Ubuntu16.04上用Jenkin持续部署和交付

—— 陈亮 2019/1

**用Gitlab-runner+gitlab+idea+ubuntu16.04在开发和测试环境完成测试实现持续集成后,就可以上生成环境用Jenkin持续部署和交付了**

## 一:基于 Docker 安装 Jenkins

创建docker-compose.yml文件:

vi /usr/local/docker/jenkins/docker-compose.yml

version: '3'

services:

docker\_jenkins:

restart: always

# jenkins用长期有效版lts

image: jenkins/jenkins:lts

# image: environment

# 容器名

container\_name: docker\_jenkins

ports:

- '8080:8080'

- '50000:50000'

# 挂在数据卷

volumes:

# 映射jenkins\_home，将数据存储到宿主机，这样配置就不会随容器丢失

- ./data/:/var/jenkins\_home

# 把宿主机docker 映射到容器内，可以直接在容器内使用宿主机docker

- /var/run/docker.sock:/var/run/docker.sock

# 挂载docker相关的文件

- /usr/bin/docker:/usr/bin/docker

- /usr/lib/x86\_64-linux-gnu/libltdl.so.7:/usr/lib/x86\_64-linux-gnu/libltdl.so.7

# 自主安装的一些运行时软件(如java/maven/git...),次案例安装到/usr/local/java目录

- /opt:/opt

# 映射时区文件，保证容器的时区和宿主机相同（官网的jenkins容器时区不是中国的时区）

- /etc/timezone:/etc/timezone

# 插件挂在:java目录下安装有jdk、maven、tomcat等

- /usr/local/java:/usr/local/java

# 挂在环境变量的配置文件profile,如果没生效，即jenkins不能构建时,手动

# 进入容器 docker exec -it 容器id bash,执行source etc/profile让配置生效

- /etc/profile:/etc/profile

# 指定时间变量

environment:

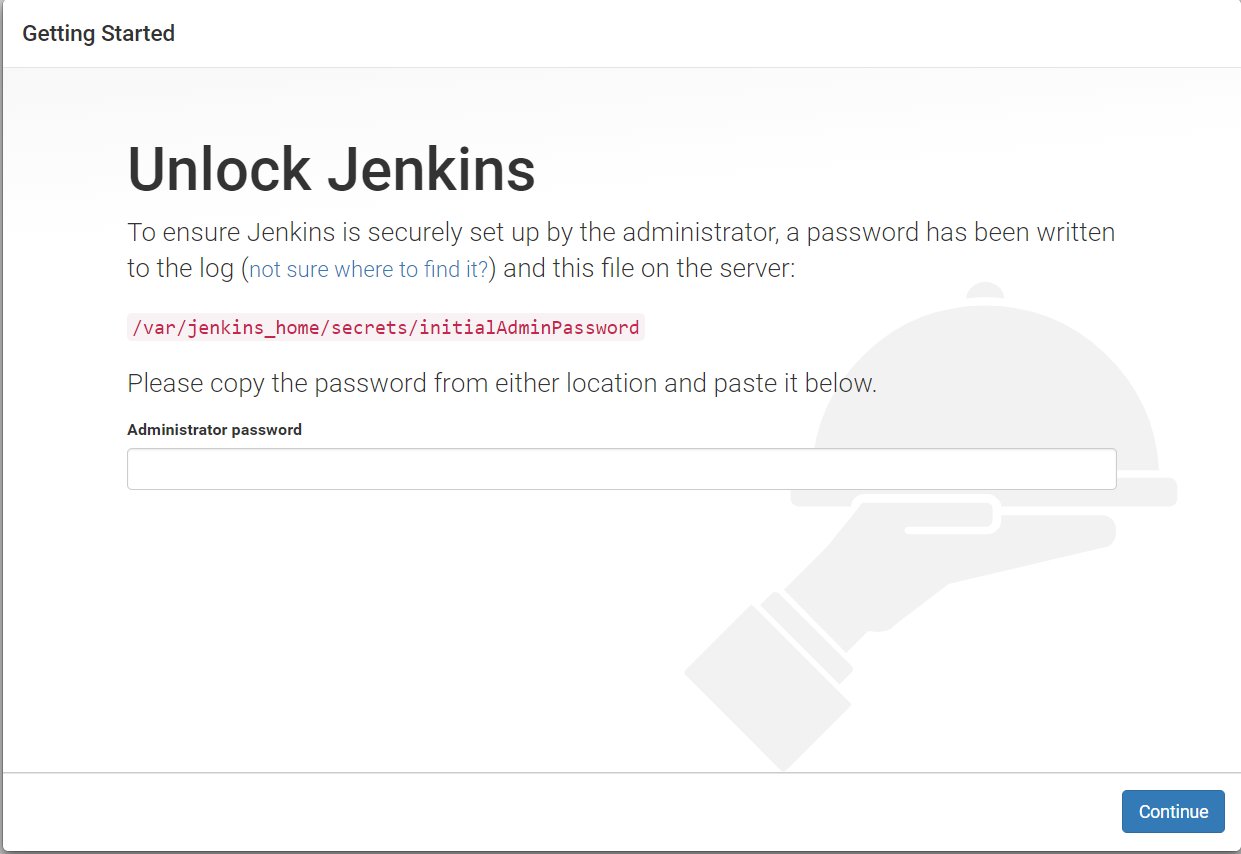
TZ: Asia/Shanghai

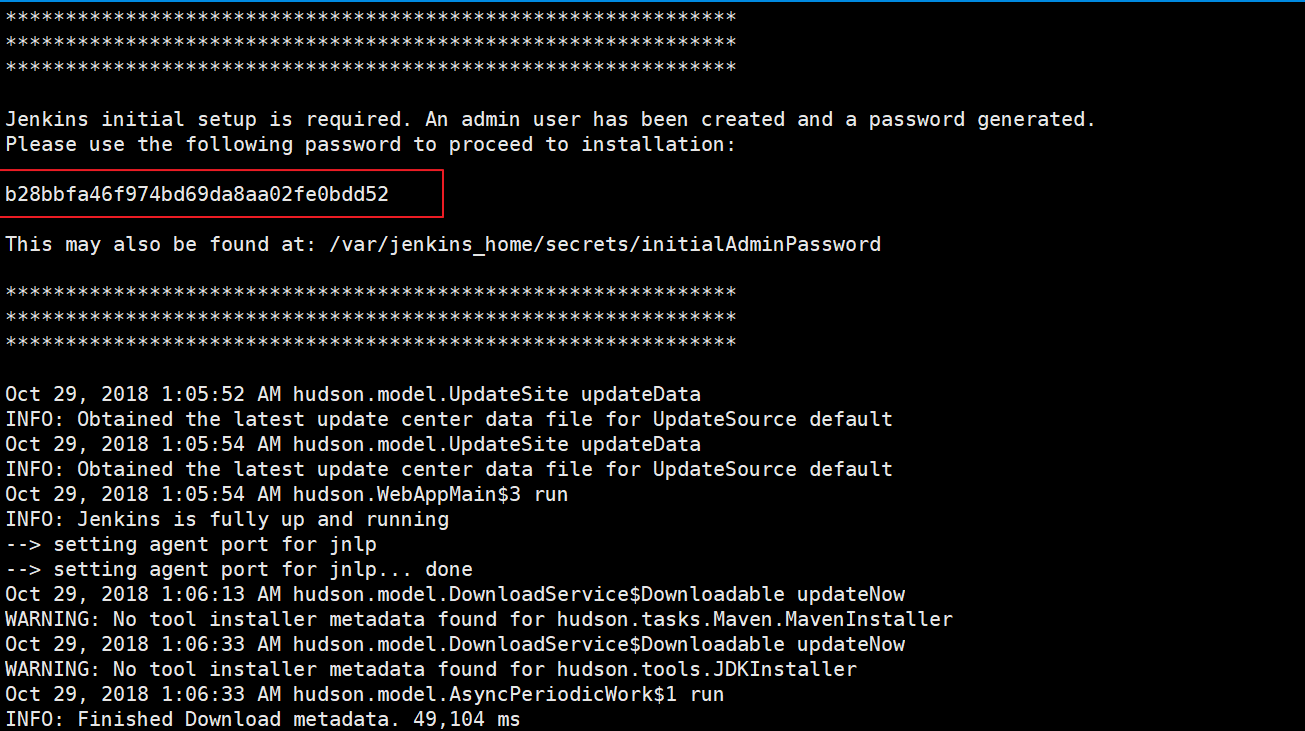
## 安装过程中会出现 Docker 数据卷 权限问题，用以下命令解决：

chown -R 1000 /usr/local/docker/jenkins/data

## **解锁 Jenkins**

访问 主机: ip:8080,比如192.168.1.8::8080;Jenkins 第一次启动时需要输入一个初始密码用以解锁安装流程，使用 docker logs jenkins 即可方便的查看到初始密码





****离线版处理:站点升级****

# **[jenkins升级](https://www.cnblogs.com/linuxws/p/10556940.html):若报错重启**

如果安装的jenkins版本过低会导致插件无法安装的问题，需要升级：插件管理==>上传插件

　　　　==>升级站点：获取插件的站点

  站点信息从：https://updates.jenkins.io/update-center.json 改为如下地址【四选一即可】,次处使用

https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/current/update-center.json

http://mirror.xmission.com/jenkins/updates/update-center.json # 推荐

<http://mirrors.shu.edu.cn/jenkins/updates/current/update-center.json>

https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/update-center.json  
https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/current/update-center.json

