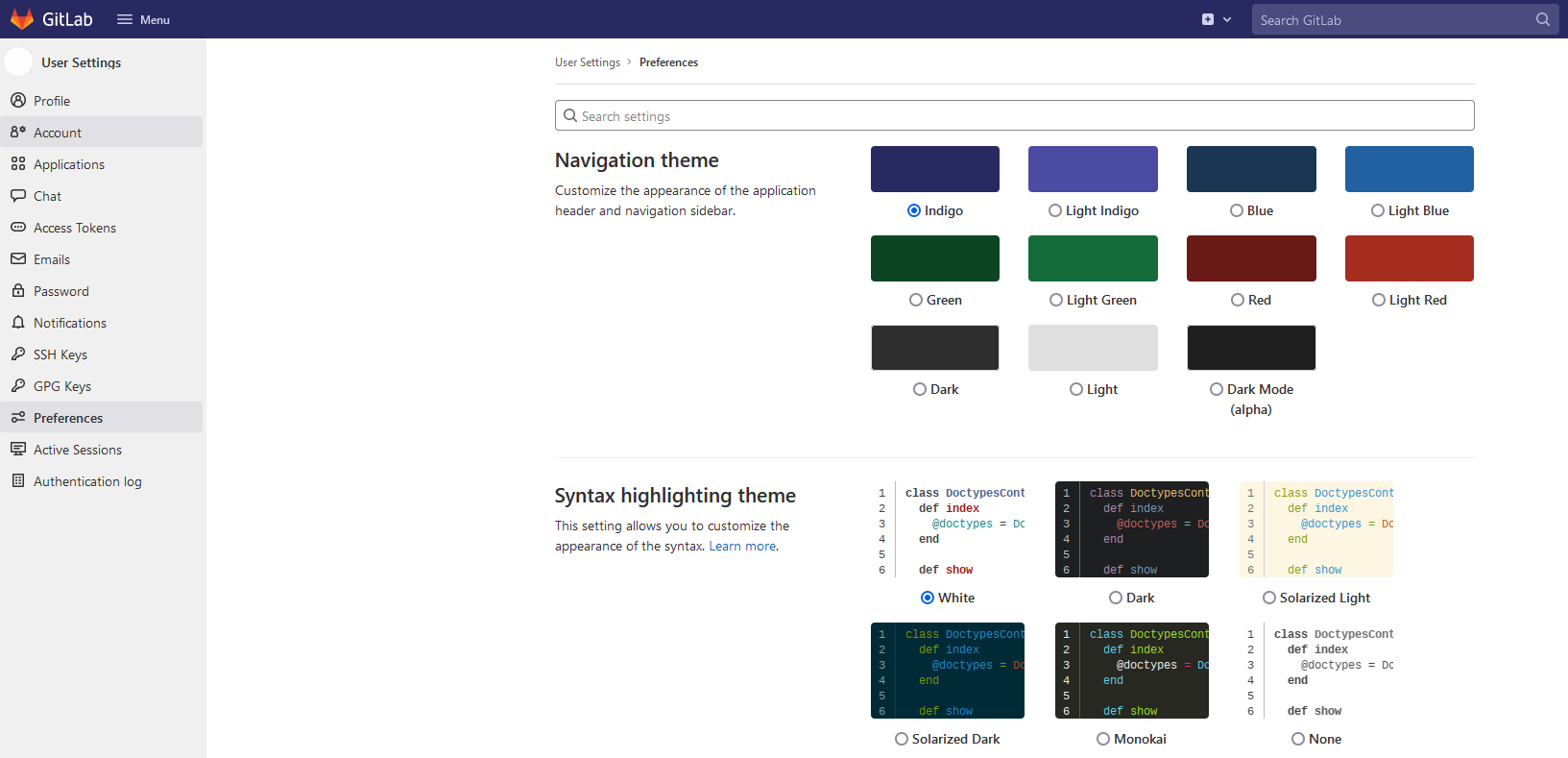
配置完成后保存。

（2）创建Gitlab API Token

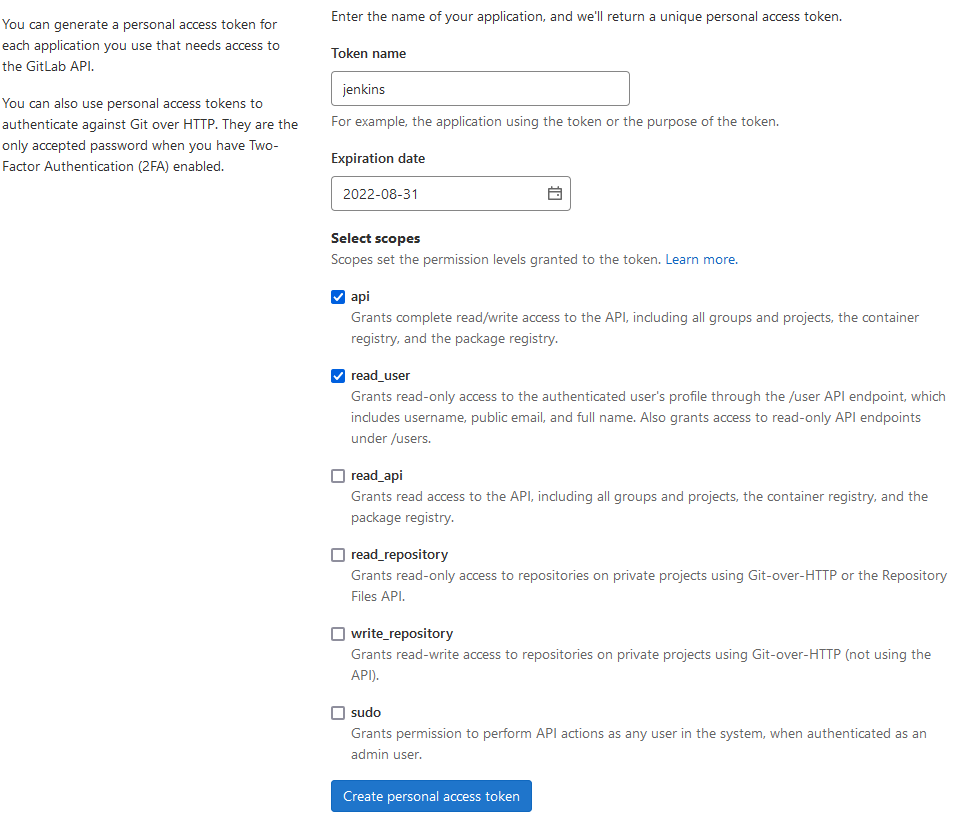
点击Gitlab用户头像图标，如图所示：



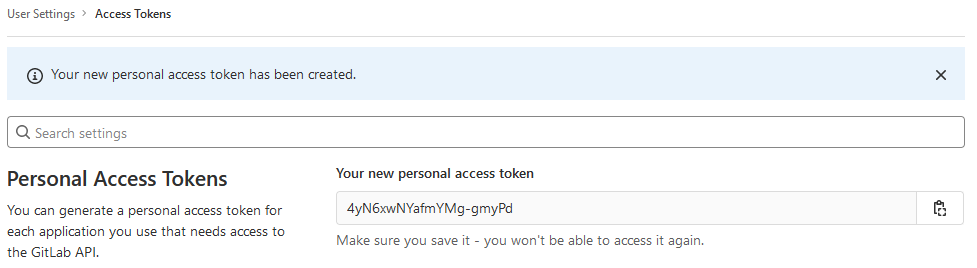
点击“Preferences”，如图所示：



点击左侧导航栏的“Access Tokens”添加token，如图所示：



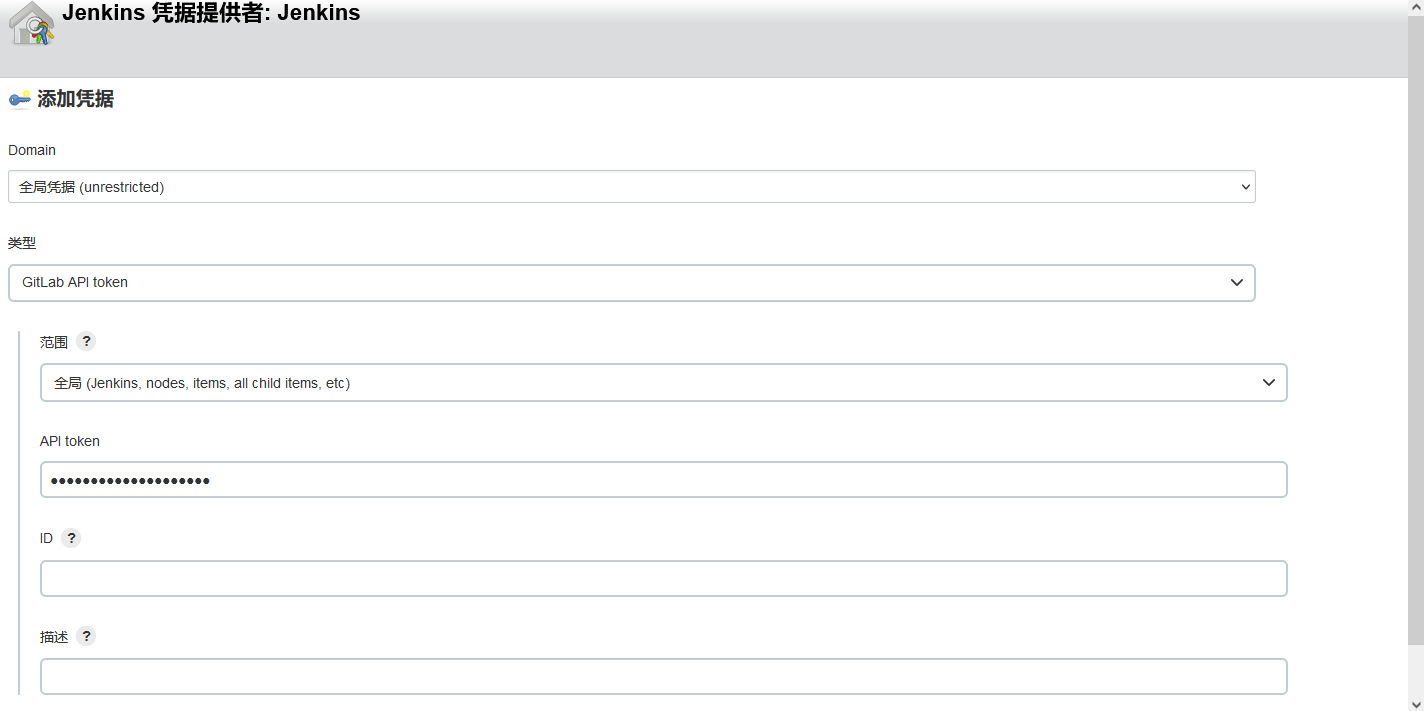
点击“Create personal access token”生成Token，如图所示：



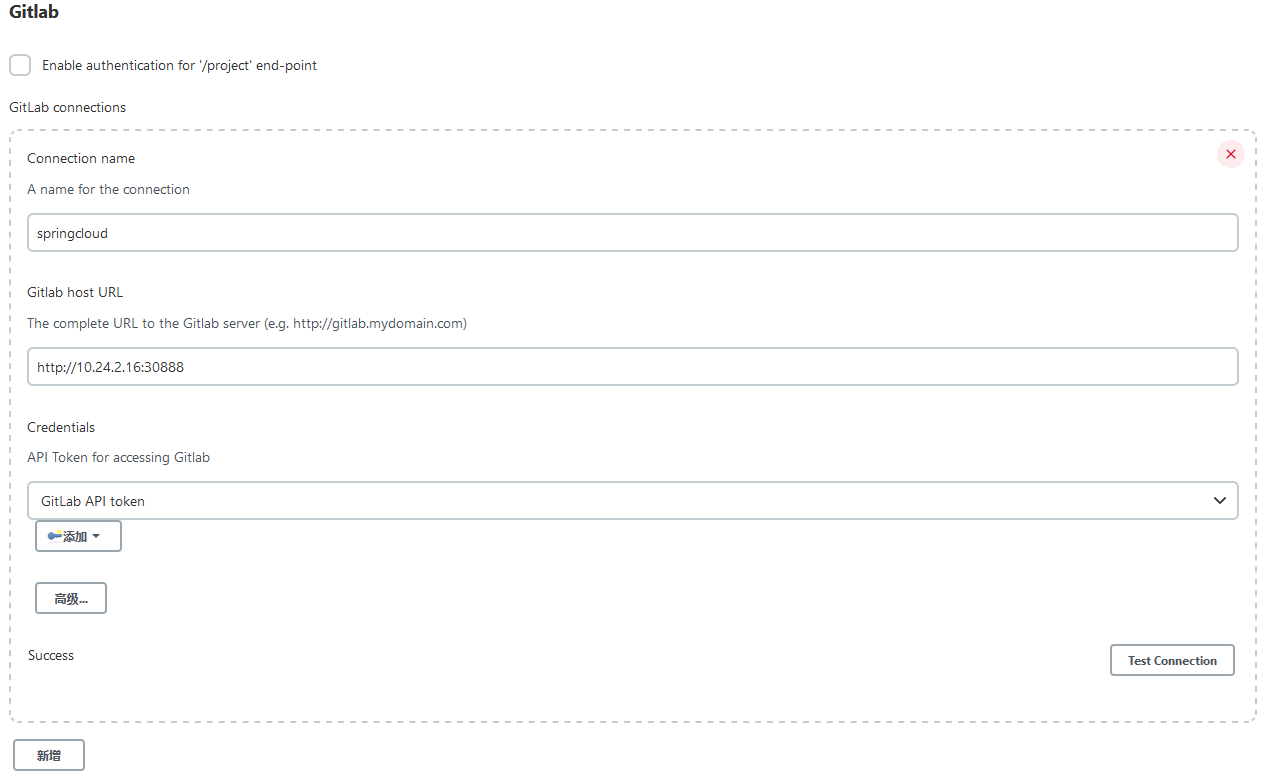
复制Token（4yN6xwNYafmYMg-gmyPd），后面配置Jenkins时会用到。

（3）设置Jenkins

登录Jenkins首页，点击“系统管理”→“系统配置”，配置Gitlab信息，取消勾选“Enable authentication for '/project' end-point”，输入“Connection name”和“Gitlab host URL”，然后点击“添加”→“Jenkins”添加认证信息，将Gitlab API Token填入，如图所示：



点击“Test Connection”，如图所示：



**4.配置Jenkins连接maven**

（1）安装maven

由于Jenkins是采用docker in docker的方式启动的，所以需要在jenkins容器内安装maven：

[root@master springcloud]# cd ../

[root@master Jenkins]# kubectl cp apache-maven-3.6.3-bin.tar.gz jenkins-7d9bd56966-9kd72:/

[root@master Jenkins]# kubectl exec -it jenkins-7d9bd56966-9kd72 -- bash

root@jenkins-7d9bd56966-9kd72:/# tar -zxvf apache-maven-3.6.3-bin.tar.gz -C .

root@jenkins-7d9bd56966-9kd72:/# mv apache-maven-3.6.3/ /usr/local/maven

root@jenkins-7d9bd56966-9kd72:/# sed -i '$a\export M2\_HOME=/usr/local/maven' /etc/profile

root@jenkins-7d9bd56966-9kd72:/# sed -i '$a\export PATH=\$PATH:\$M2\_HOME/bin' /etc/profile

root@jenkins-7d9bd56966-9kd72:/# sed -i '$a\source /etc/profile' /root/.bashrc

退出容器重新进入：

[root@master Jenkins]# kubectl exec -it jenkins-7d9bd56966-9kd72 -- bash

root@jenkins-7d9bd56966-9kd72:/# mvn -v

Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)

