# 项目规范

目录

[项目规范 1](#_Toc63685928)

[1、创建用户 1](#_Toc63685929)

[2、gitlab要求（以专家库项目为例，项目缩写：mall） 4](#_Toc63685930)

[2.1创建Dockerfile文件： 5](#_Toc63685931)

[2.2创建Jenkinsfile文件： 5](#_Toc63685932)

[2.3创建部署的yaml文件 8](#_Toc63685933)

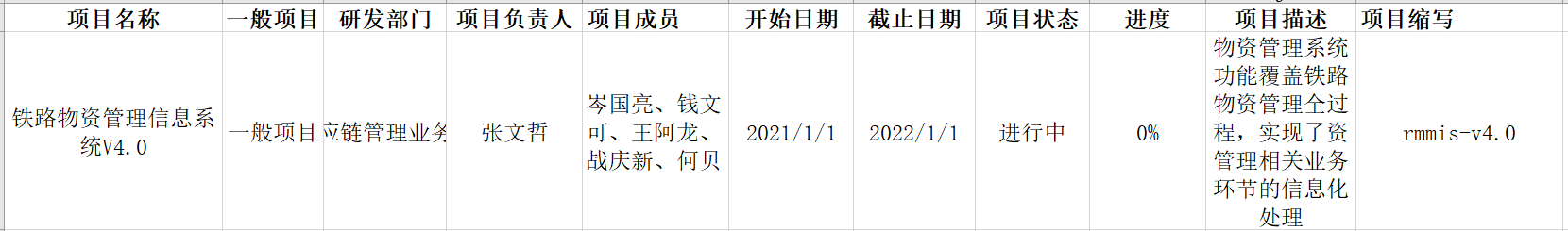
[3、Jenkins要求 11](#_Toc63685934)

[4、k8s要求 12](#_Toc63685935)

[5、harbor要求 12](#_Toc63685936)

[6、sonarqube 13](#_Toc63685937)