****注意：**** 安装时可能会因为网速等原因导致安装时间比较长，请大家耐心等待。如果长时间停留在安装页没反应，请尝试使用 F5 刷新一下。

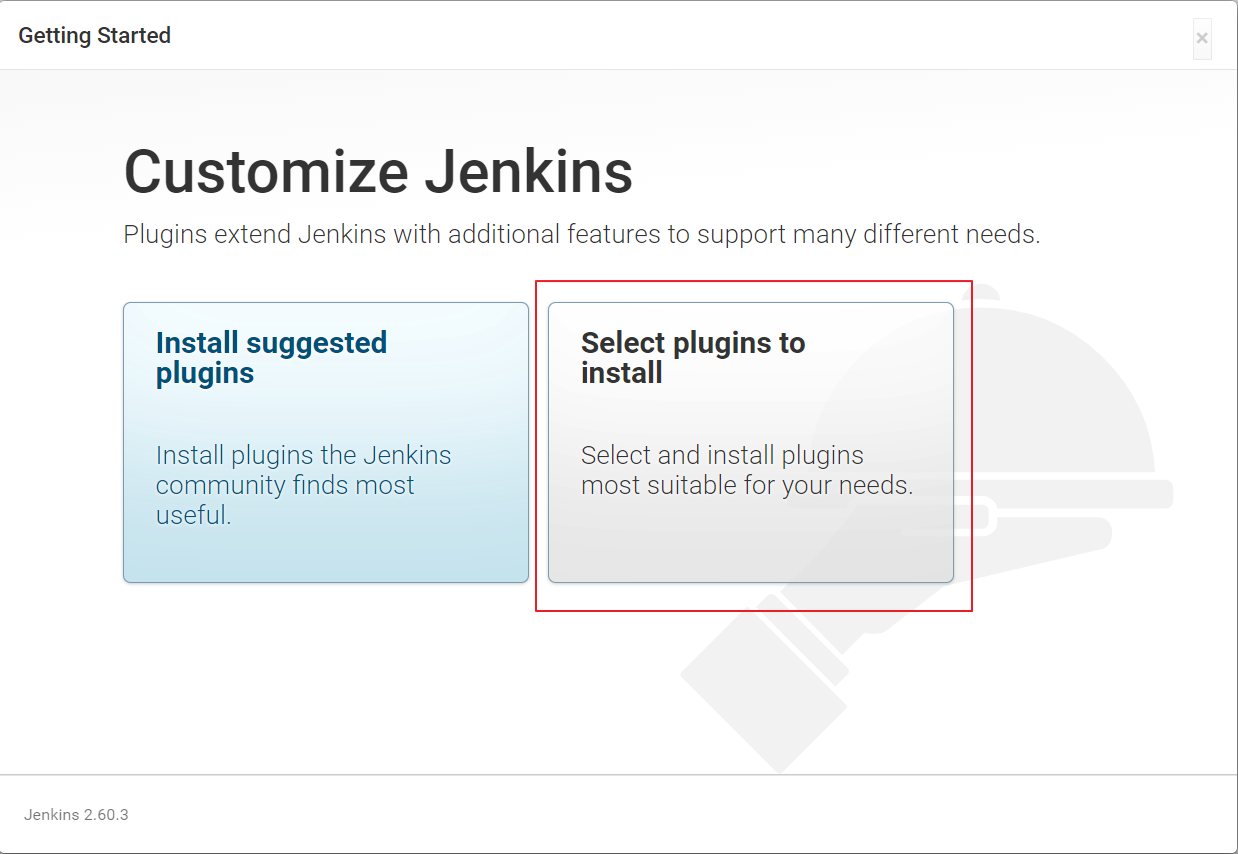
主要插件安装:locale,mailer,extended-choice-parameter,git,git-client ,pipeline-model-extensions,publish-over-ssh,subversion ,ssh-slaves,collabnet-automic-deplo或deploy to container,ws-cleanup,zanataw

## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-Jenkins.html" \l "使用自定义插件的方式安装)**使用自定义插件的方式安装**