Maven home: /usr/local/maven

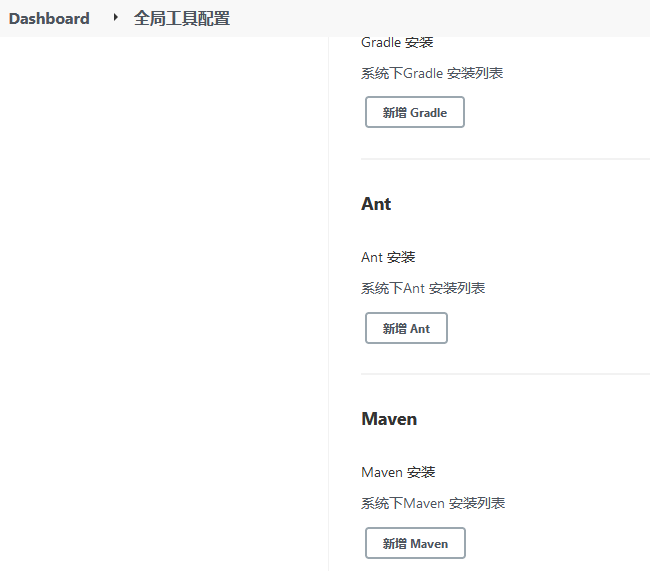
Java version: 11.0.14.1, vendor: Eclipse Adoptium, runtime: /opt/java/openjdk

Default locale: en, platform encoding: UTF-8

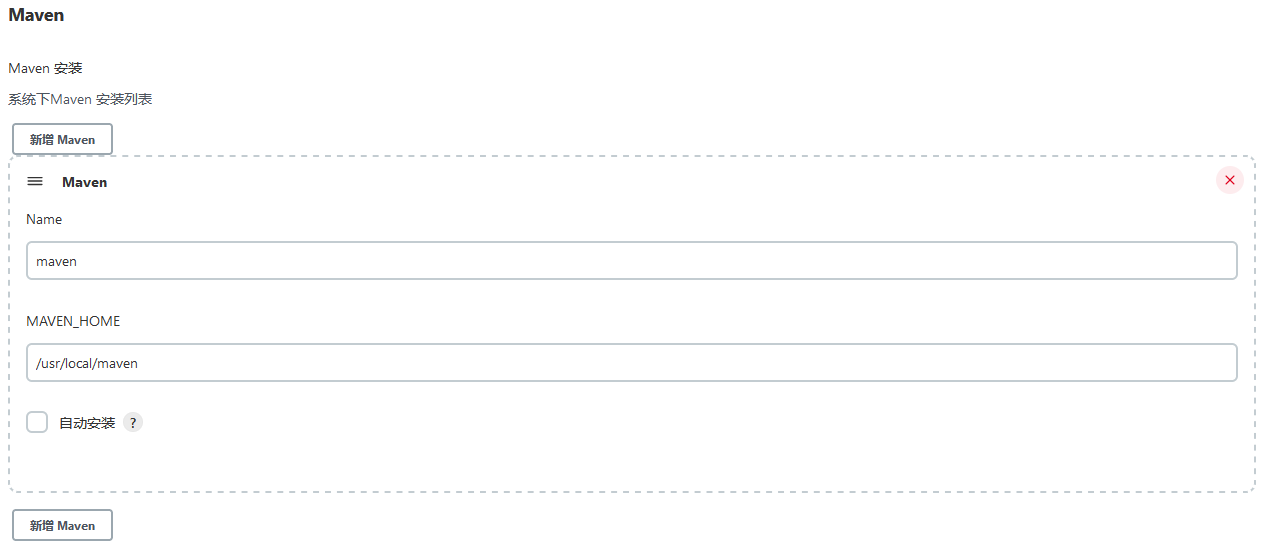
OS name: "linux", version: "3.10.0-1160.el7.x86\_64", arch: "amd64", family: "unix"

（2）连接maven

登录Jenkins首页，点击“系统管理”→“全局工具配置”，如图所示：



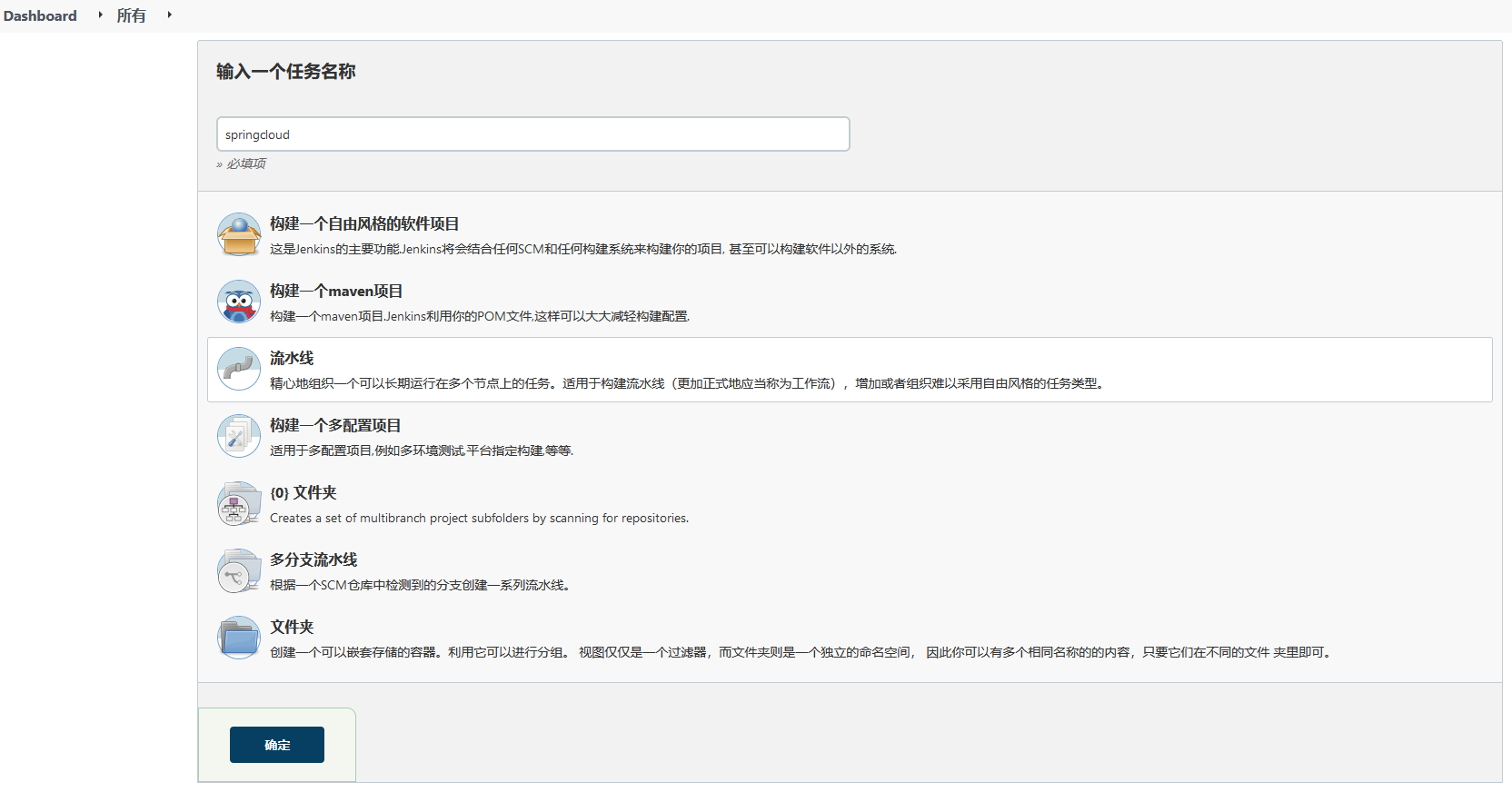
点击“新增Maven”，如图所示。取消勾选“自动安装”，填入maven名称和安装路径，配置完成后点击“应用”。