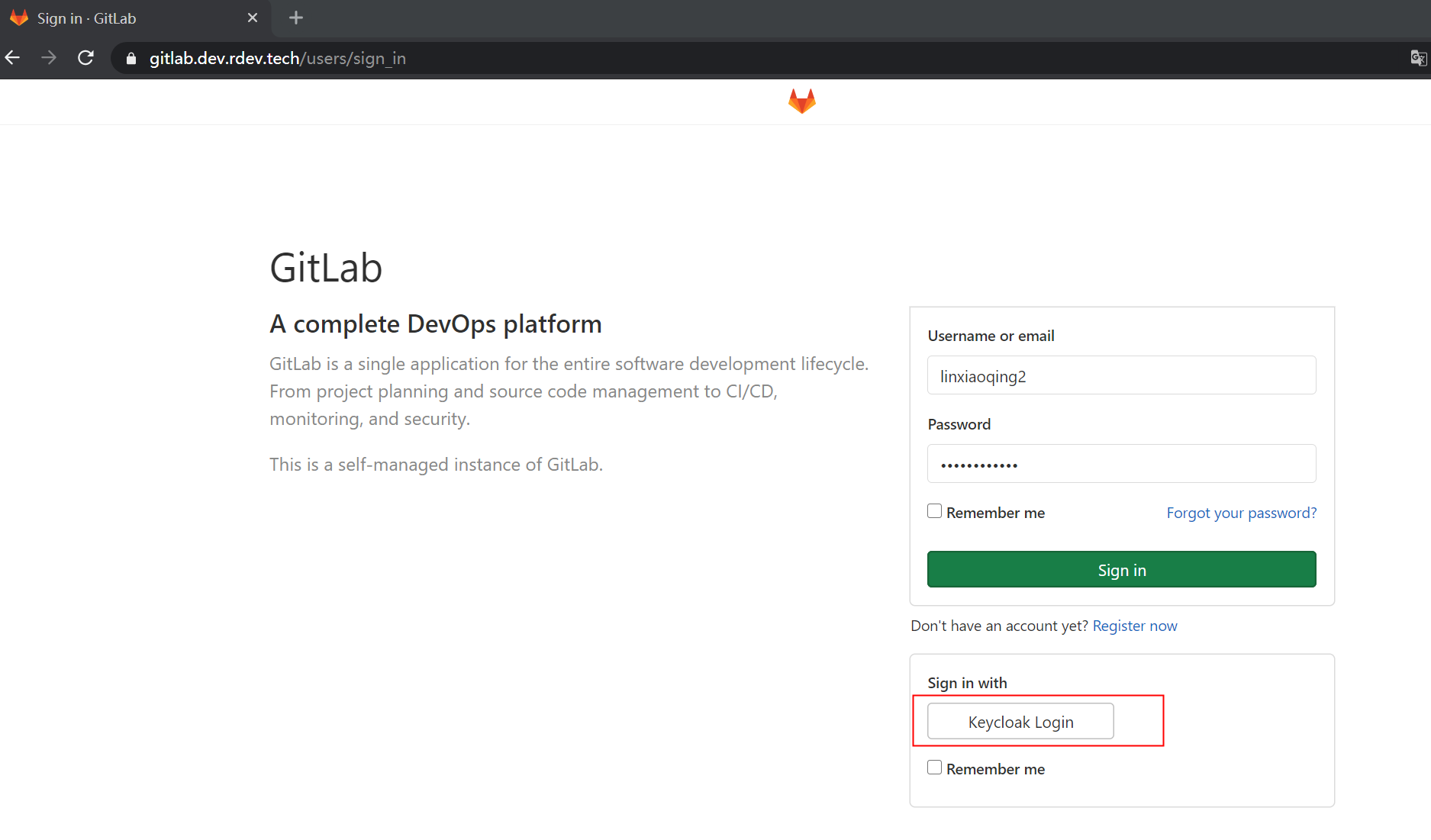
## 1、创建用户



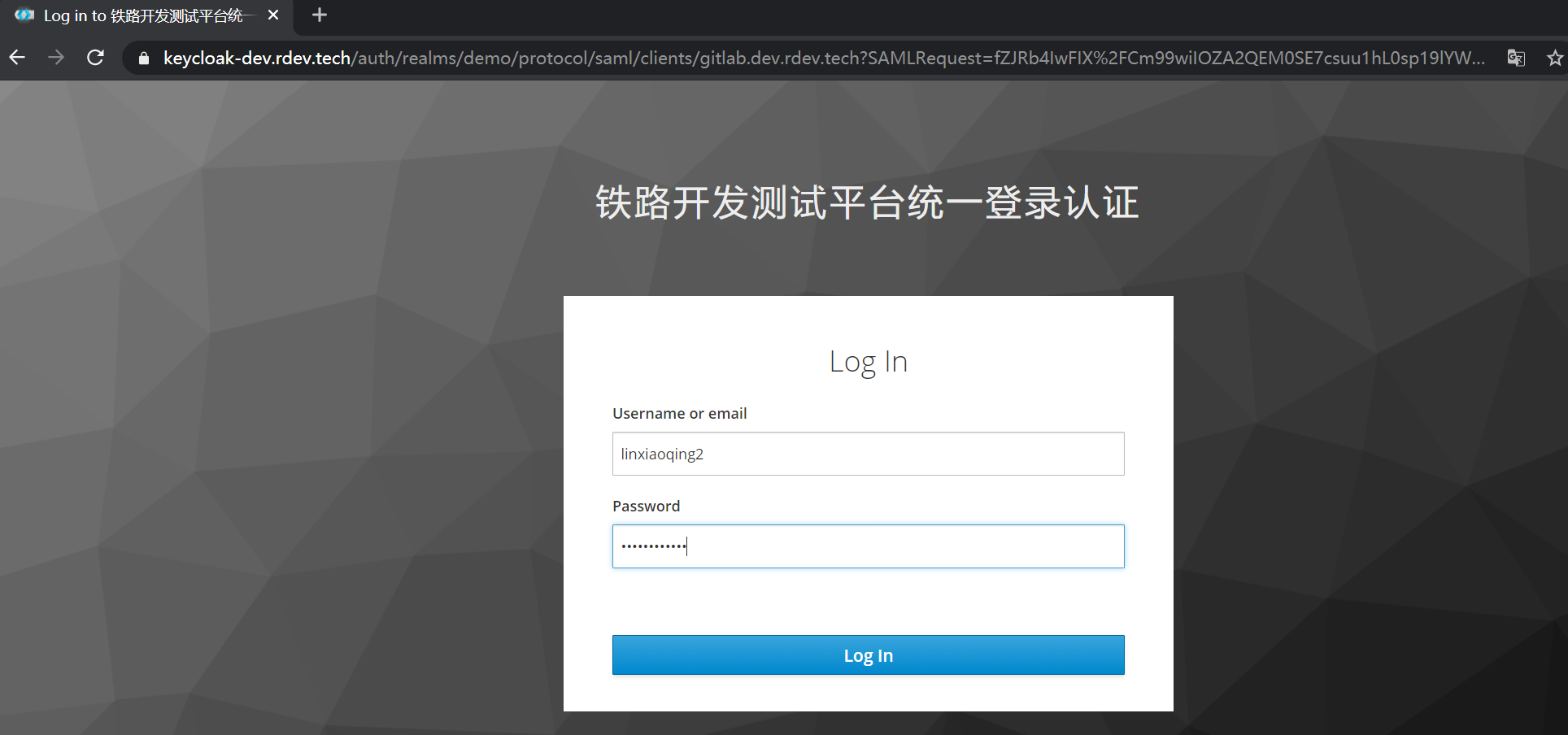
根据反馈的项目人员名单创建账号（默认用户名和密码均为拼音全称，如zhangwenzhe/zhangwenzhe）后，需要每个开发人员登录gitlab：<https://gitlab.dev.rdev.tech/>