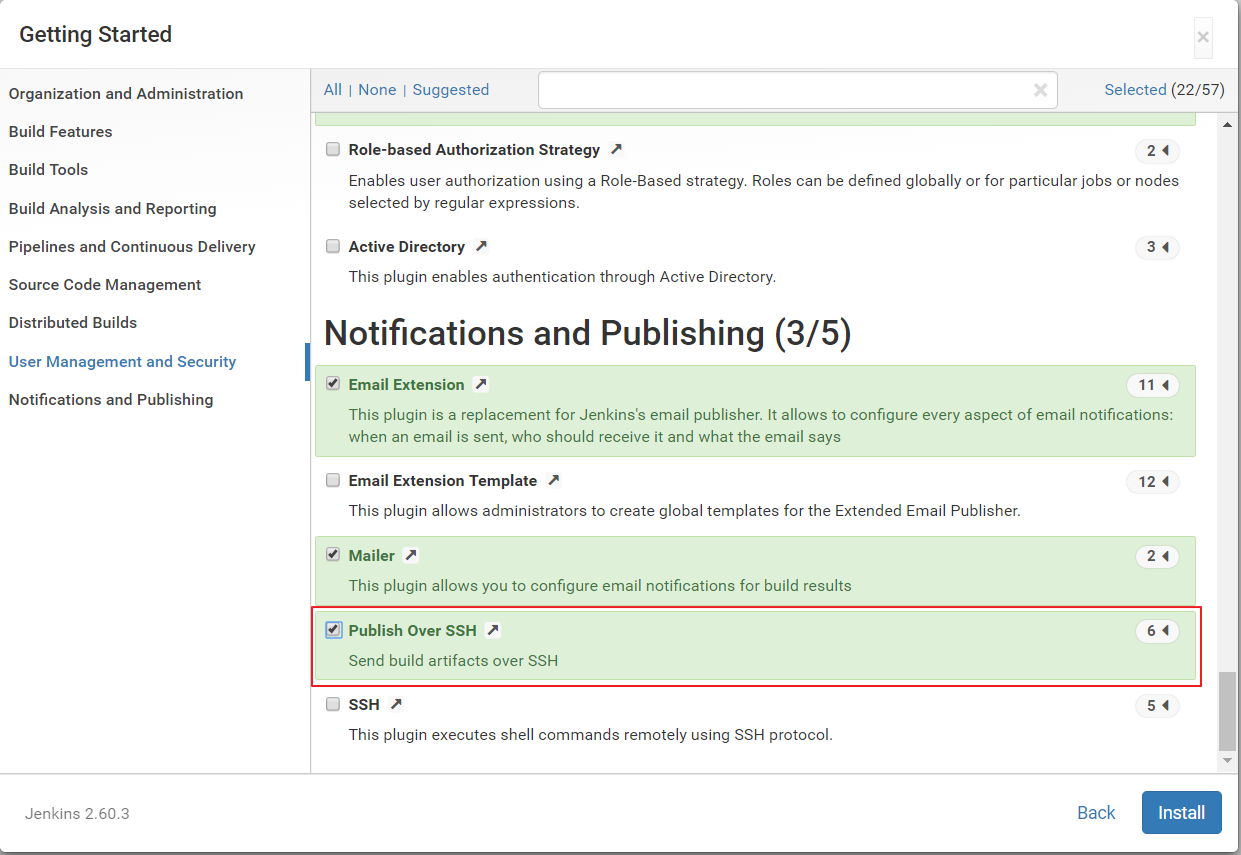
安装一次不能成功,直接选择不用安装,后续手动安装

插件是 Jenkins 的核心，其丰富的插件（截止到 2018.10.29 共有 77350 个插件）可以满足不同人群的不同需求

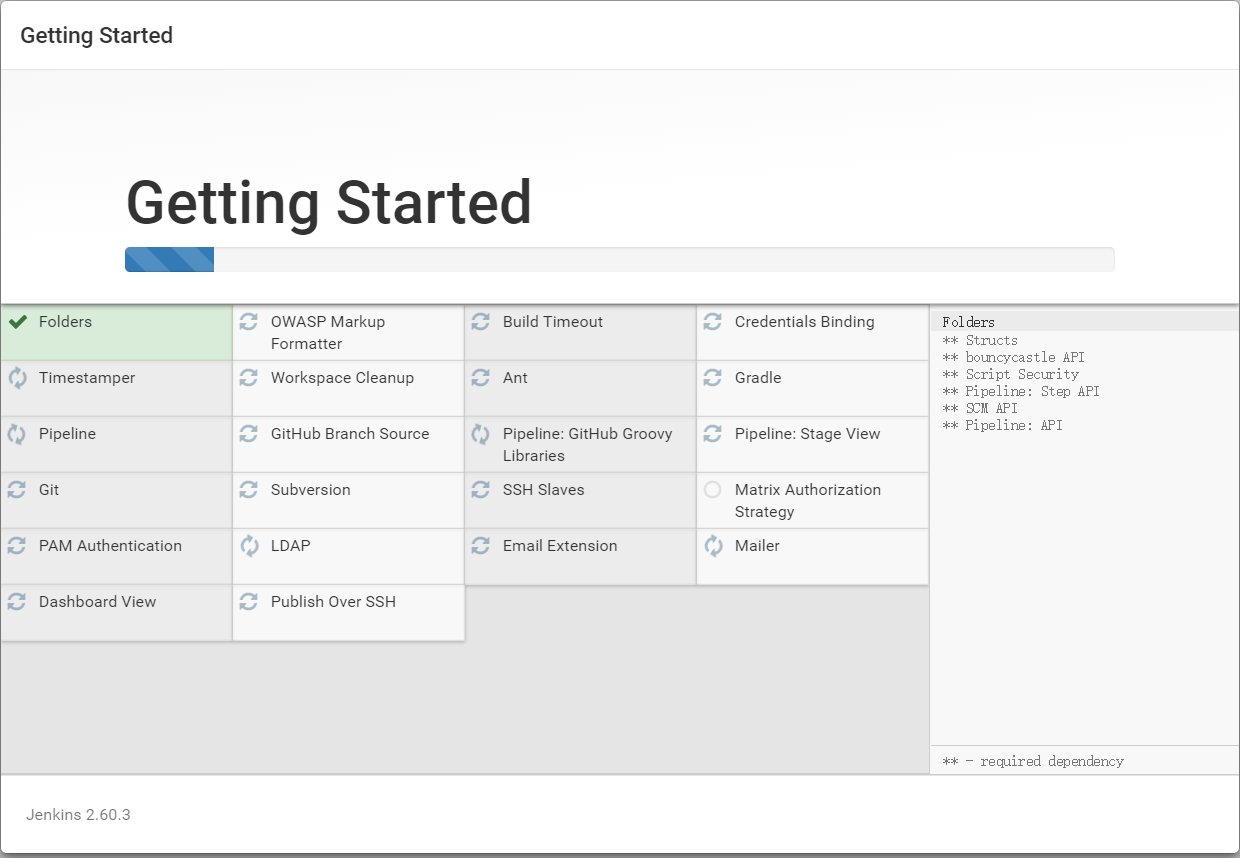
插件地址：https://plugins.jenkins.io/



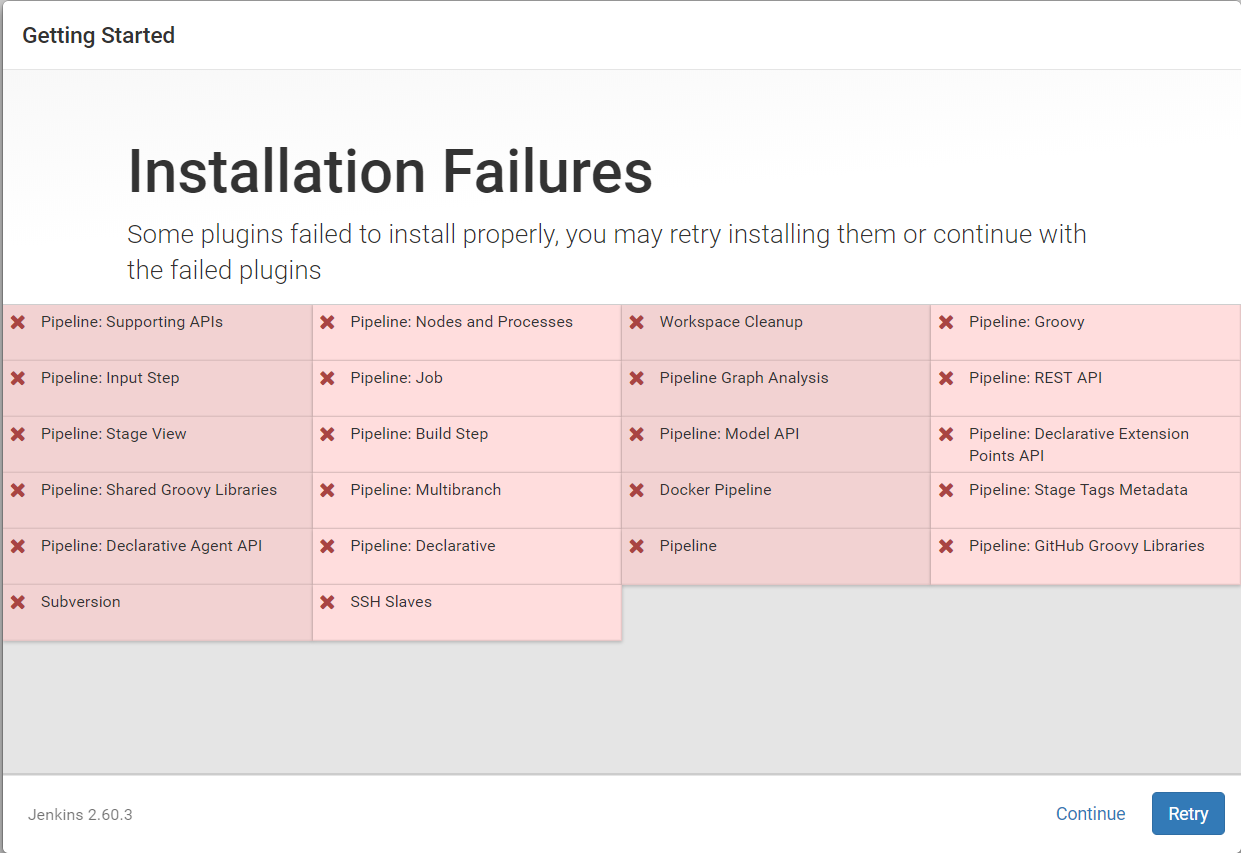
****注意：**** 除了默认勾选的插件外，一定要勾选 Publish over SSH 插件，这是我们实现持续交付的重点插件。



****开始安装了，根据网络情况，安装时间可能会比较长，请耐心等待****

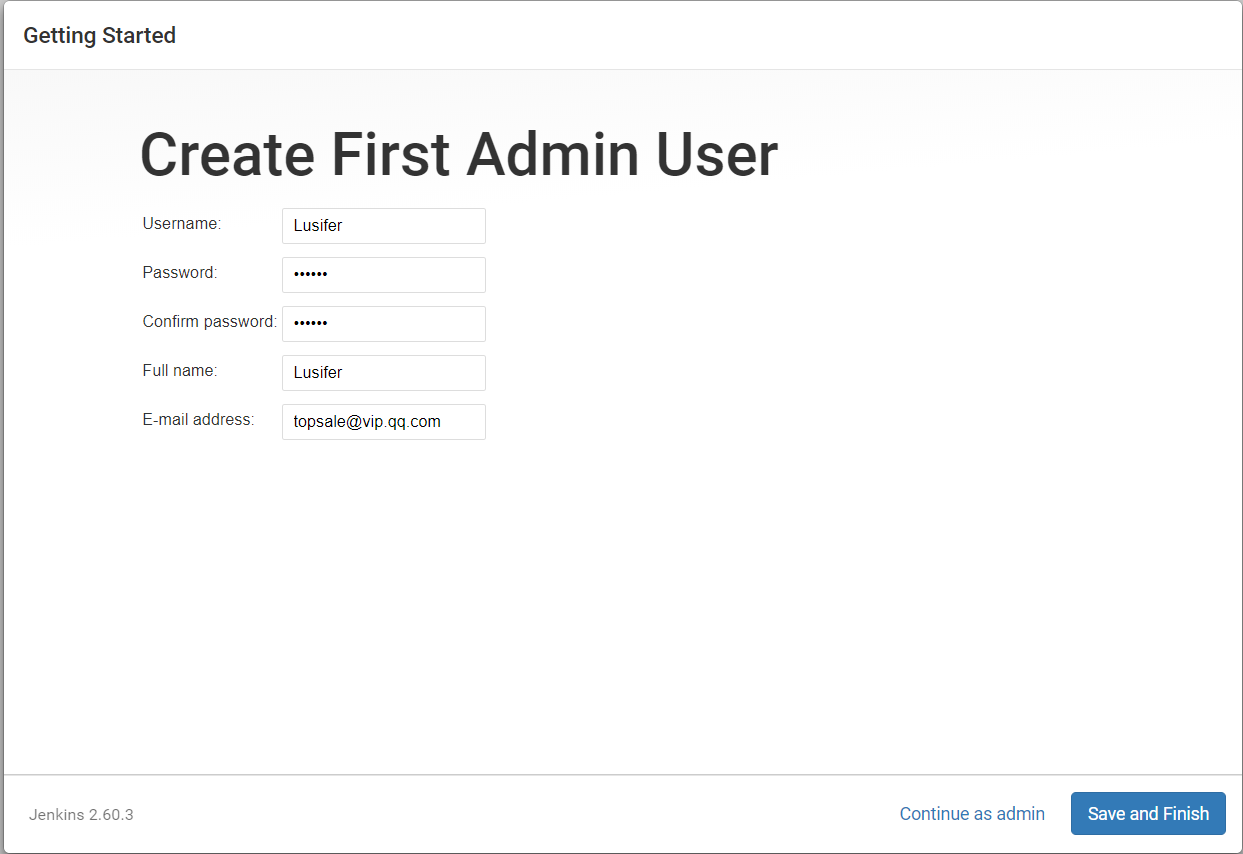


****很多插件装不上怎么办？不要慌，记住这些插件的名字，咱们稍后可以手动安装****

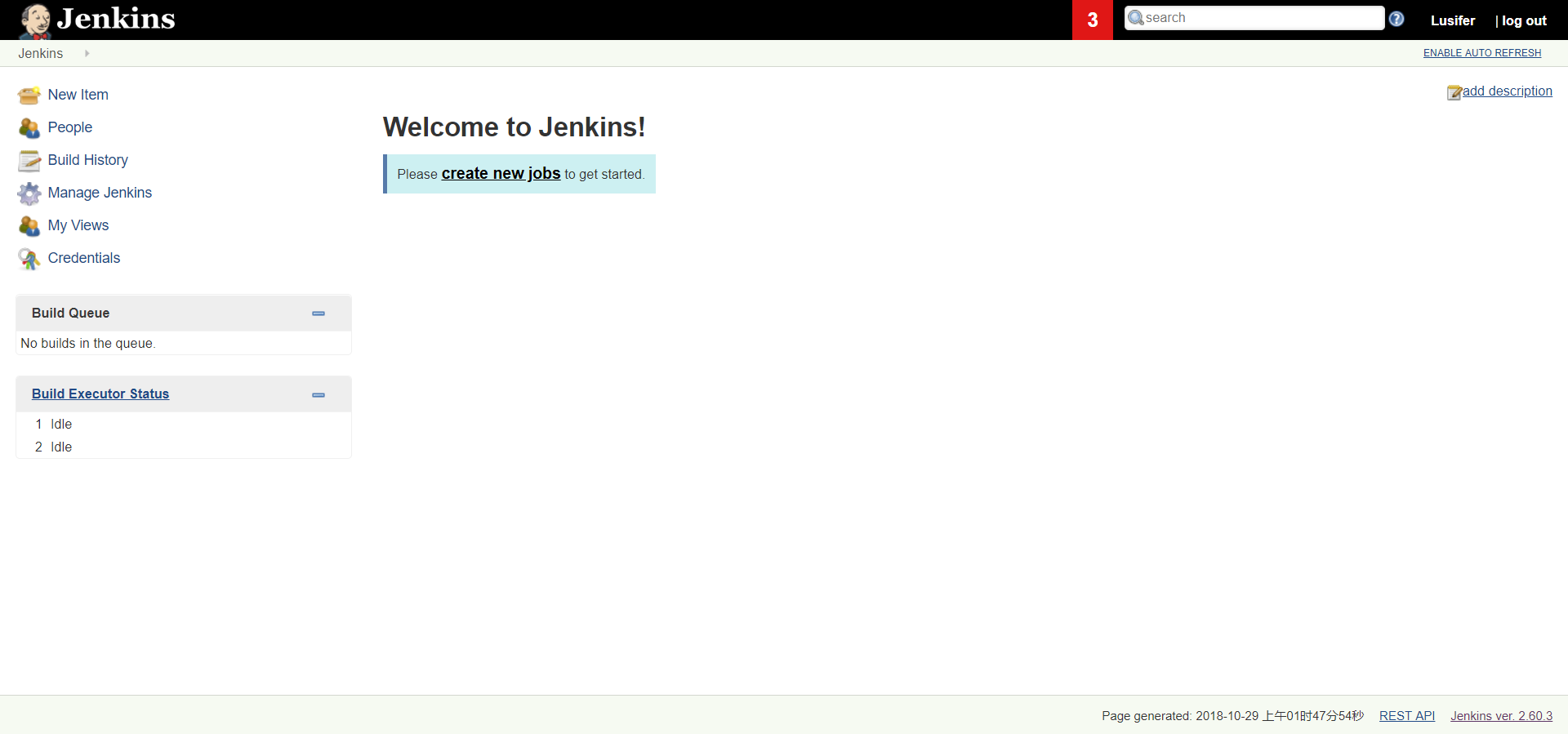


## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-Jenkins.html" \l "安装成功效果图)**安装成功效果图**