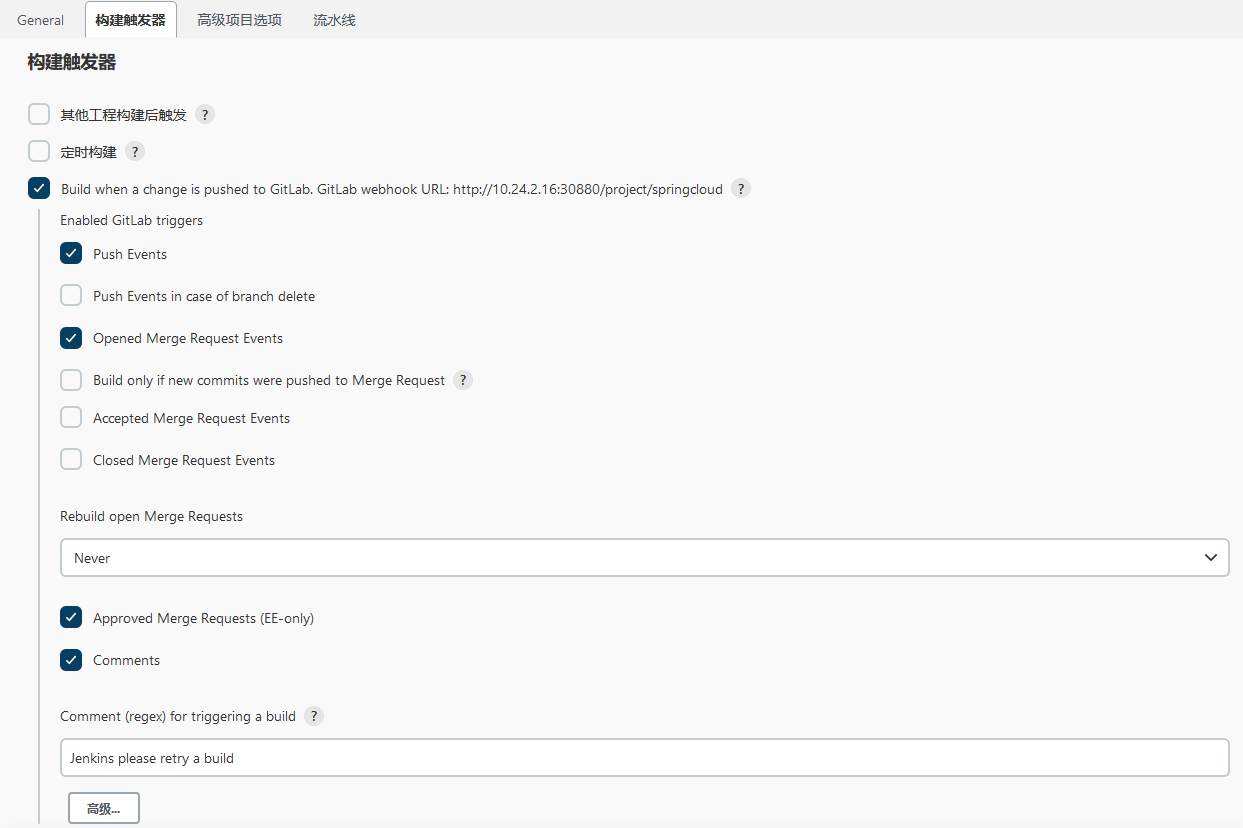
**5. 配置CI/CD**

（1）新建任务

登录Jenkins首页，点击左侧导航栏“新建任务”，如图所示，选择构建一个流水线。

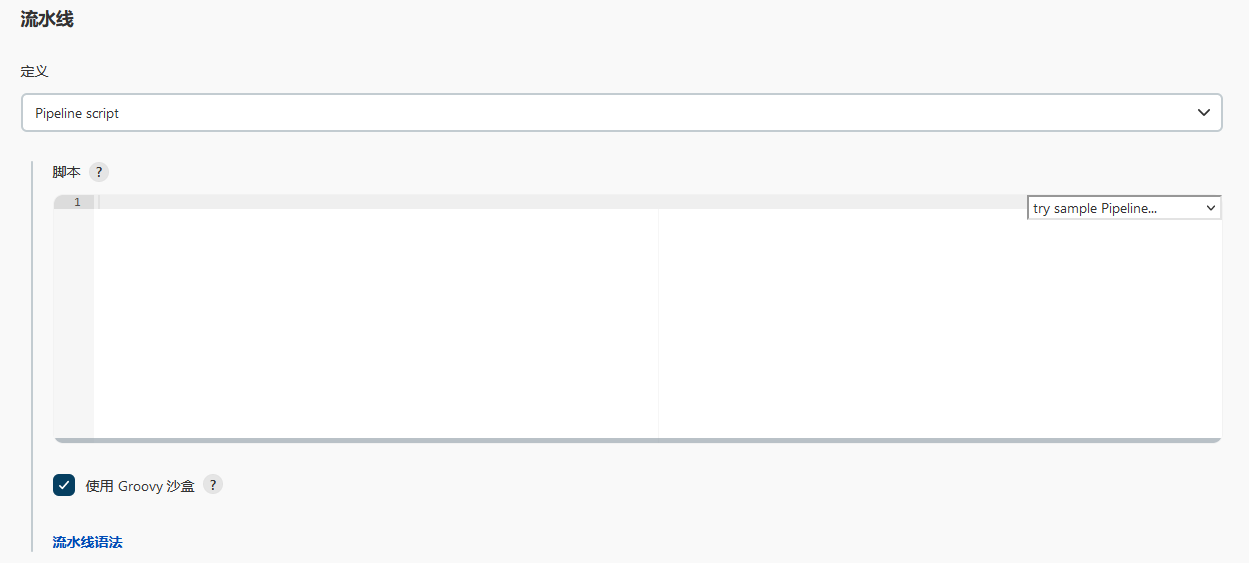


点击“确定”，配置构建触发器，如图所示：

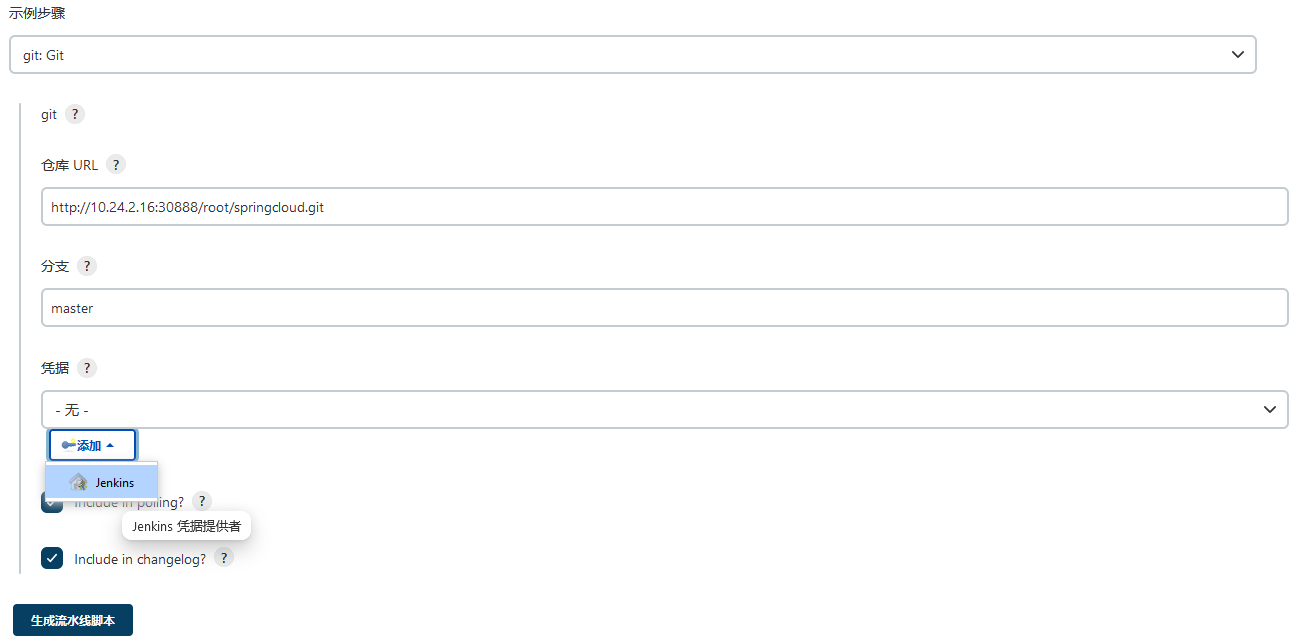


记录下GitLab webhook URL的地址（http://10.24.2.16:30880/project/springcloud），后期配置webhook需要使用。