，首次登录流程如下：

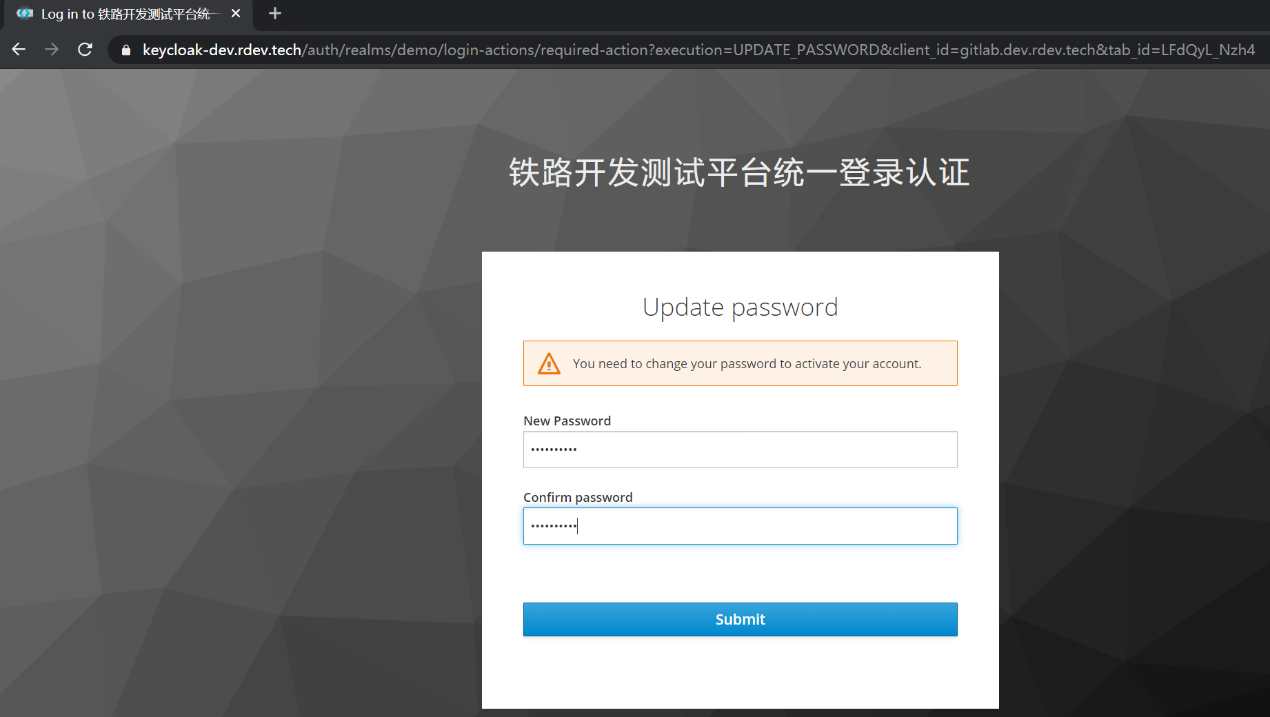
首次登录，选择Keycloak Login，



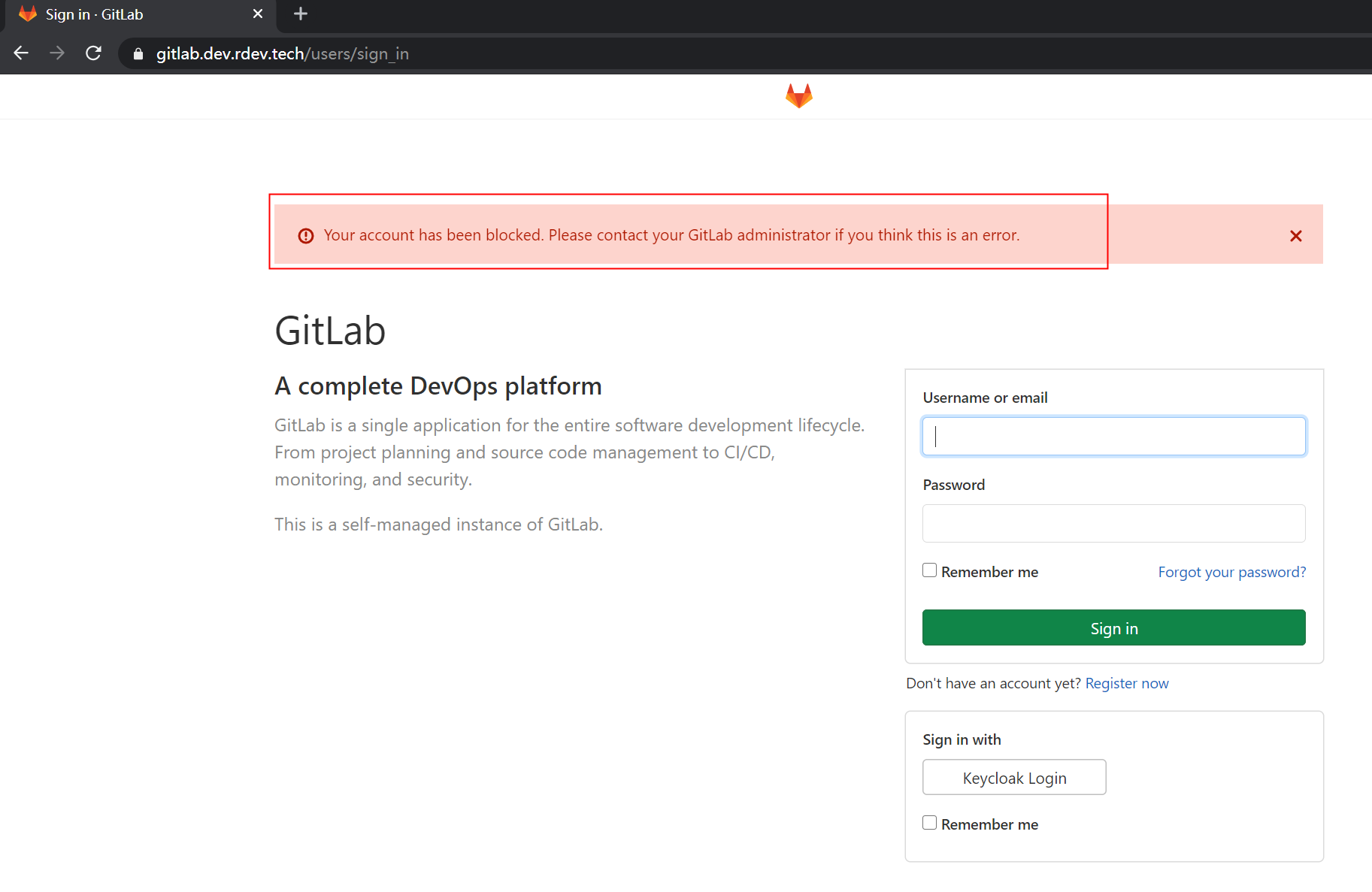
会跳转到keycloak页面，这里一定要手动输入，不要copy，



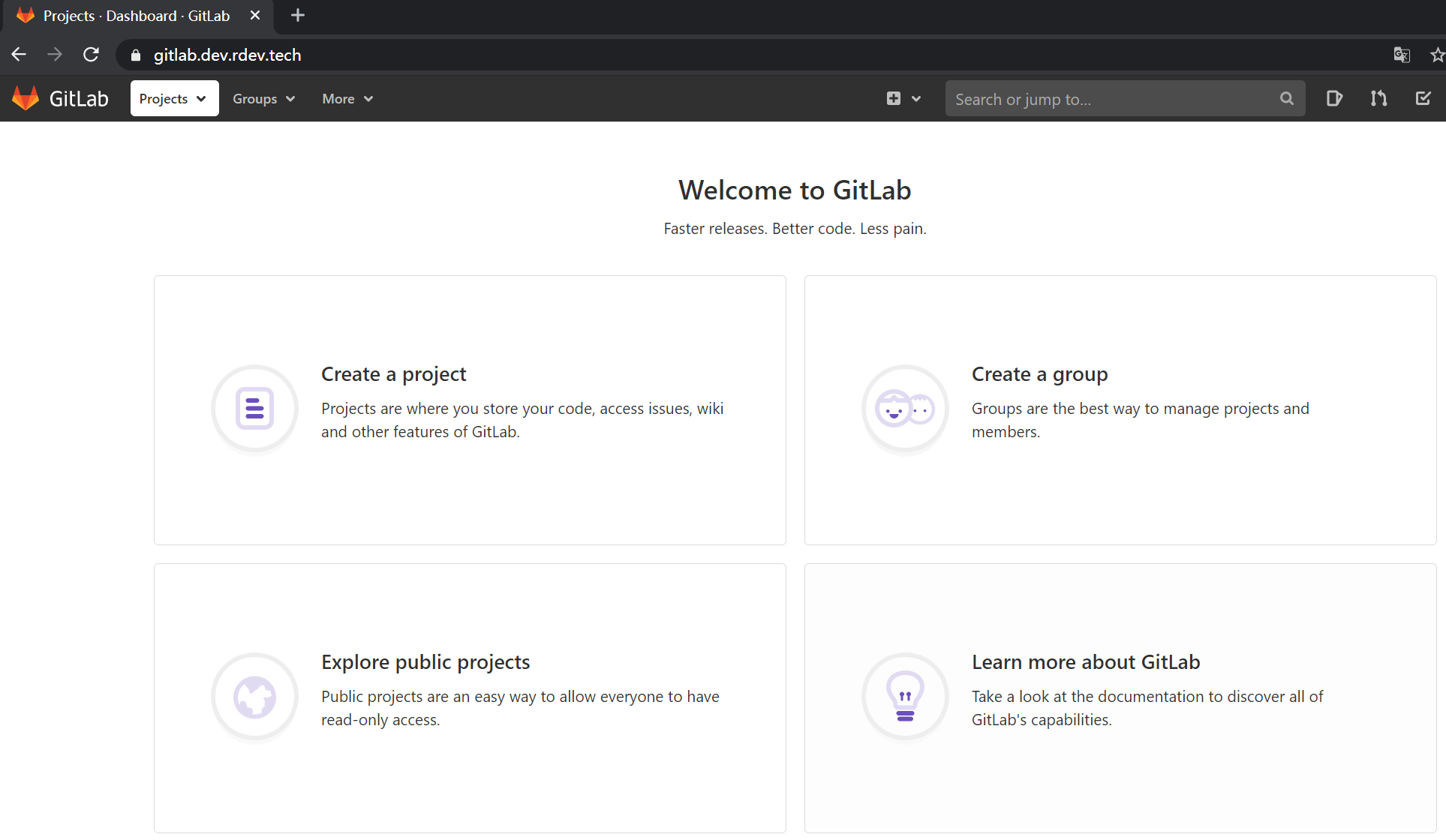
登录后提示修改密码：



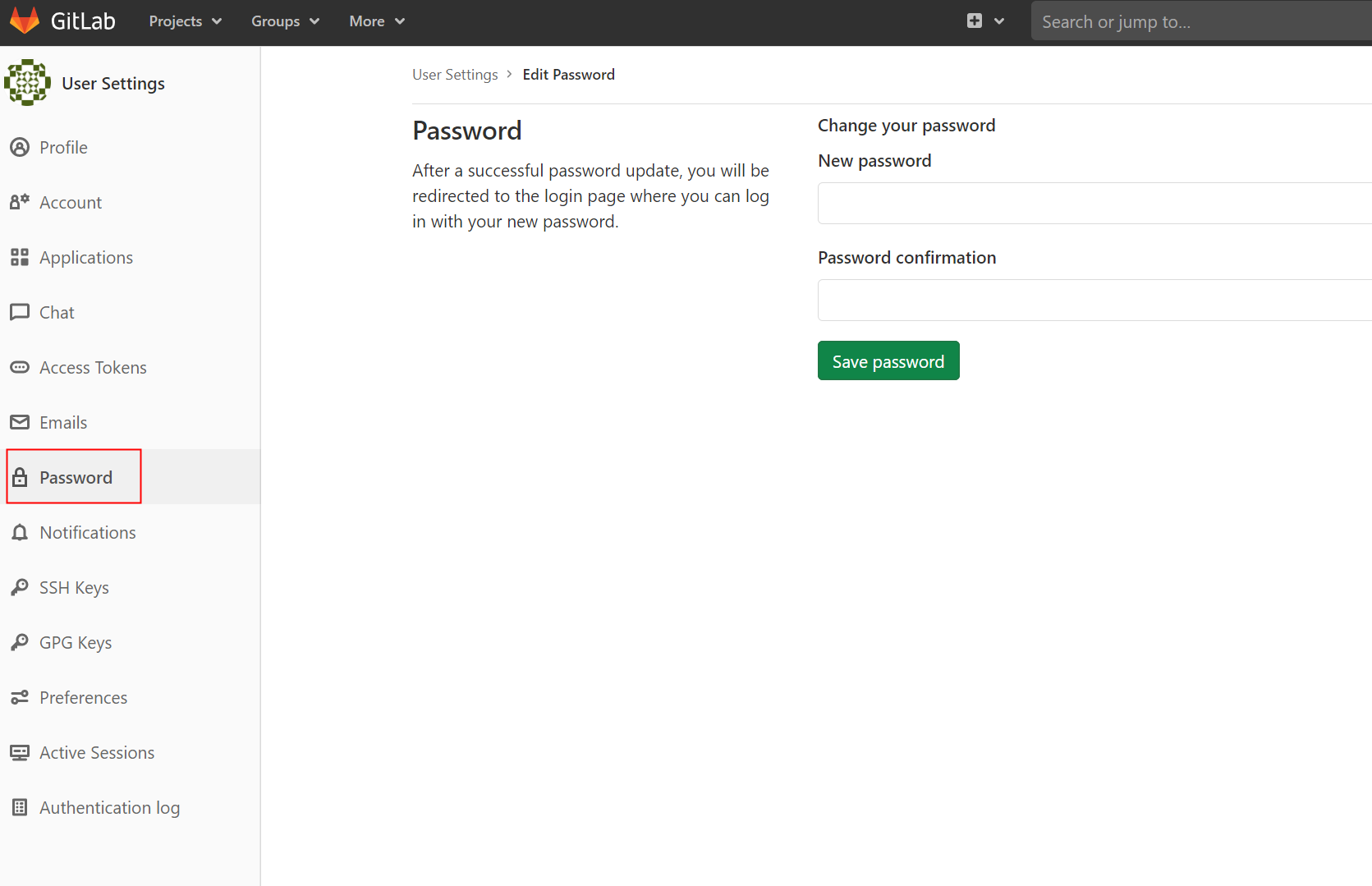
然后用修改后的密码登录，提示用户被锁定，



联系管理员解锁，解锁后即可登录gitlab：

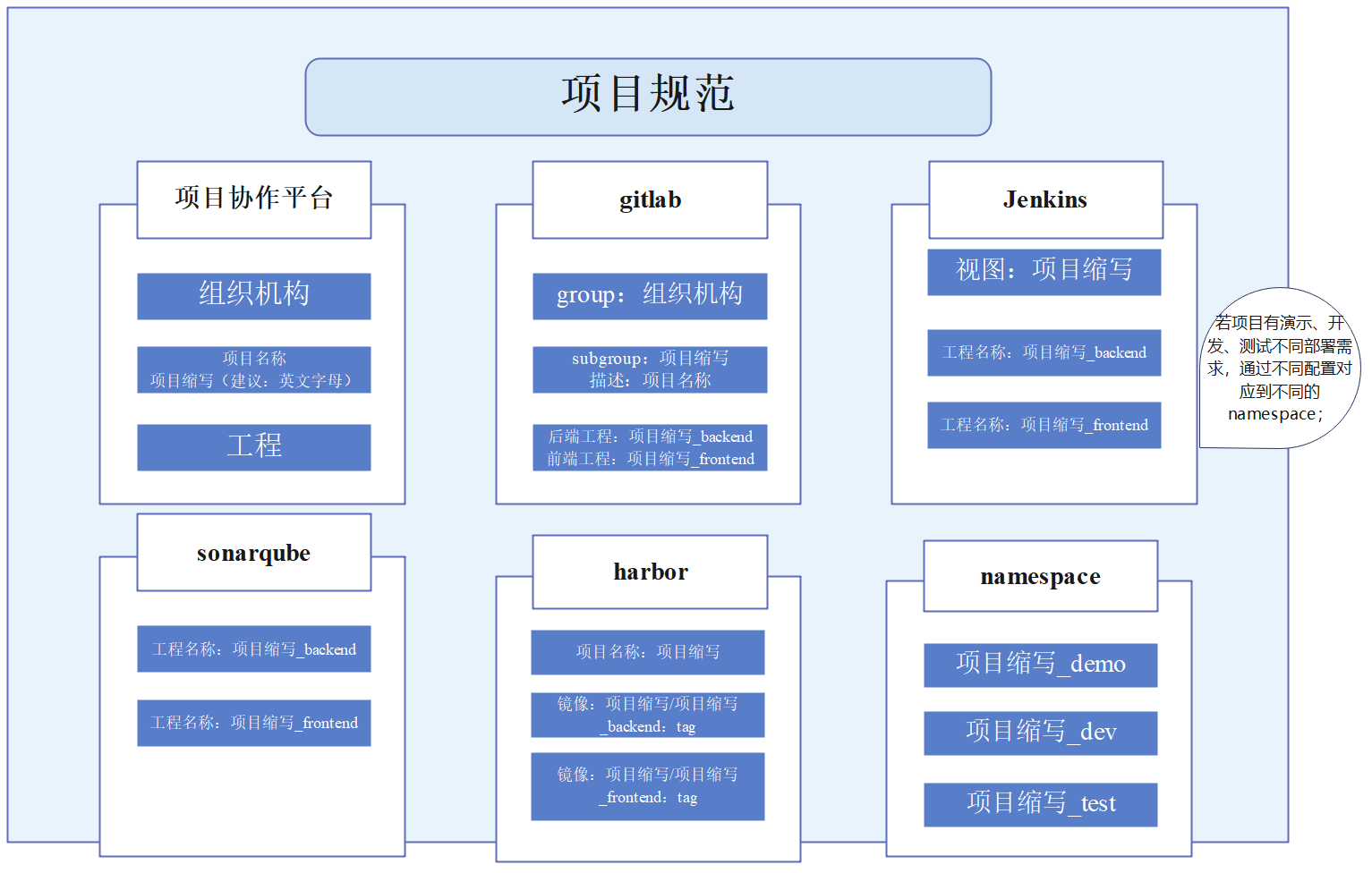


登录gitlab后设置密码，与keycloak的密码设置一致；（若不设置gitlab密码，会导致无法下载代码）