* 创建管理员



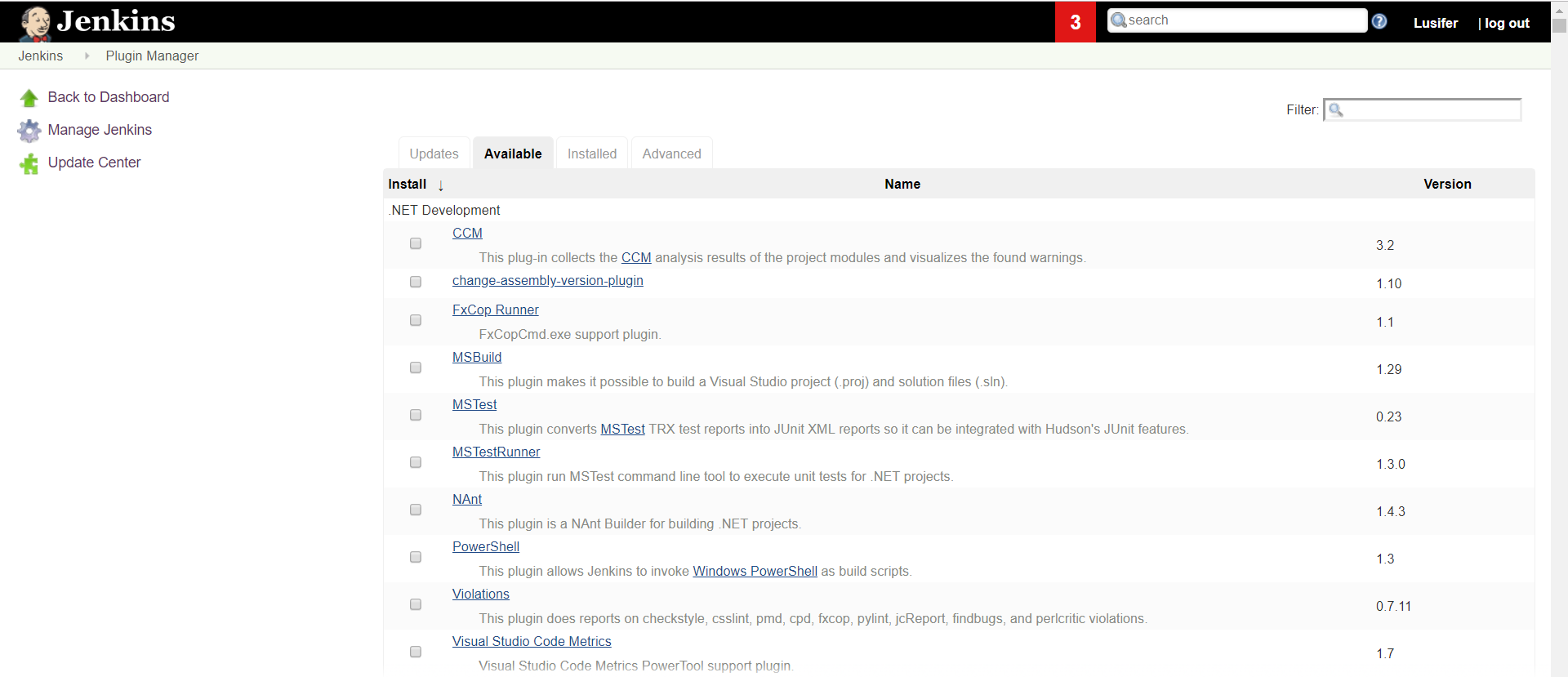
* 安装完成，进入首页



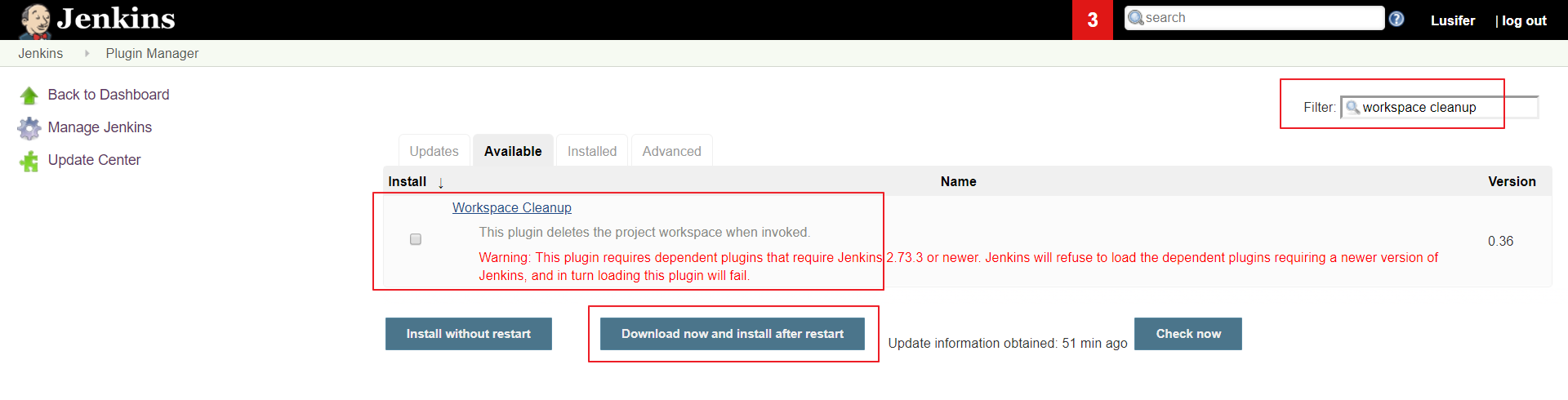
## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-Jenkins.html" \l "附：jenkins-手动安装插件)**附：Jenkins 手动安装插件**

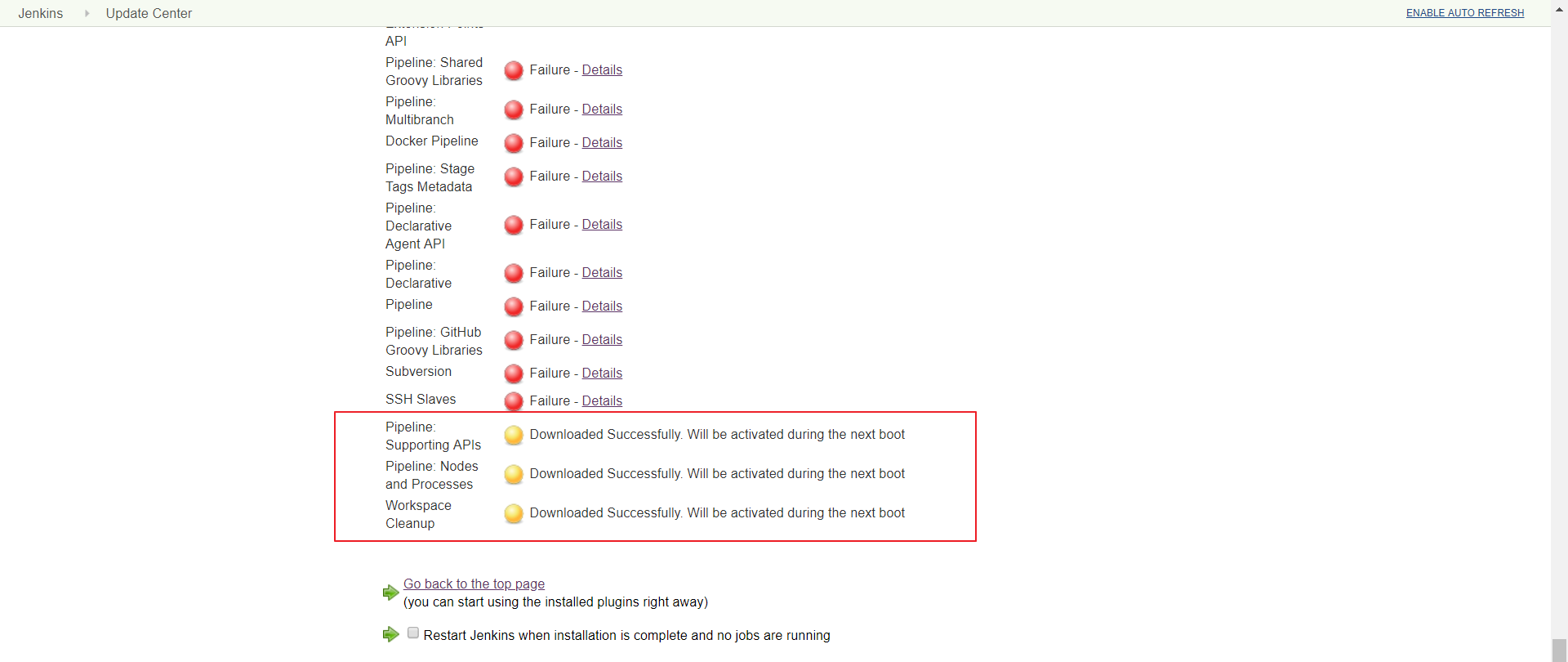
### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-Jenkins.html" \l "使用插件管理器安装)**使用插件管理器安装**