配置流水线，如图所示：



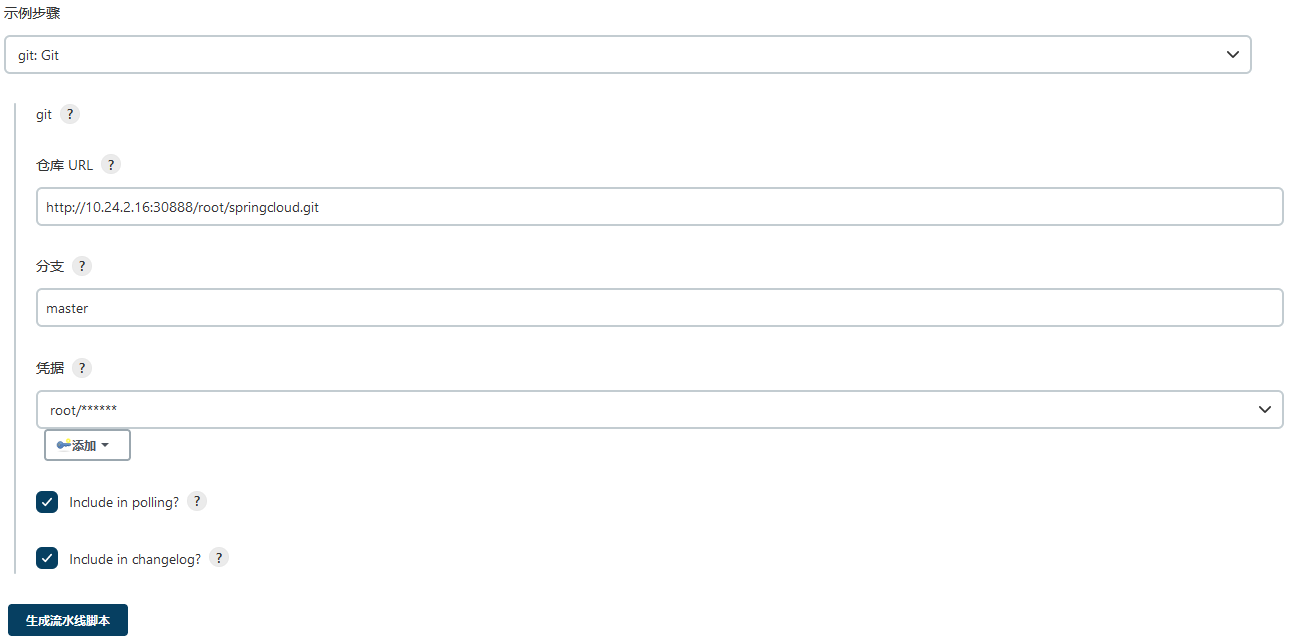
点击“流水线语法”，如图所示，示例步骤选择“git：Git”，将springcloud项目地址填入仓库URL。



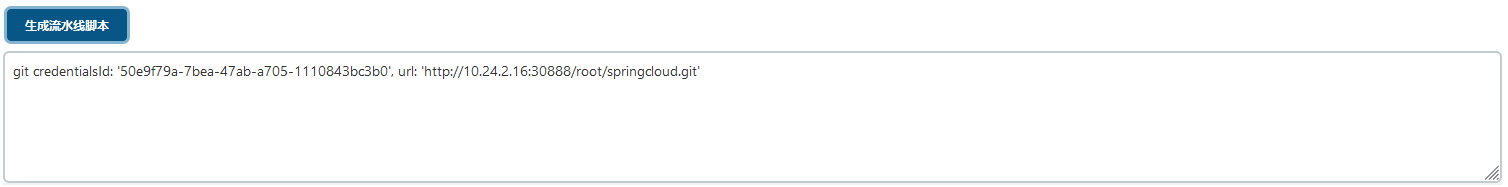
点击“添加”→“jenkins”添加凭据，如图所示。类型选择“Username with password”，用户名和密码为Gitlab仓库的用户名和密码。



添加凭据后选择凭据，如图所示：



点击“生成流水线脚本”，如图所示：



记录生成的值，并将其写入流水线脚本中，完整的流水线脚本如下：

node{

stage('git clone'){

//check CODE

git credentialsId: '50e9f79a-7bea-47ab-a705-1110843bc3b0', url: 'http://10.24.2.16:30888/root/springcloud.git'

}

stage('maven build'){

sh '''/usr/local/maven/bin/mvn package -DskipTests -f /var/jenkins\_home/workspace/springcloud'''

}

stage('image build'){

sh '''

echo $BUILD\_ID

docker build -t 10.24.2.16/springcloud/gateway:$BUILD\_ID -f /var/jenkins\_home/workspace/springcloud/gateway/Dockerfile /var/jenkins\_home/workspace/springcloud/gateway

docker build -t 10.24.2.16/springcloud/config:$BUILD\_ID -f /var/jenkins\_home/workspace/springcloud/config/Dockerfile /var/jenkins\_home/workspace/springcloud/config'''

}

stage('test'){

sh '''docker run -itd --name gateway 10.24.2.16/springcloud/gateway:$BUILD\_ID

docker ps -a|grep springcloud|grep Up

if [ $? -eq 0 ];then

echo "Success!"

docker rm -f gateway

else

docker rm -f gateway

exit 1

fi

'''

}

stage('upload registry'){

sh '''docker login 10.24.2.16 -u=admin -p=Harbor12345

docker push 10.24.2.16/springcloud/gateway:$BUILD\_ID

docker push 10.24.2.16/springcloud/config:$BUILD\_ID'''

}

stage('deploy Rancher'){

//执行部署脚本

sh 'sed -i "s/sqshq\\/piggymetrics-gateway/10.24.2.16\\/springcloud\\/gateway:$BUILD\_ID/g" /var/jenkins\_home/workspace/springcloud/yaml/deployment/gateway-deployment.yaml'

sh 'sed -i "s/sqshq\\/piggymetrics-config/10.24.2.16\\/springcloud\\/config:$BUILD\_ID/g" /var/jenkins\_home/workspace/springcloud/yaml/deployment/config-deployment.yaml'

sh 'kubectl create ns springcloud'

sh 'kubectl apply -f /var/jenkins\_home/workspace/springcloud/yaml/deployment/gateway-deployment.yaml --kubeconfig=/root/.kube/config'

sh 'kubectl apply -f /var/jenkins\_home/workspace/springcloud/yaml/deployment/config-deployment.yaml --kubeconfig=/root/.kube/config'

sh 'kubectl apply -f /var/jenkins\_home/workspace/springcloud/yaml/svc/gateway-svc.yaml --kubeconfig=/root/.kube/config'

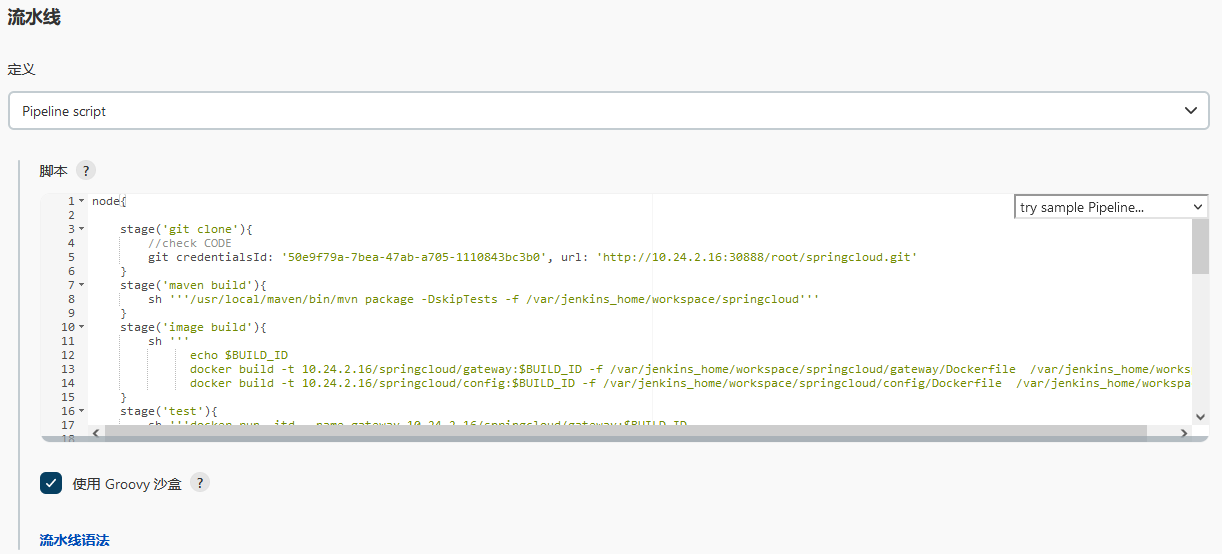
sh 'kubectl apply -f /var/jenkins\_home/workspace/springcloud/yaml/svc/config-svc.yaml --kubeconfig=/root/.kube/config'