根据表格，项目负责人会赋给Maintainer角色，其他组员均为Developer角色。

至此，即可通过keycloak实现单点登录gitlab。再设置sshkey等。



## 2、gitlab要求（以专家库项目为例，项目缩写：mall）

（1）以项目缩写mall为subgroup名称，创建在以组织机构为Group的大group下；有特殊要求的可以继续建立subgroup；

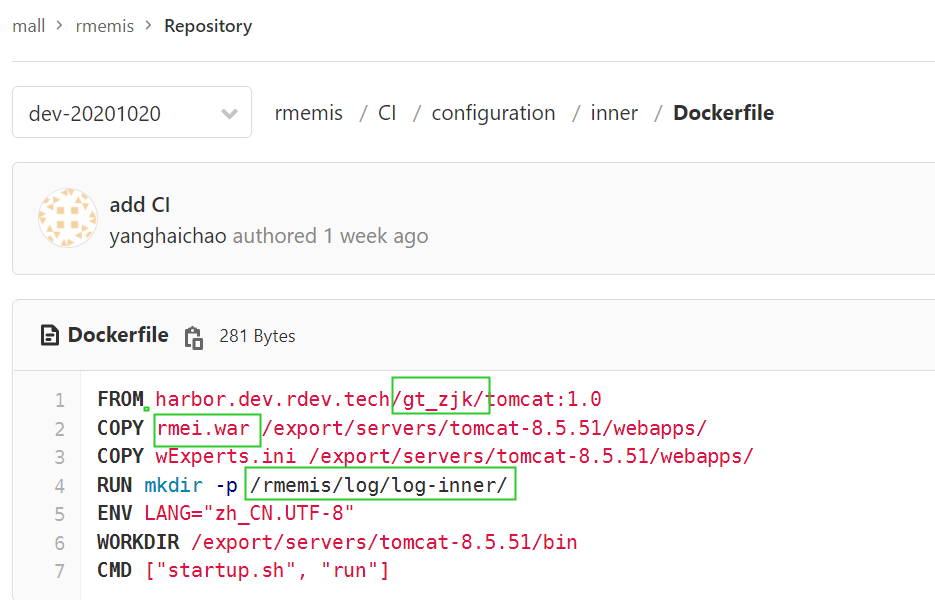
（2）在subgroup下创建工程：项目命名，后端命名类似\*\*\_backend，前端命名类似\*\*\_frontend；

（3）需要创建CD的部署工程；

（4）在代码库的工程创建CI/configuration/项目模块/Jenkinsfile；