* Manage Jenkins -> Manage Plugins -> Avaliable



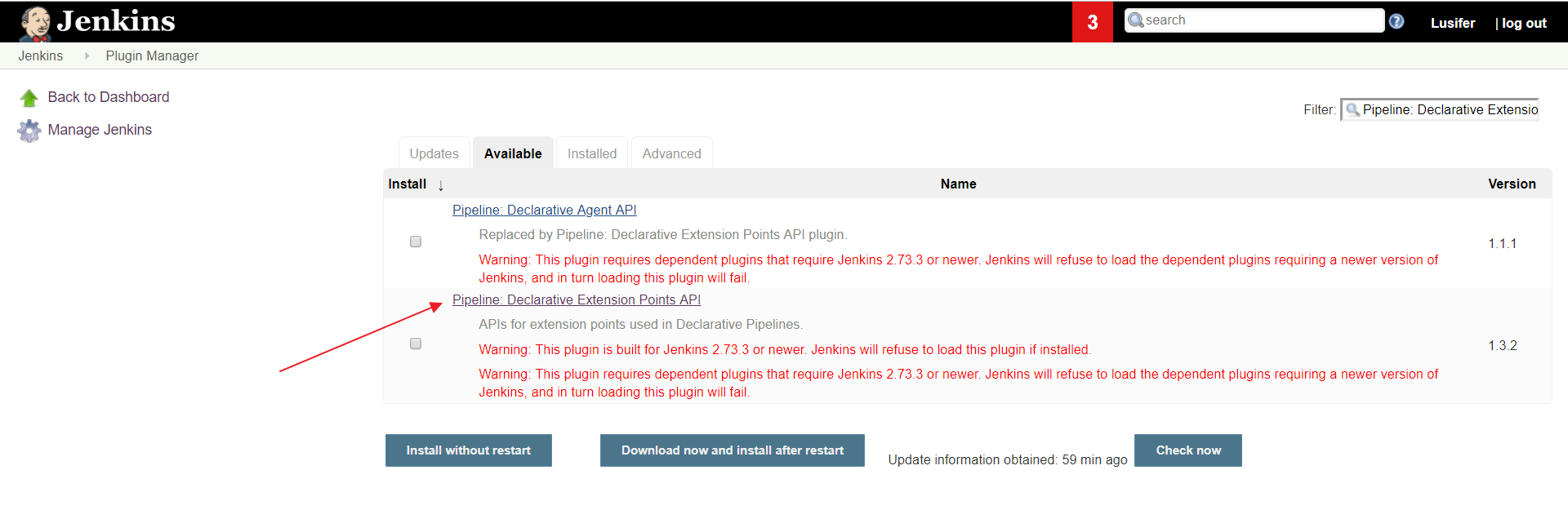
* 过滤出想要安装的插件，然后点击 Download now and install after restart



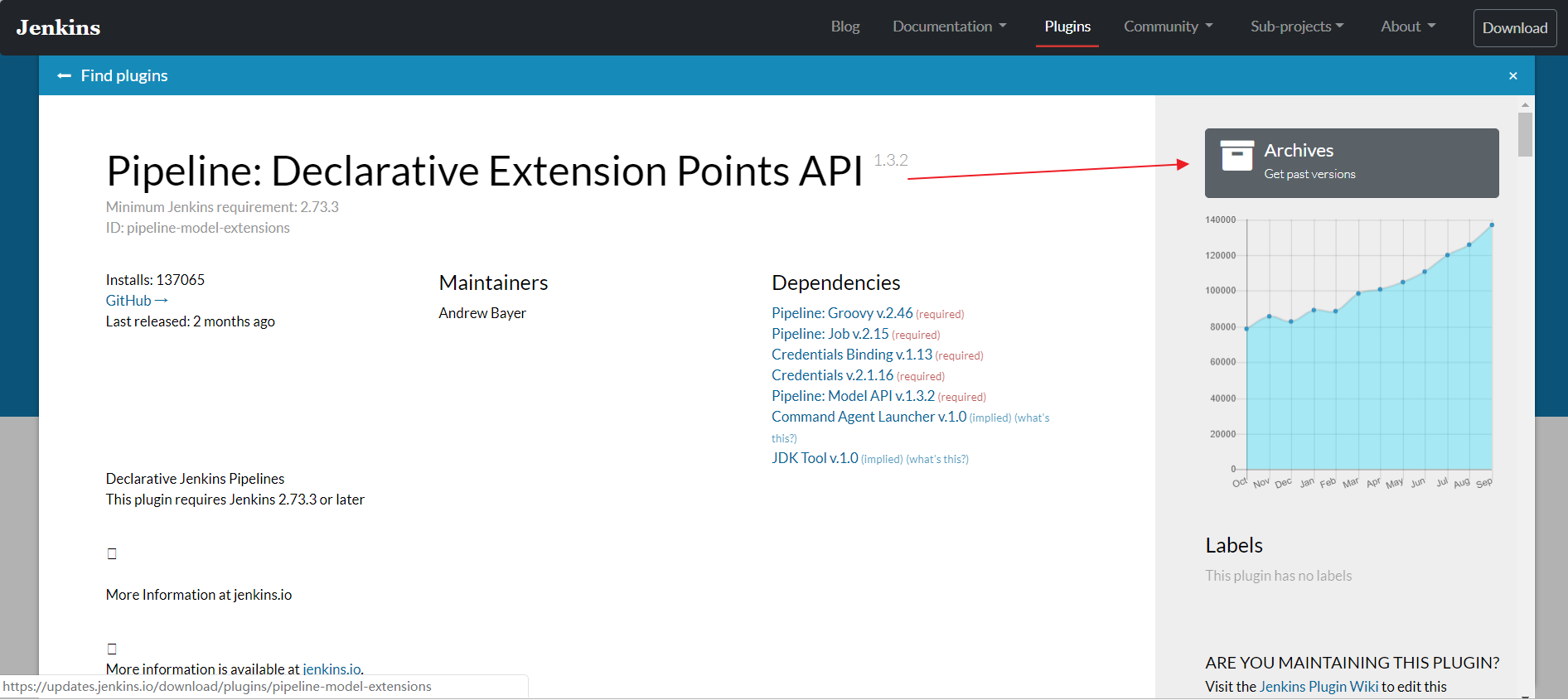


### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-Jenkins.html" \l "手动上传-hpi-文件)**手动上传 .hpi 文件**