}

}

脚本中所有IP均为Harbor仓库的地址。

在网页写入完整的流水线脚本，如图所示，完成后点击“应用”。



（2）开启Jenkins匿名访问

登录Jenkins首页，点击“系统管理”→“全局安全配置”，勾选“任何用户可以做任何事(没有任何限制)”，如图所示。



（3）创建仓库项目

登录Harbor，新建项目springcloud，访问级别设置为公开，创建完成后如图所示：



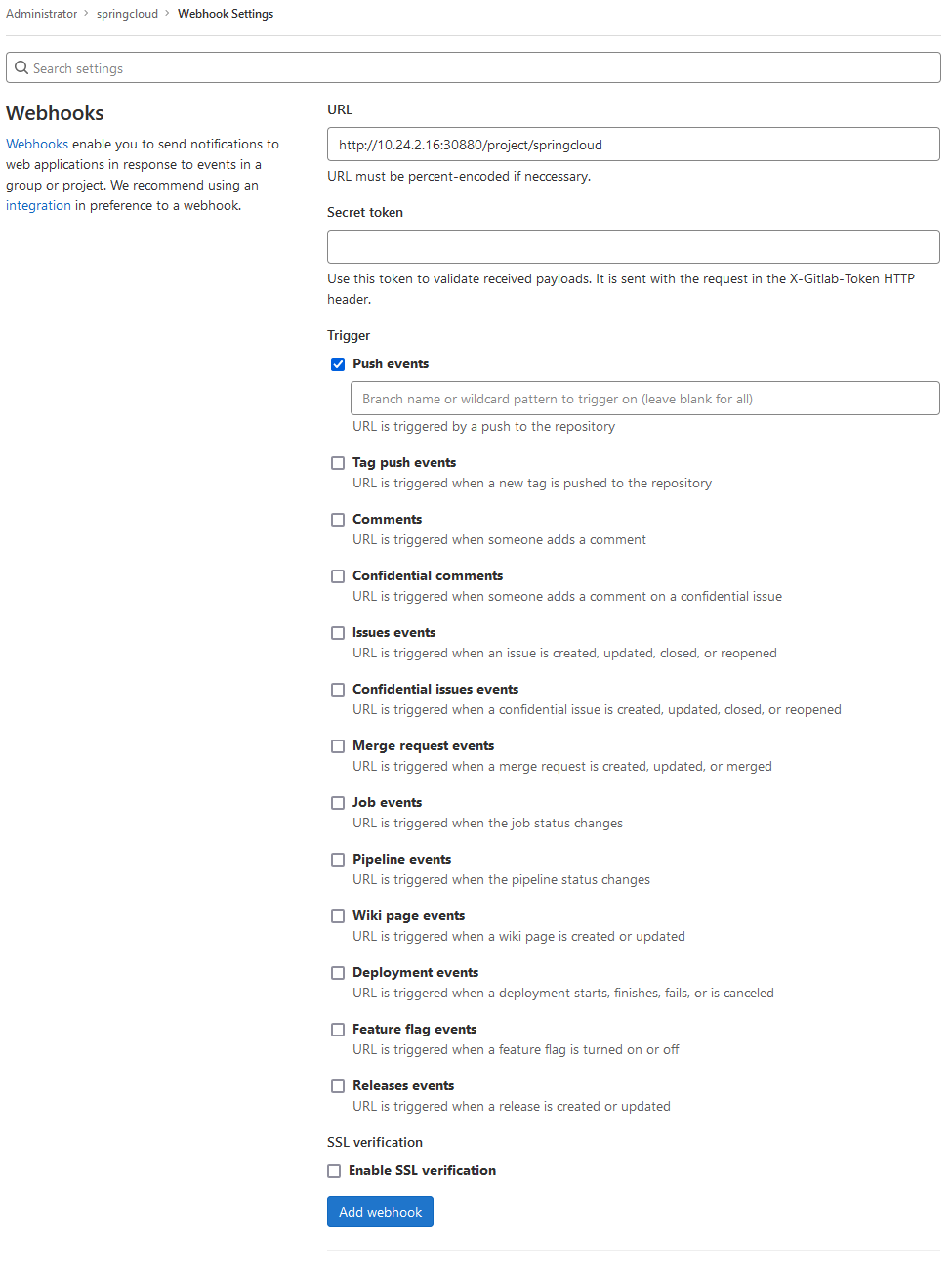
进入项目查看镜像列表，如图所示，此时为空，无任何镜像：



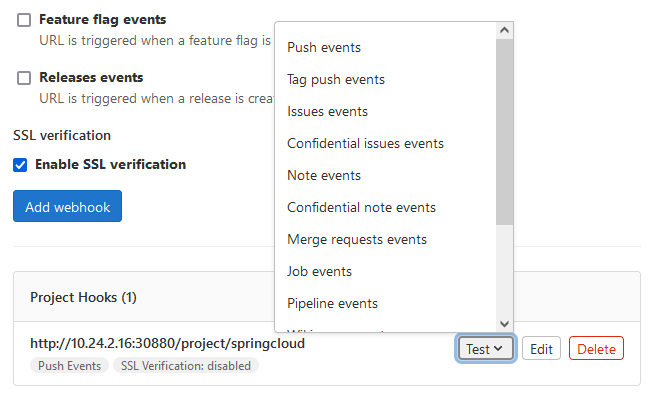
**6. 触发CI/CD**

（1）触发构建

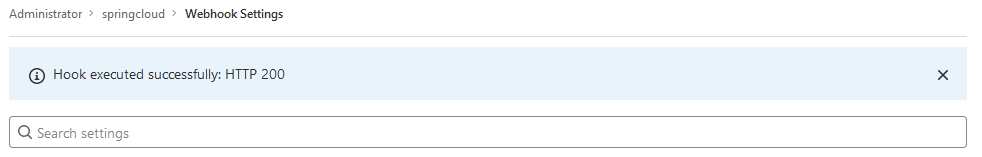
登录Gitlab，进入springcloud项目，点击左侧导航栏“Settings”→“Webhooks”，将前面记录的GitLab webhook URL地址填入URL处，禁用SSL认证，如图所示。