### 2.1创建Dockerfile文件：

在对应的分支下创建Dockerfile文件，建议路径为：项目缩写/工程名称/CI/configuration/模块名称/Dockerfile



其中，

第1行：为harbor上对应项目的基础镜像；

第2~7行，为制作版本镜像的流程，需要根据特定版本进行定制；

FROM harbor.dev.rdev.tech/gt\_zjk/tomcat:1.0

COPY rmei.war /export/servers/tomcat-8.5.51/webapps/

COPY wExperts.ini /export/servers/tomcat-8.5.51/webapps/

RUN mkdir -p /rmemis/log/log-inner/

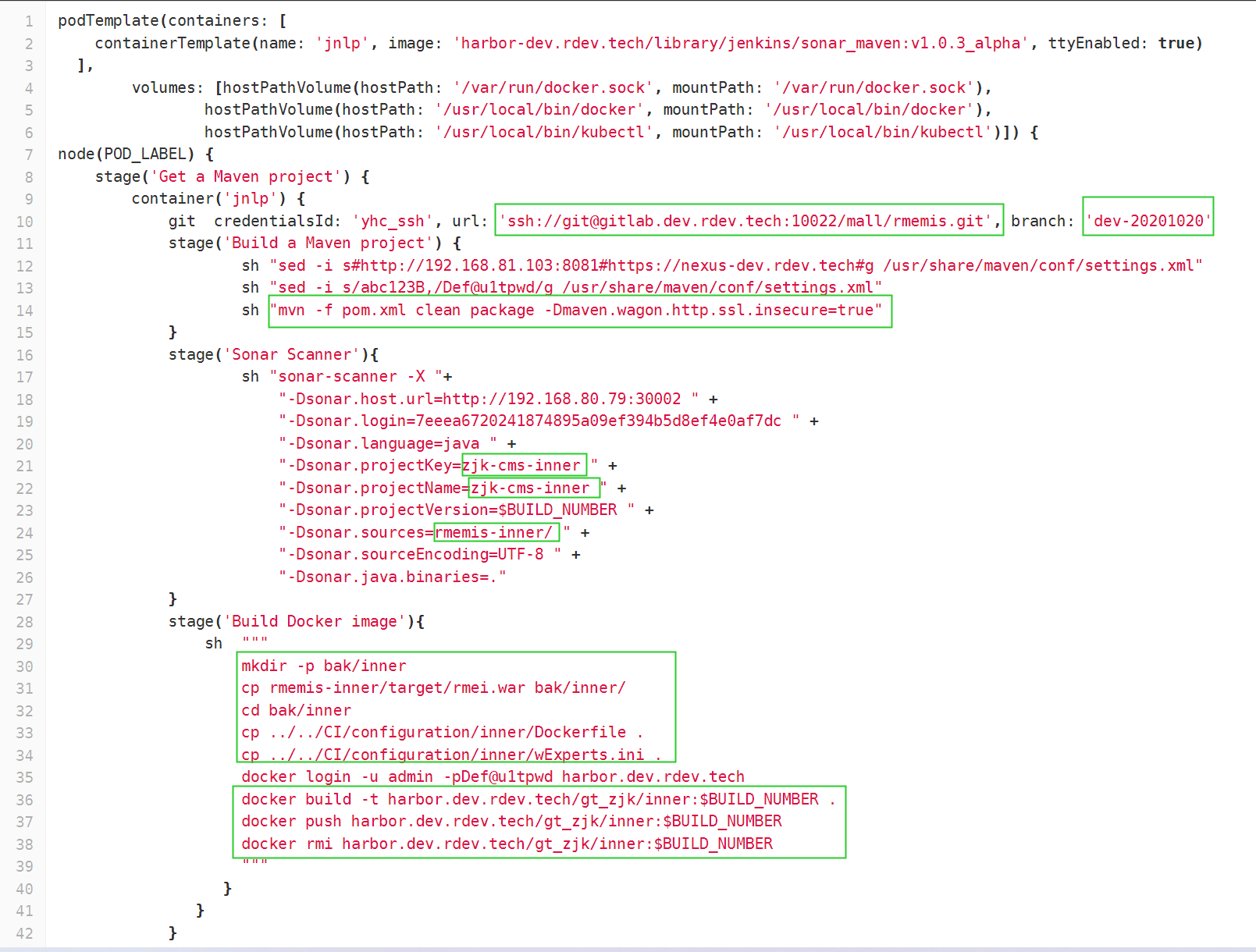
ENV LANG="zh\_CN.UTF-8"