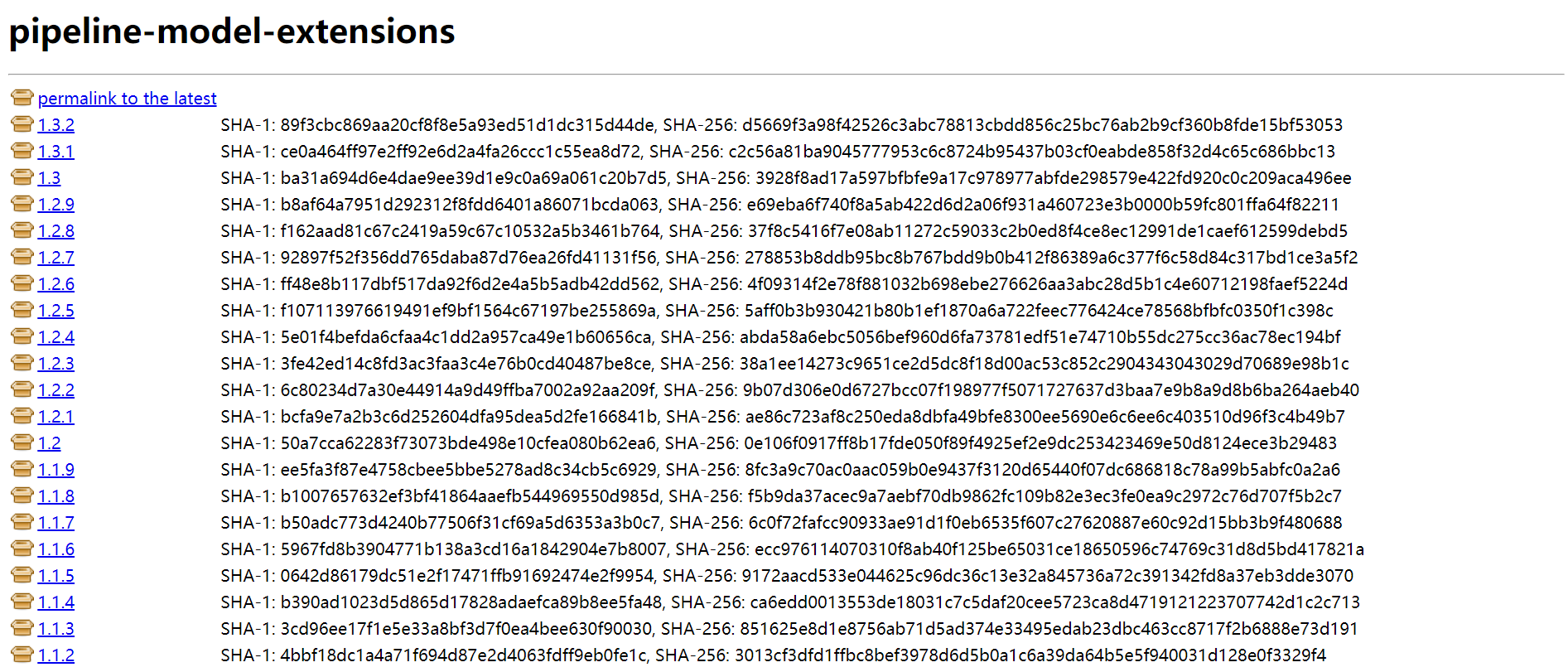
* 点击进入插件中心



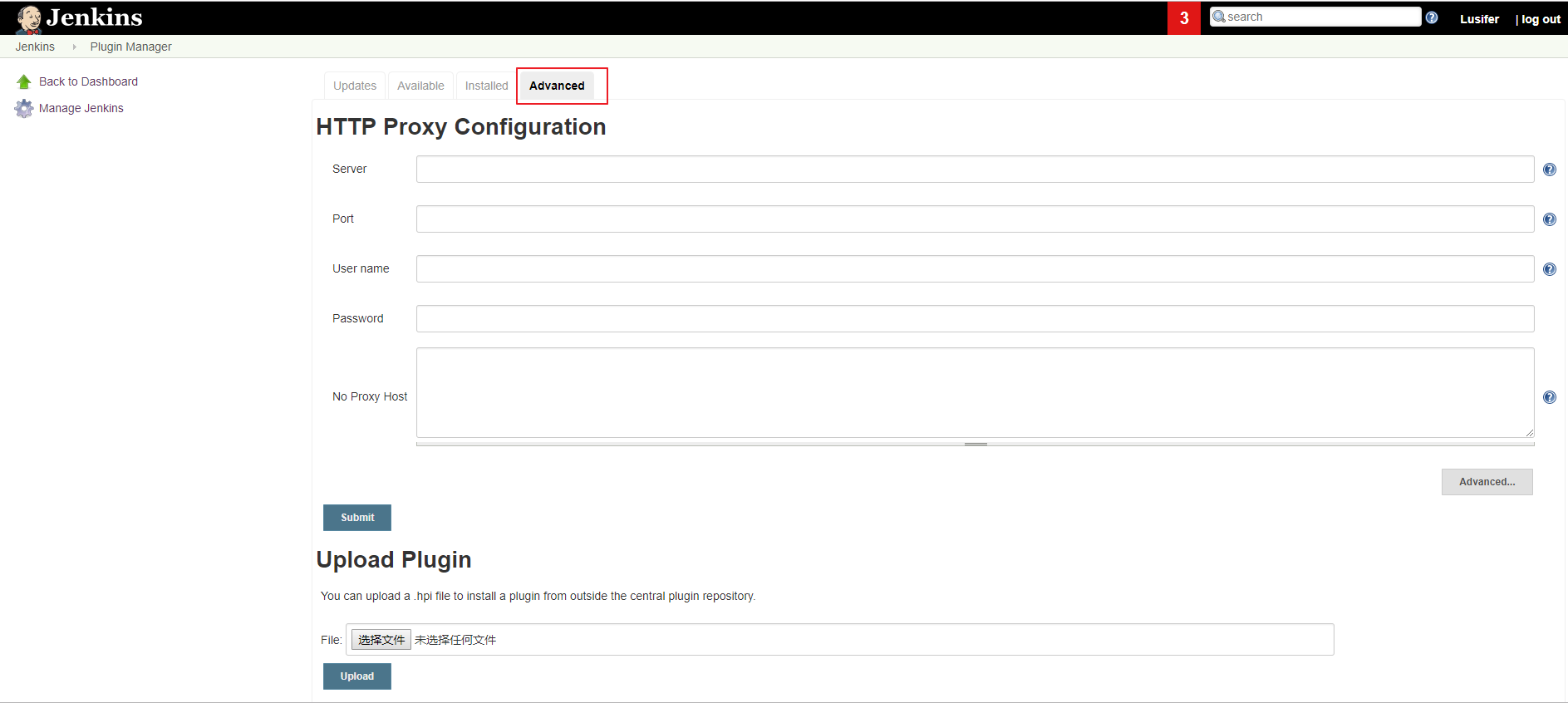
* 点击 Archives



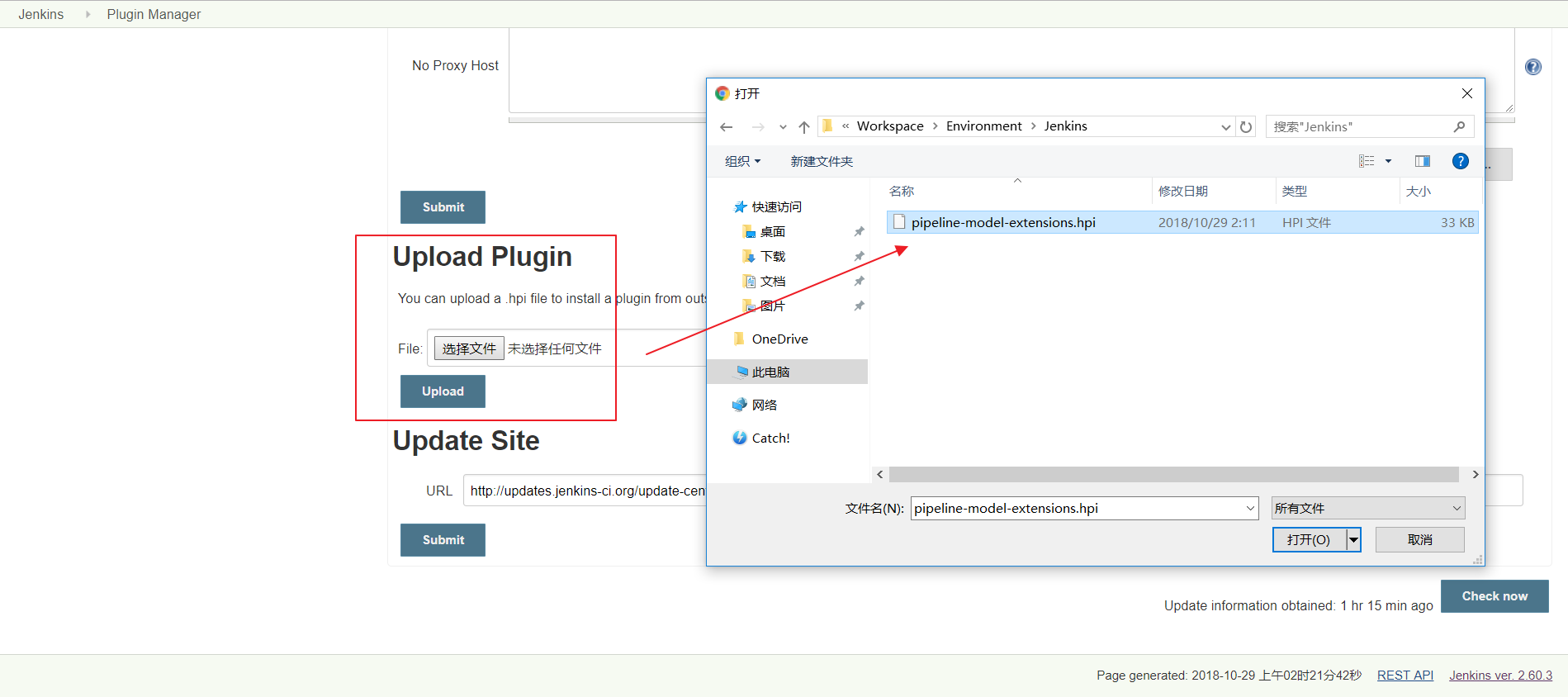
* 下载需要的版本



* 在插件管理器中选择 Advanced



* 选择上传即可



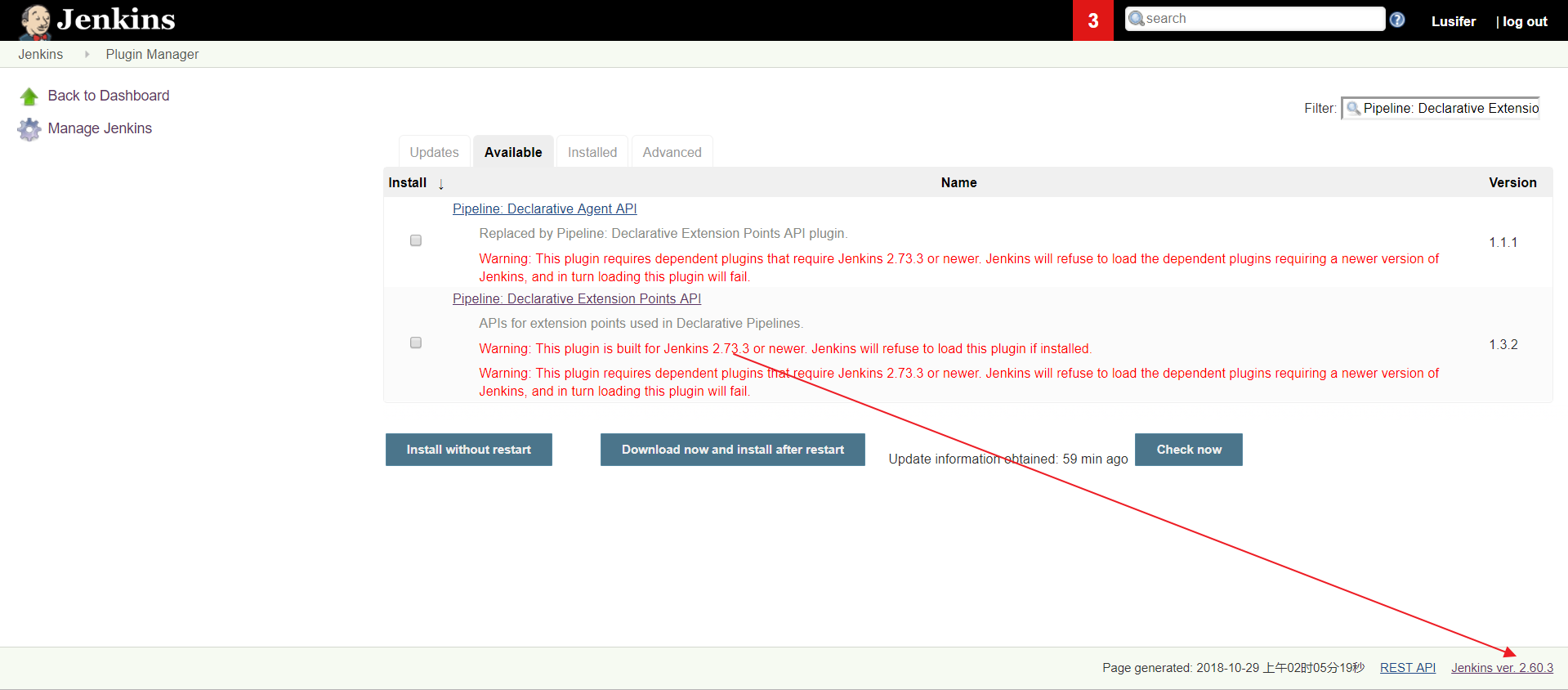
### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-Jenkins.html" \l "重启-jenkins)**重启 Jenkins**

docker-compose down

docker-compose up -d

1  
2

****注意：**** 请留意需要下载插件的警告信息，如果不满足安装条件，Jenkins 是会拒绝安装的。如下图：



## 二:配置jenkins

## **配置 JDK & Maven**

保证docker-compose.yml中挂载插件和环境变量目录数据卷时，宿主机和jenkins容器目录相同(此案例插件目录为

-/usr/local/java:/usr/local/java;环境变量为- /etc/profile:/etc/profile),这样jdk,maven或其他插件都只配置一个。