点击“Add webhook”添加webhook，完成后如图所示：



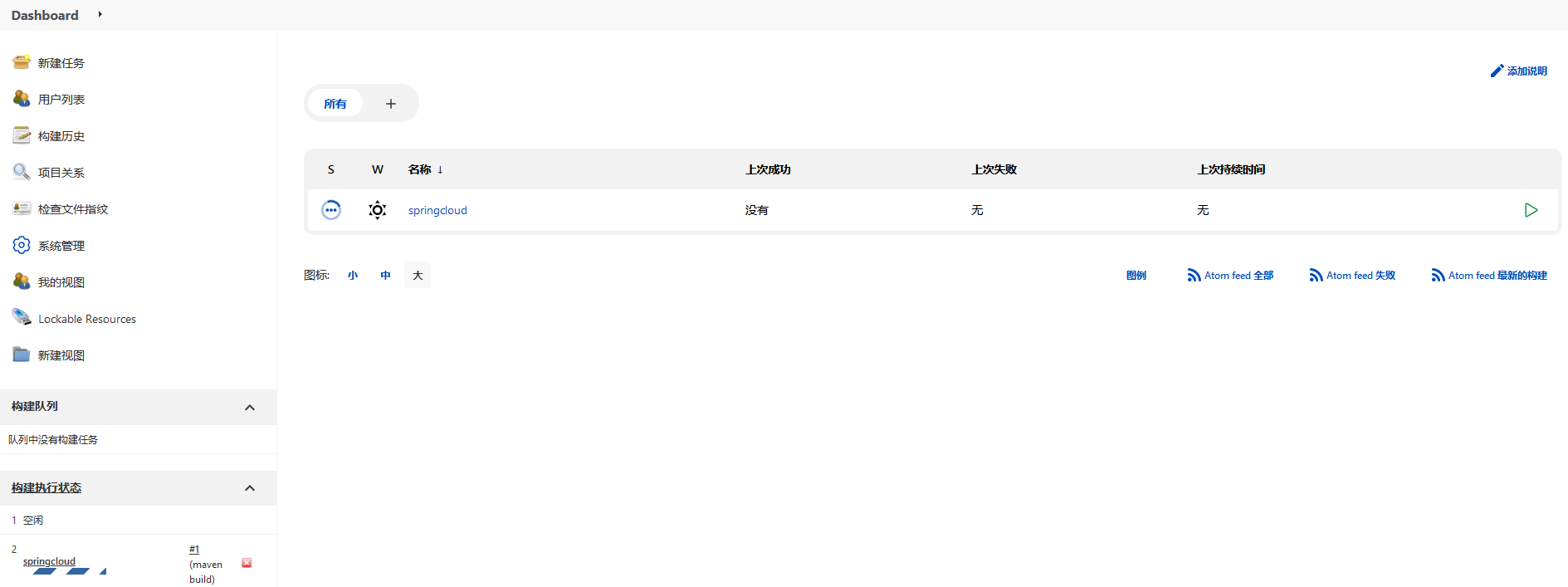
点击“Test”→“Push events”进行测试，如图所示：



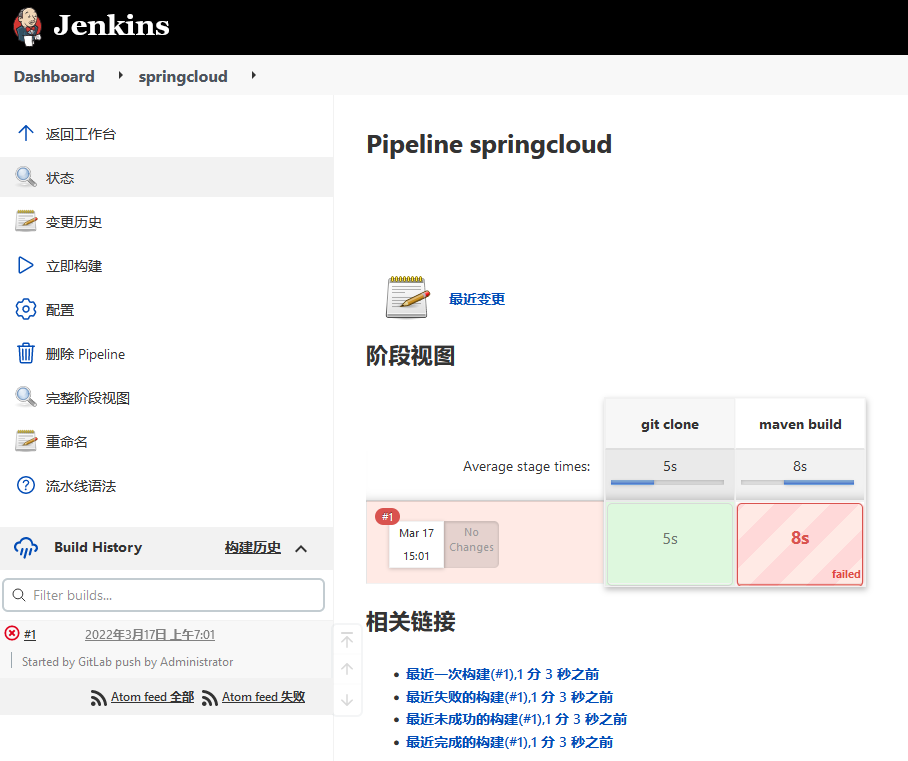
结果返回HTTP 200则表明Webhook配置成功。

（2）Jenkins查看

登录Jenkins，可以看到springcloud项目已经开始构建，如图所示：



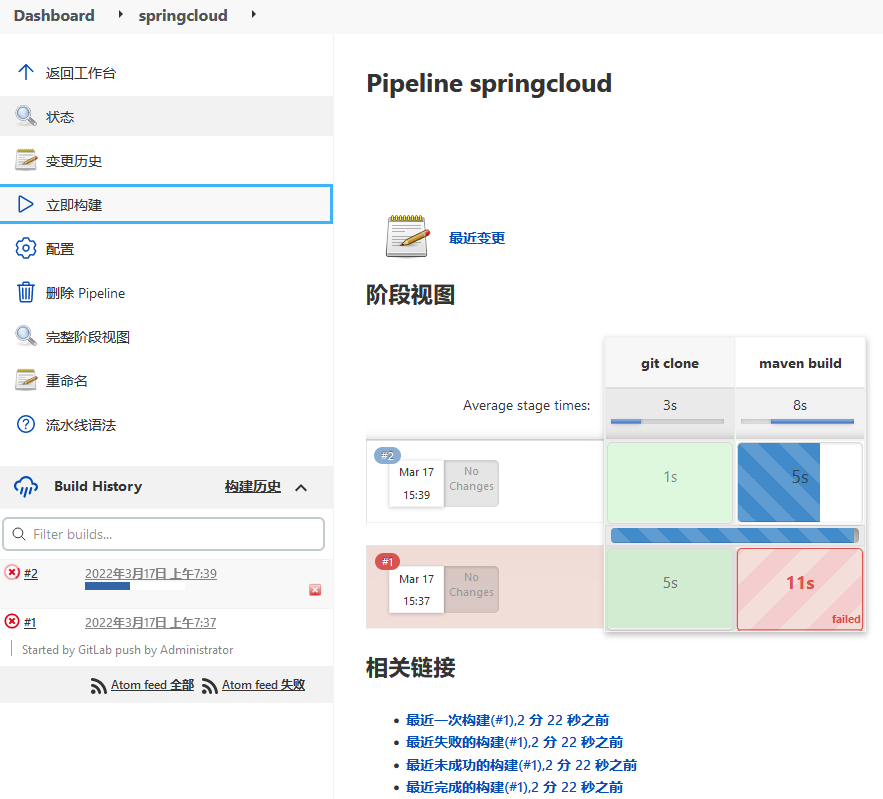
点击项目名称查看流水线阶段视图，如图所示：



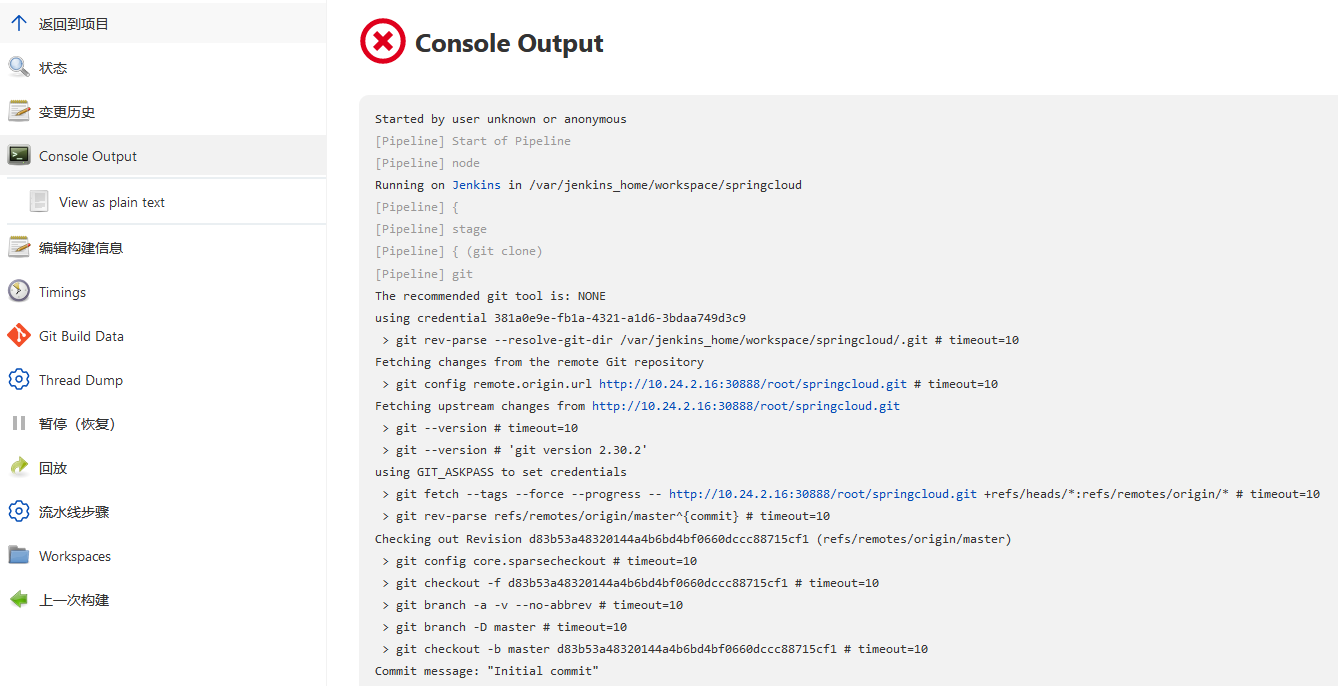
可以看到，项目构建失败，这是因为未配置Maven Repository：

[root@master Jenkins]# kubectl cp repository/ jenkins-7d9bd56966-9kd72:/root/.m2/

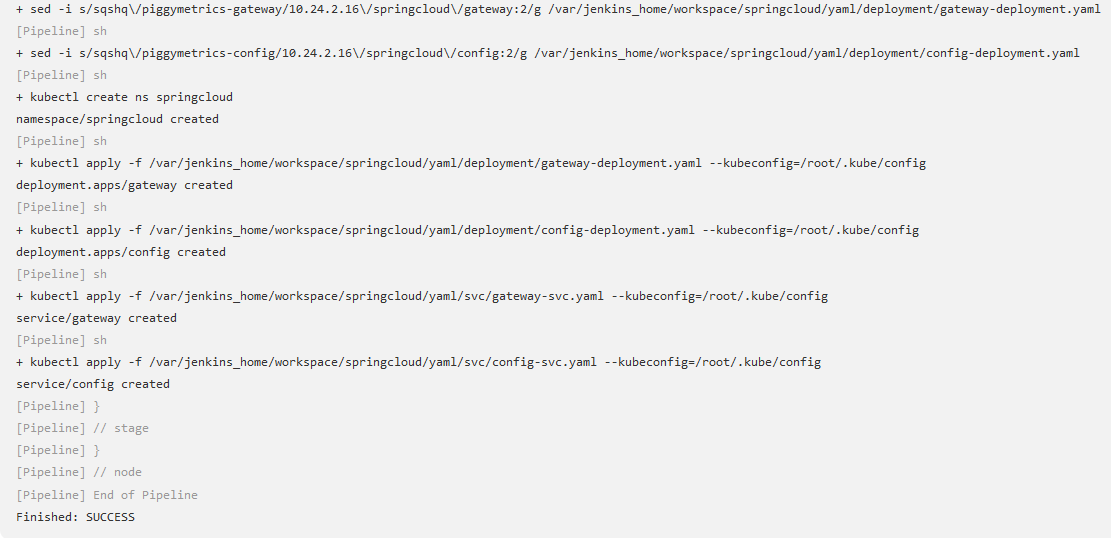
在浏览器端输入“http://master\_IP:30880/restart”重启Jenkins，重启完成后点击右侧导航栏的“立即构建”，如图所示：



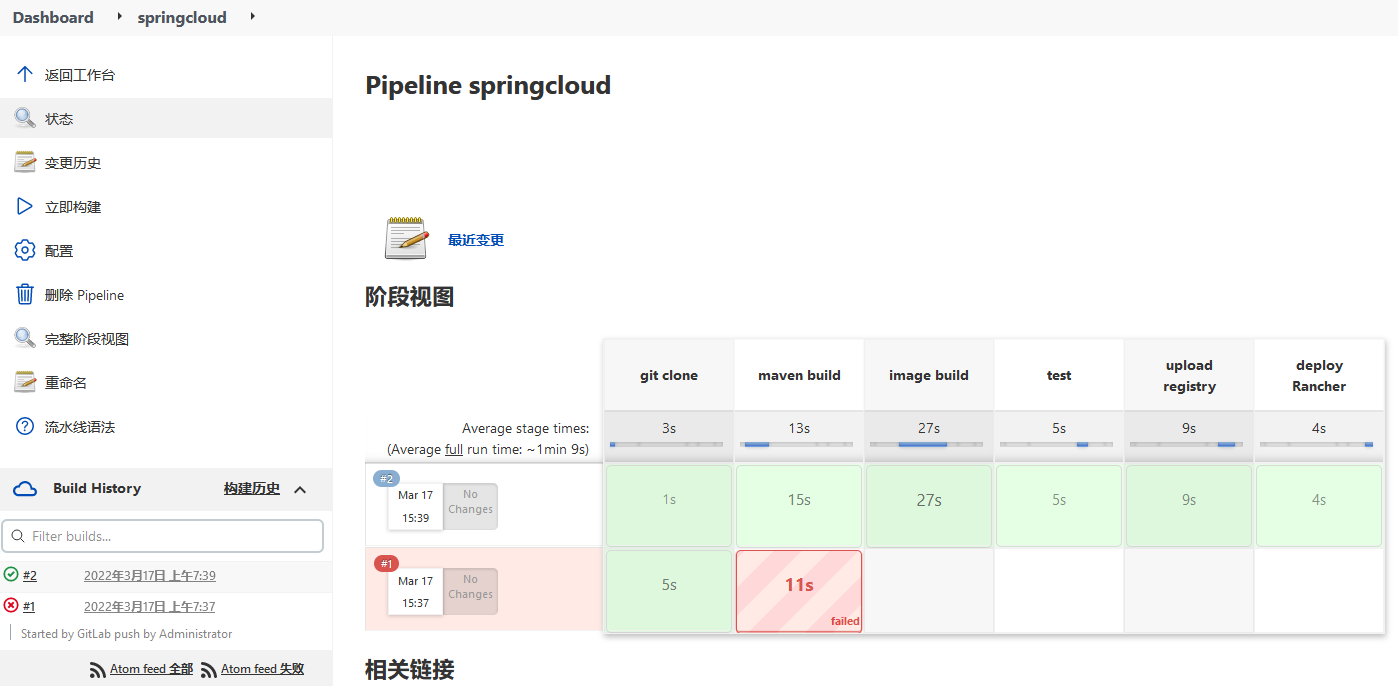
切换到“Console Output”界面可查看控制台输出，此处会显示构建的详细进程，如图所示：



构建完成后控制台输出如图所示：



返回项目查看流水线阶段视图，如图所示：



（3）Harbor查看

进入Harbor仓库springcloud项目查看镜像列表，可以看到已自动上传了一个gateway镜像，如图所示：



（4）Kubernetes查看

Pod的启动较慢，需等待3--5分钟。在命令行查看Pod：

[root@master ~]# kubectl -n springcloud get pods

NAME READY STATUS RESTARTS AGE

config-6c988c4dc5-2522c 1/1 Running 0 21m

gateway-6545fc58c5-d6rgn 1/1 Running 0 21m

查看service：

[root@master ~]# kubectl -n springcloud get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

config NodePort 10.101.42.47 <none> 8888:30015/TCP 22m

gateway NodePort 10.100.62.39 <none> 4000:30010/TCP 22m

通过端口30010访问服务，如图所示：



至此，完整的CI/CD流程就完成了。