WORKDIR /export/servers/tomcat-8.5.51/bin

CMD ["startup.sh", "run"]

### 2.2创建Jenkinsfile文件：

在对应的分支下创建Dockerfile文件，建议路径为：项目缩写/工程名称/CI/configuration/模块名称/Jenkinsfile



其中：

第10行：代码git仓库地址，需要构建的分支；

第14行：打包方式；

第21~24行：对应sonarqube上的工程名称；

第30~34行：制作镜像的准备工作，需要修改为对应项目的Dockerfile和配置文件所在路径；

第36~38行：制作镜像并push到harbor上，需要修改为对应项目的镜像路径；

podTemplate(containers: [

containerTemplate(name: 'jnlp', image: 'harbor-dev.rdev.tech/library/jenkins/sonar\_maven:v1.0.3\_alpha', ttyEnabled: true)

],

volumes: [hostPathVolume(hostPath: '/var/run/docker.sock', mountPath: '/var/run/docker.sock'),

hostPathVolume(hostPath: '/usr/local/bin/docker', mountPath: '/usr/local/bin/docker'),

hostPathVolume(hostPath: '/usr/local/bin/kubectl', mountPath: '/usr/local/bin/kubectl')]) {

node(POD\_LABEL) {

stage('Get a Maven project') {

container('jnlp') {

git credentialsId: 'yhc\_ssh', url: 'ssh://git@gitlab.dev.rdev.tech:10022/mall/rmemis.git', branch: 'dev-20201020'

stage('Build a Maven project') {

sh "sed -i s#http://192.168.81.103:8081#https://nexus-dev.rdev.tech#g /usr/share/maven/conf/settings.xml"

sh "sed -i s/abc123B,/Def@u1tpwd/g /usr/share/maven/conf/settings.xml"

sh "mvn -f pom.xml clean package -Dmaven.wagon.http.ssl.insecure=true"

}

stage('Sonar Scanner'){

sh "sonar-scanner -X "+

"-Dsonar.host.url=http://192.168.80.79:30002 " +

"-Dsonar.login=7eeea6720241874895a09ef394b5d8ef4e0af7dc " +

"-Dsonar.language=java " +

"-Dsonar.projectKey=zjk-cms-inner " +

"-Dsonar.projectName=zjk-cms-inner " +

"-Dsonar.projectVersion=$BUILD\_NUMBER " +

"-Dsonar.sources=rmemis-inner/ " +

"-Dsonar.sourceEncoding=UTF-8 " +

"-Dsonar.java.binaries=."

}

stage('Build Docker image'){

sh """

mkdir -p bak/inner

cp rmemis-inner/target/rmei.war bak/inner/

cd bak/inner

cp ../../CI/configuration/inner/Dockerfile .

cp ../../CI/configuration/inner/wExperts.ini .

docker login -u admin -pDef@u1tpwd harbor.dev.rdev.tech

docker build -t harbor.dev.rdev.tech/gt\_zjk/inner:$BUILD\_NUMBER .

docker push harbor.dev.rdev.tech/gt\_zjk/inner:$BUILD\_NUMBER

docker rmi harbor.dev.rdev.tech/gt\_zjk/inner:$BUILD\_NUMBER

"""

}

}

}

}

}

### 2.3创建部署的yaml文件

（1）CD/模块名称/\*\*\*\*.yaml

（2）其中，

namespace：对应k8s上的namespace；

app：为对应的服务名称，以项目为准；

image：为harbor上的镜像名称；



apiVersion: v1

kind: Service

metadata:

name: inner

namespace: zjk

labels:

app: inner

group: dev

spec:

type: NodePort

ports:

- name: http

port: 8001

selector:

app: inner

group: dev

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: inner

namespace: zjk

spec:

replicas: 1

strategy:

type: RollingUpdate

selector:

matchLabels:

app: inner

group: dev

template:

metadata:

labels:

app: inner

group: dev

spec:

containers:

- name: inner

image: harbor.dev.rdev.tech/gt\_zjk/inner:481

imagePullPolicy: Always

ports:

- name: http

containerPort: 8001

hostPort: 8001

protocol: TCP

args:

- -c

- sh startup.sh;while true;do echo running;sleep 10;done

command:

- /bin/bash

volumeMounts:

- name: inner-logs

mountPath: /rmemis

volumes:

- name: inner-logs

persistentVolumeClaim:

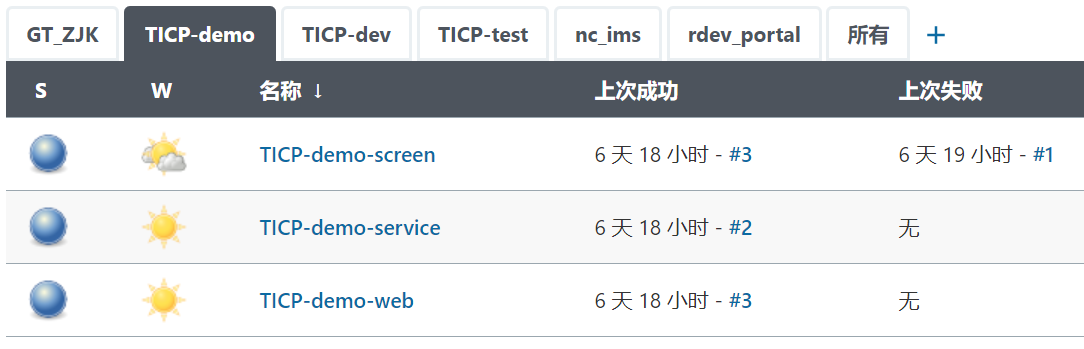
claimName: inner-logs

## 3、Jenkins要求

（1）创建视图名称要求：以subgroup为视图标签；

若项目有演示、开发、测试，多个工程需求，则创建的视图名称可设置为：项目缩写-demo/dev/test；

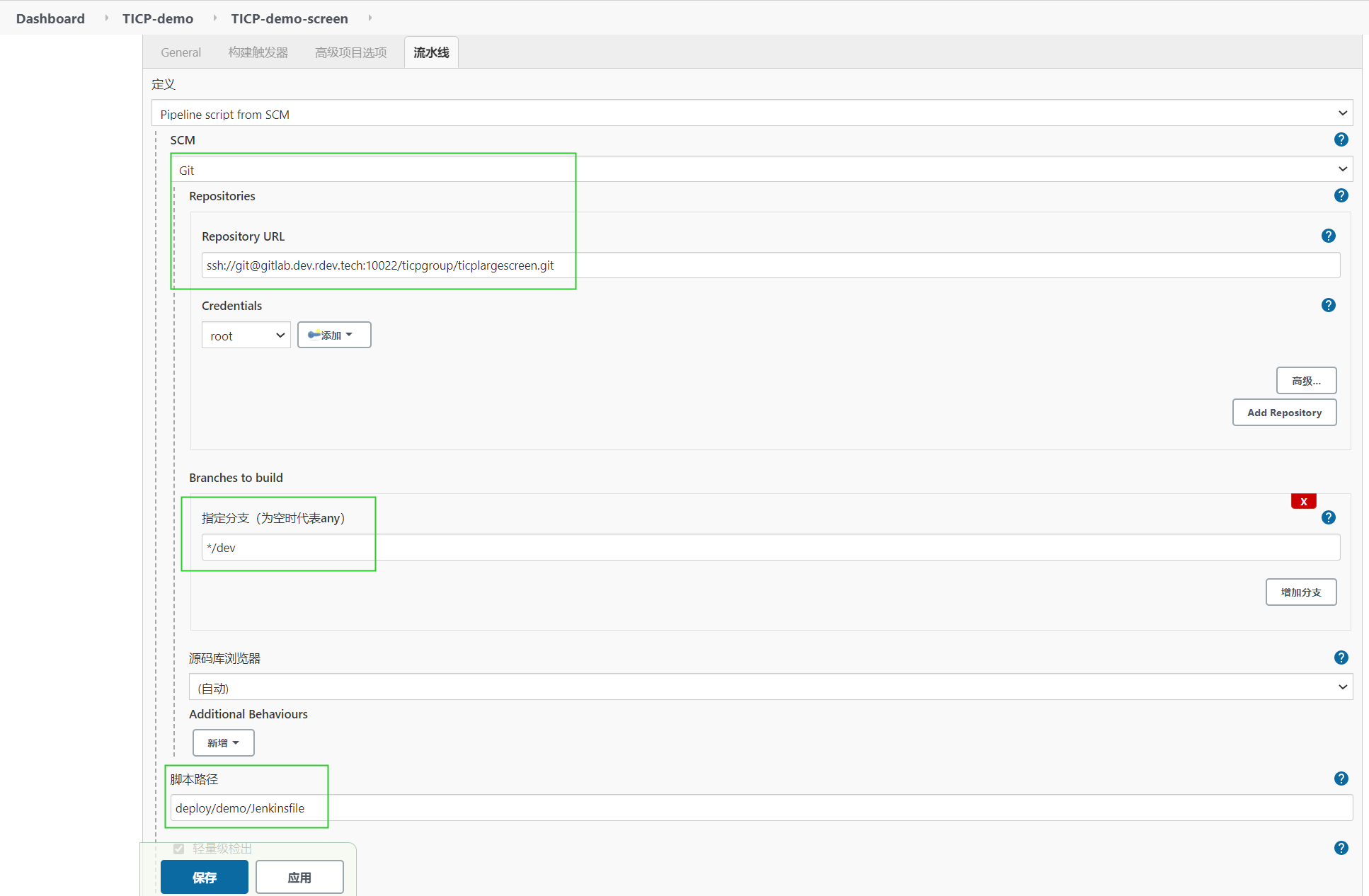
（2）视图下创建任务名称要求：项目缩写-demo/dev/test -模块名称；



（3）流水线配置：

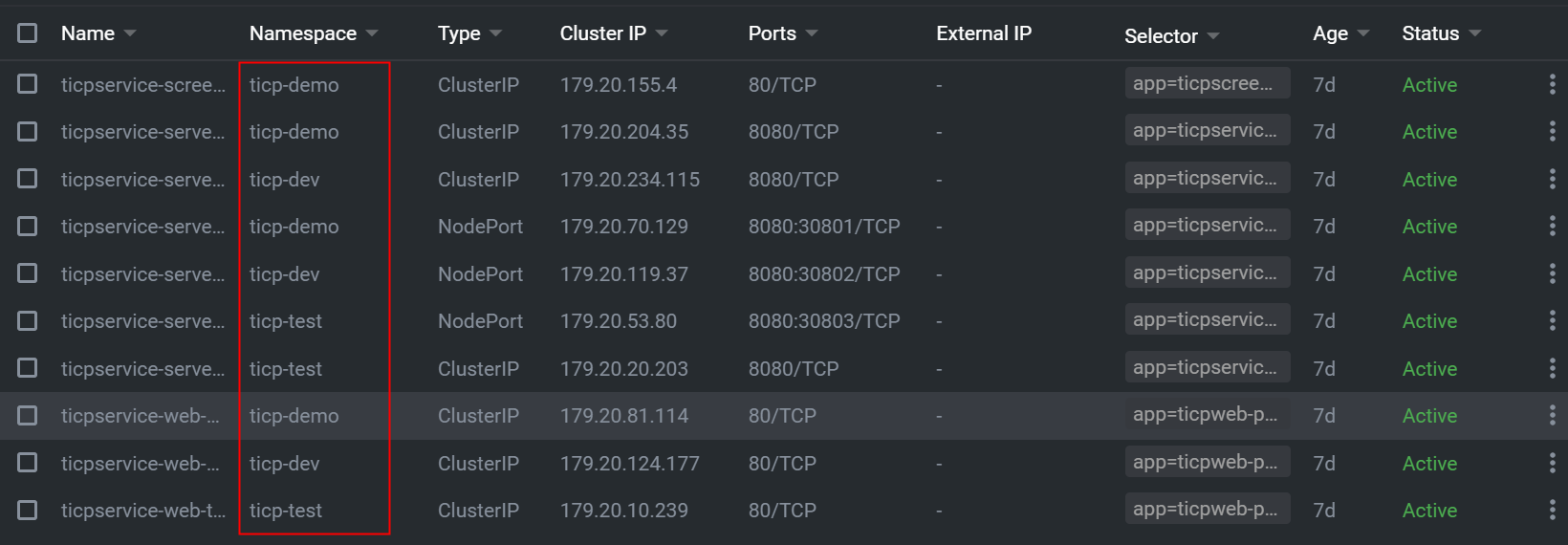
Pipeline script from SCM

指定代码仓库地址，构建的分支，Jenkinsfile文件路径



## 4、k8s要求

（1）namespace名称要求：若项目有演示、开发、测试，多个工程需求，则创建的视图名称可设置为：项目缩写-demo/dev/test；



## 5、harbor要求

（1）镜像创建路径要求：

harbor.dev.rdev.tech/项目缩写/镜像名称:tag

如：

[harbor.dev.rdev.tech/ticpservice/ticpservice-server-demo@sha256:caa3d68416c160923815e25c2111290c36bb862ac1bd209df003b77256a45f58](mailto:harbor.dev.rdev.tech/ticpservice/ticpservice-server-demo@sha256:caa3d68416c160923815e25c2111290c36bb862ac1bd209df003b77256a45f58)