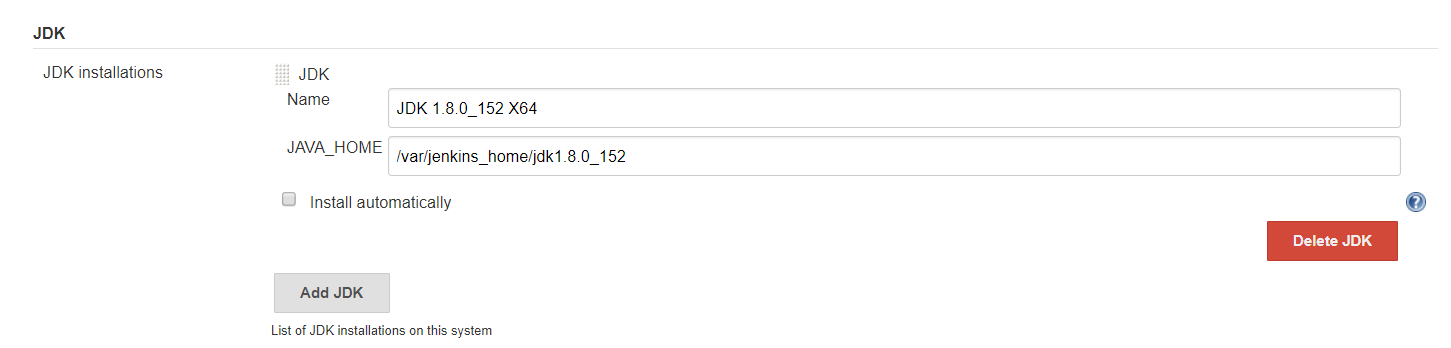
上传 JDK 和 Maven 的 tar 包到服务器（容器数据卷目录）

Manage Jenkins -> Global Tool Configuration

安装 JDK（JAVA\_HOME 的路径是宿主机目录，切记！不明白的看视频！）

/var/jenkins\_home/jdk1.8.0\_152

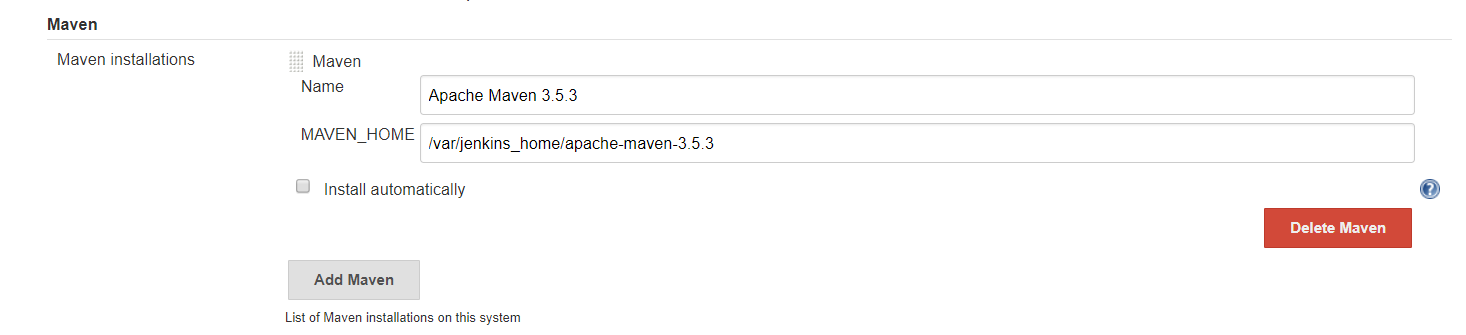
1



* 安装 Maven（MAVEN\_HOME ****的路径是宿主机目录，切记！****）

/var/jenkins\_home/apache-maven-3.5.3

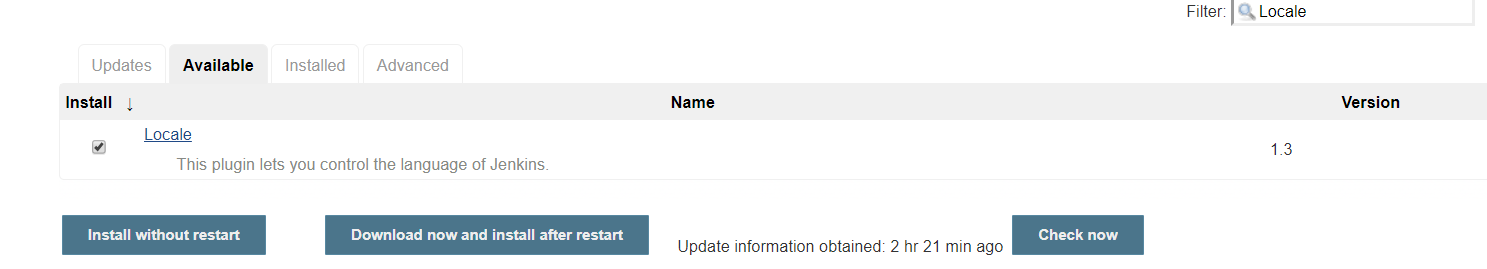
1



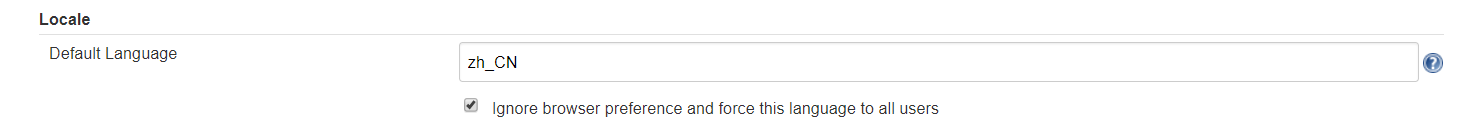
* 别忘记保存

## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E9%85%8D%E7%BD%AE-Jenkins.html" \l "配置本地化（显示中文）)**配置本地化（显示中文）**

* 安装 Locale 插件



* Manage Jenkins -> Configure System -> Locale

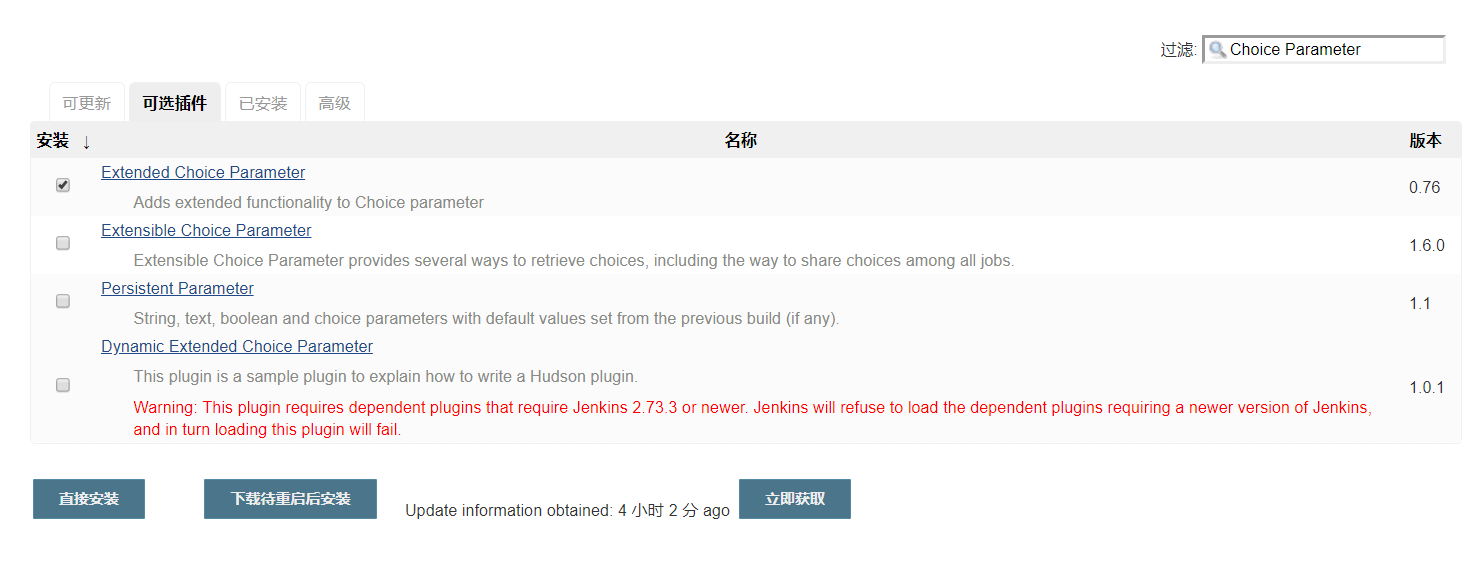


* 本地化效果图



## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E9%85%8D%E7%BD%AE-Jenkins.html" \l "安装动态参数插件)**安装动态参数插件**

该插件的主要目的是为了方便我们后面在做项目构建时可以按照版本进行构建（支持一键回滚哦）



## 三:持续交付实战用户管理服务

## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "写在前面)**前言**

Jenkins 的持续交付流程与 GitLab Runner 的持续集成差不多，但 GitLab Runner 已经默认是配置好了 Git，所以 Jenkins 需要额外配置多一个 GitLab 的 SSH 登录。按照之前 GitLab Runner 的持续集成流程，Jenkins 的持续交付流程大致如下（其实原理还是挺简单的，但对于刚刚接触 Jenkins 同学理解起来可能还是有一点难度的，最好看下本节视频加深理解）：

* 拉取代码
* 打包构建
* 上传镜像
* 运行容器
* 维护清理

## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "配置-jenkins-的-gitlab-ssh-免密登录)**配置 Jenkins 的 GitLab SSH 免密登录**

步骤同 [使用 SSH 的方式拉取和推送项目](https://www.funtl.com/zh/gitlab/GitLab-%E5%88%9B%E5%BB%BA%E7%AC%AC%E4%B8%80%E4%B8%AA%E9%A1%B9%E7%9B%AE.html" \l "使用-ssh-的方式拉取和推送项目)

交互式进入 Jenkins 容器

docker exec -it jenkins容器id /bin/bash

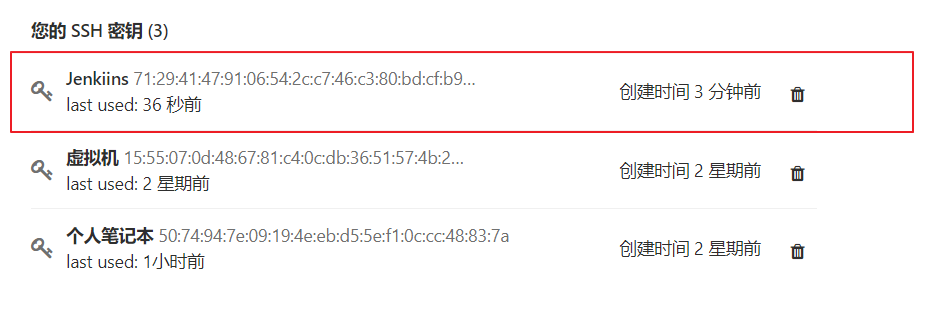
生成 SSH KEY

ssh-keygen -t rsa -C "your\_email@example.com"