## 6、sonarqube

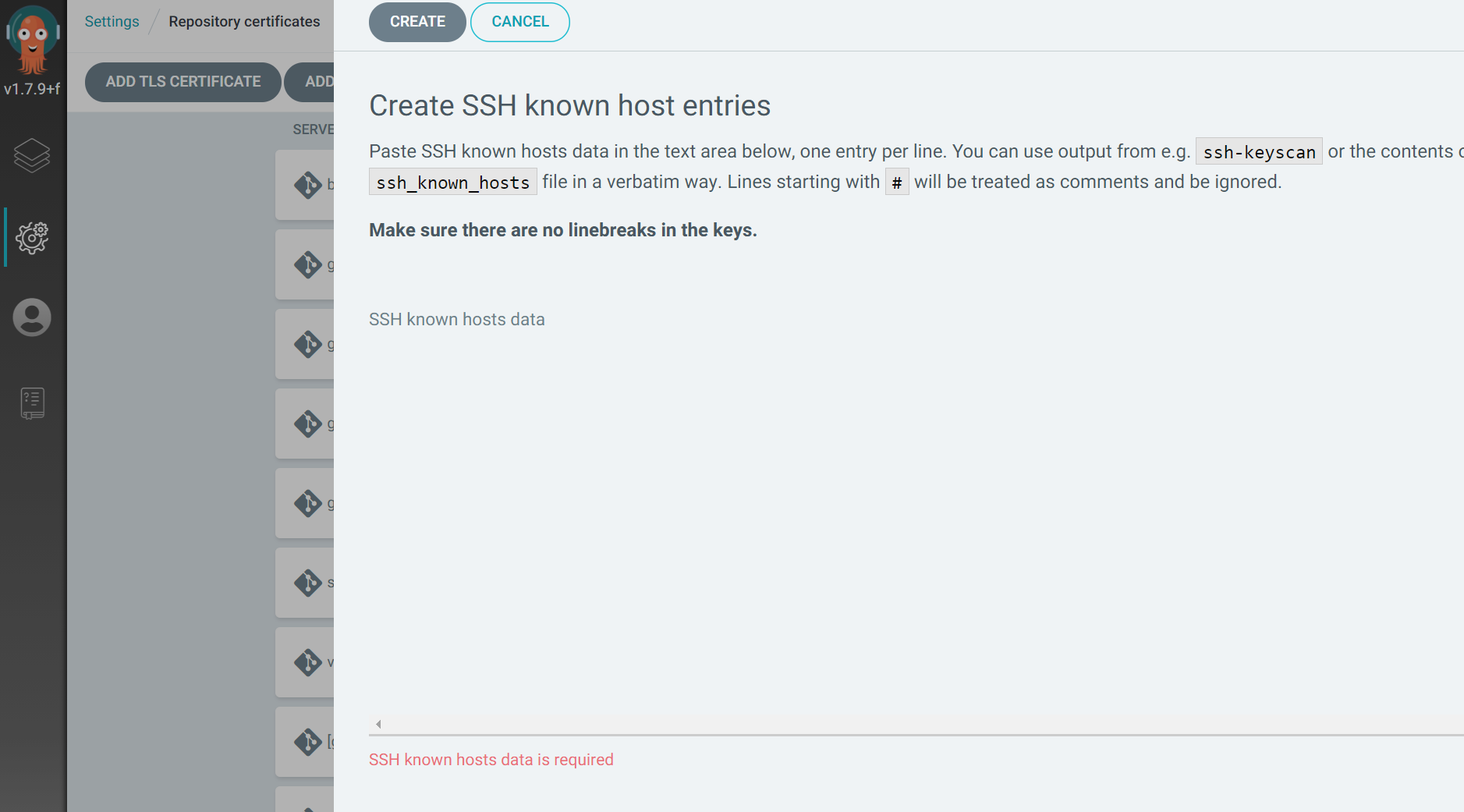
（1）工程名称要求：

sonarqube.dev.rdev.tech/项目缩写-demo

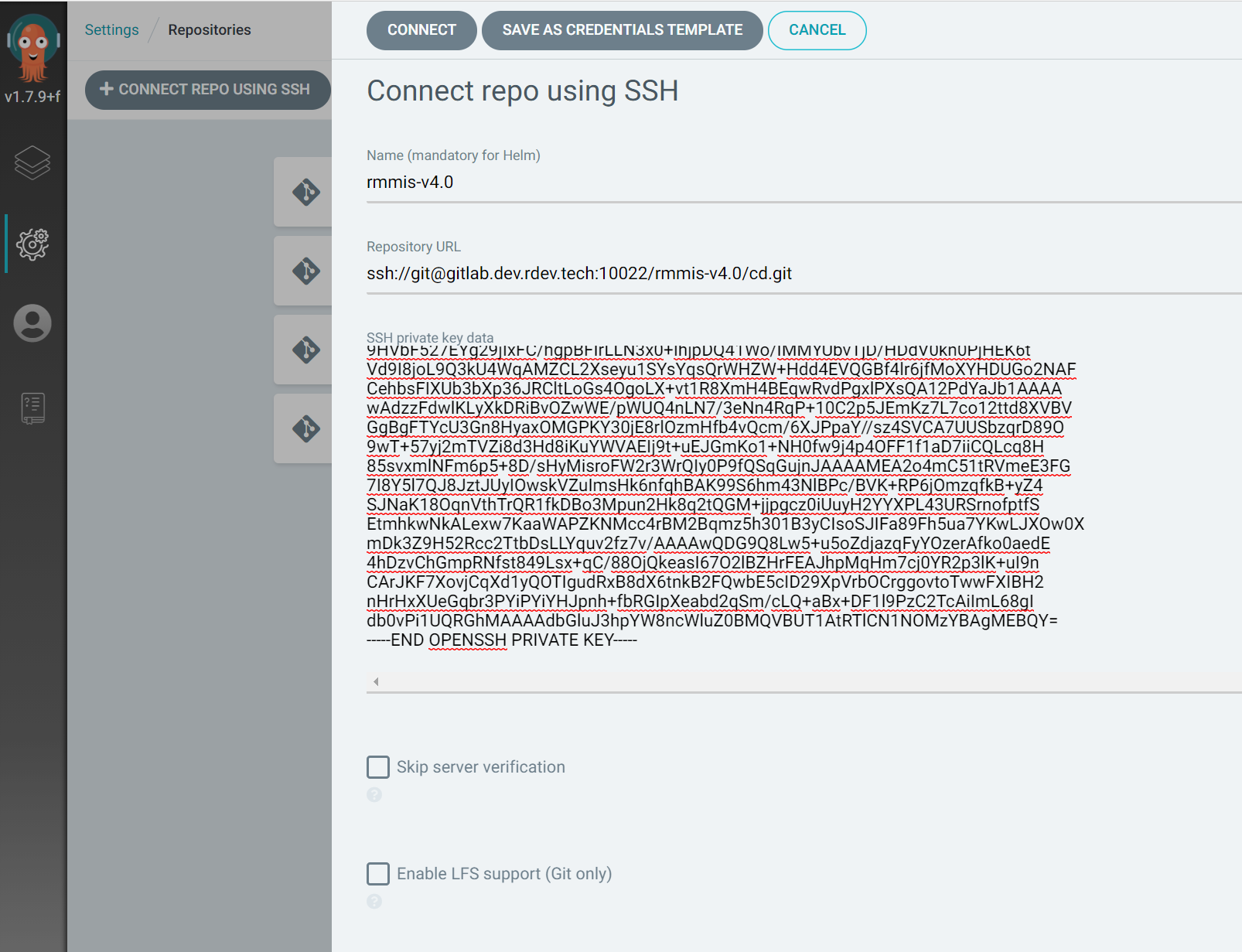
## 7.argocd配置

### 7.1设置SSH hosts

<https://argocd.dev.rdev.tech/>



### 7.2设置Repositories



### 7.3创建应用