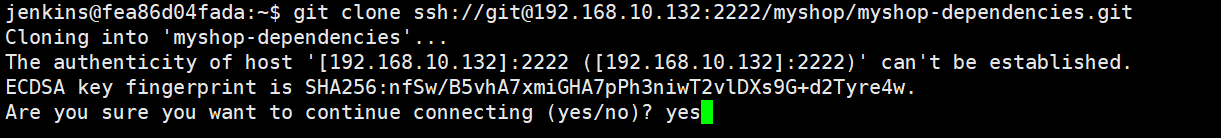
查看公钥

cat /var/jenkins\_home/.ssh/id\_rsa.pub

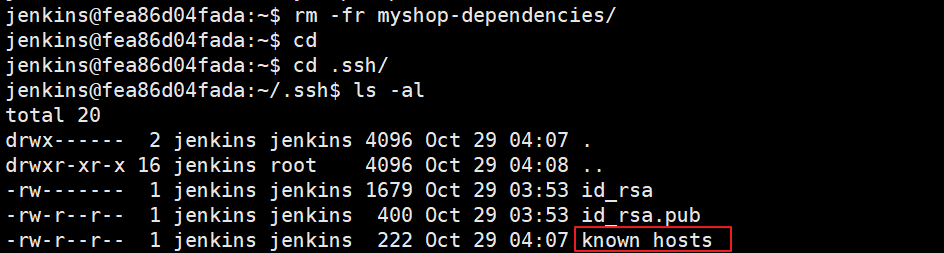
复制公钥到 GitLab



* ****手动克隆一次项目****，该步骤的主要作用是为了生成和服务器的验证信息



* 查看刚才生成的文件

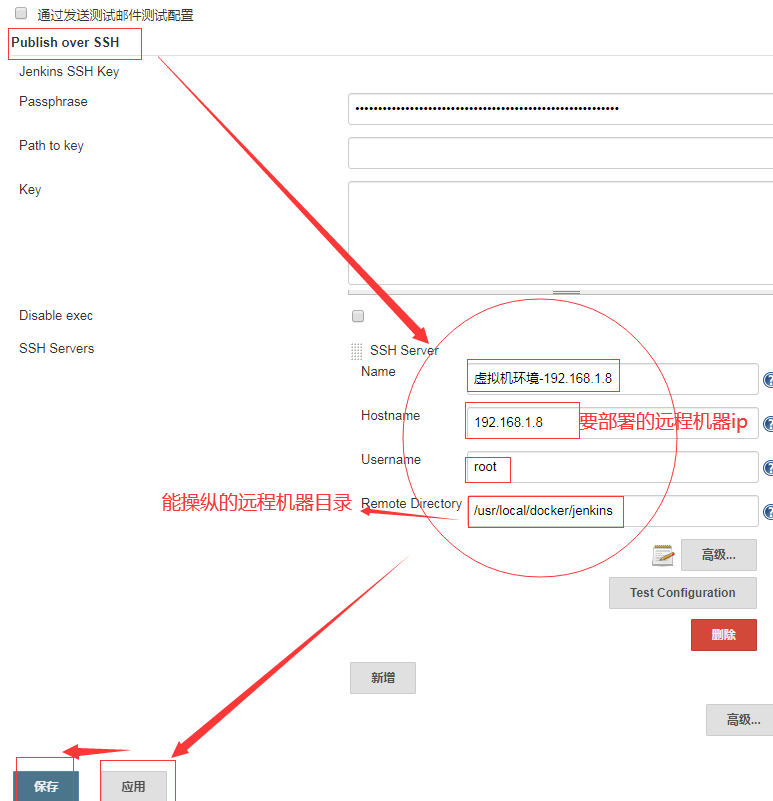


## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "配置-publish-over-ssh)**配置 Publish over SSH**

系统管理 -> 系统设置 -> Publish over SSH

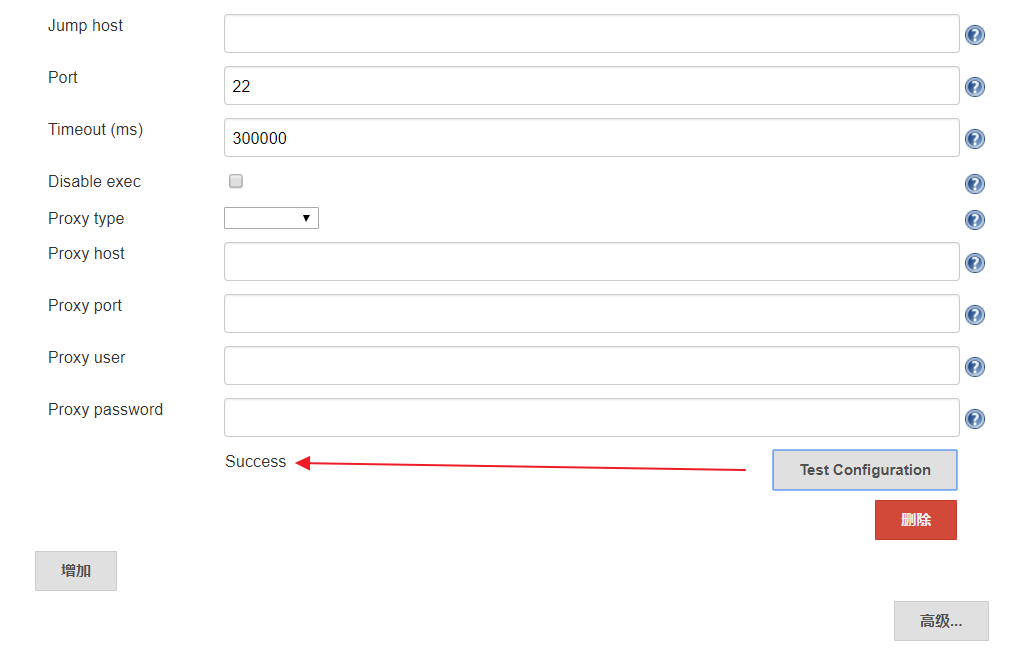
**[Manage Jenkins](http://192.168.1.8:8080/manage" \o "Manage Jenkins)**》**Configure System**

****



其中 Remote Directory 是指 Jenkins 可以在部署的远程目标服务器操作的目录:此案例就部署到jenkins所在机

* 点Test Configuration按钮测试是否能够正常通信

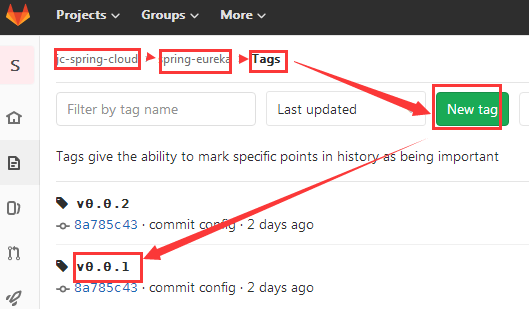


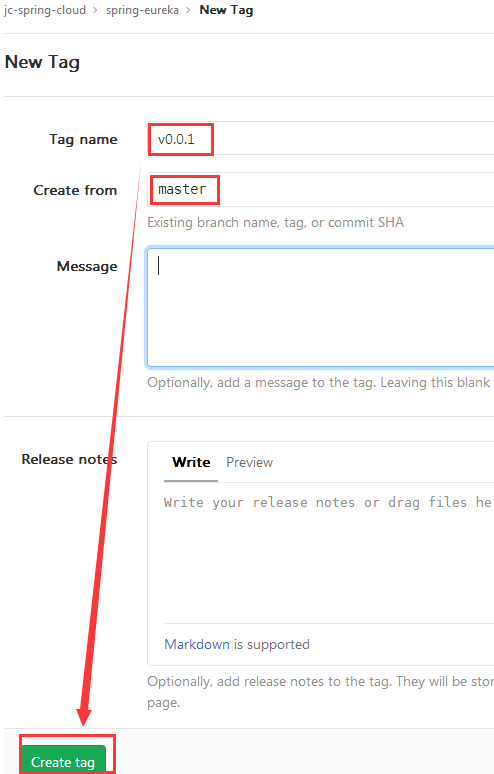
* ****别忘记保存****

## [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "持续交付依赖管理项目)**持续交付spring-eureka项目**

### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "为项目创建标签)**为项目创建标签**

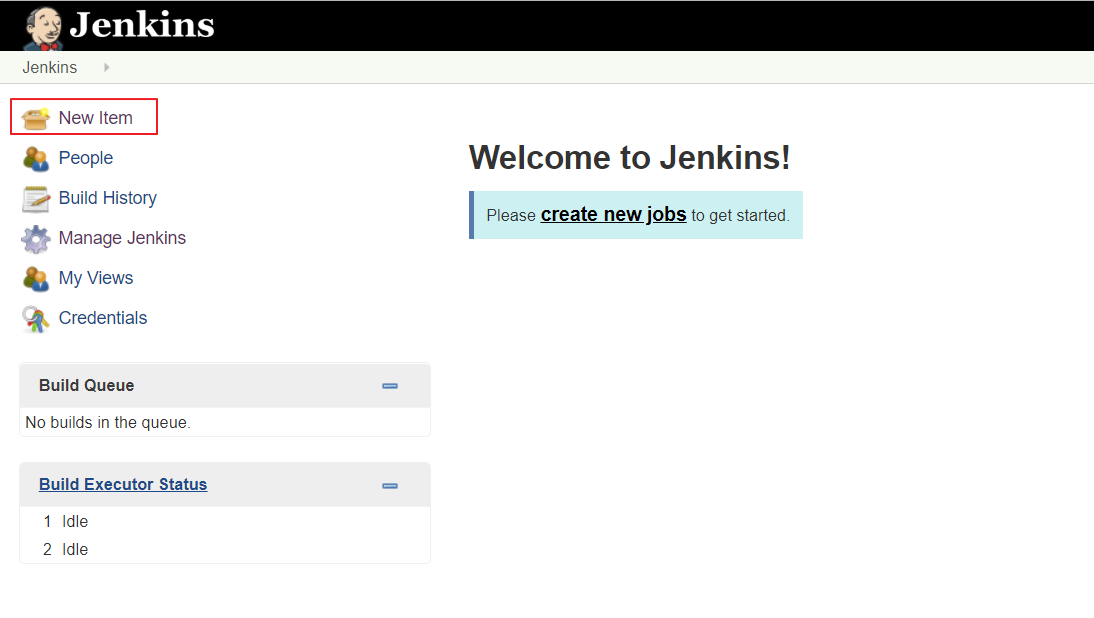
在 GitLab 中为项目创建标签

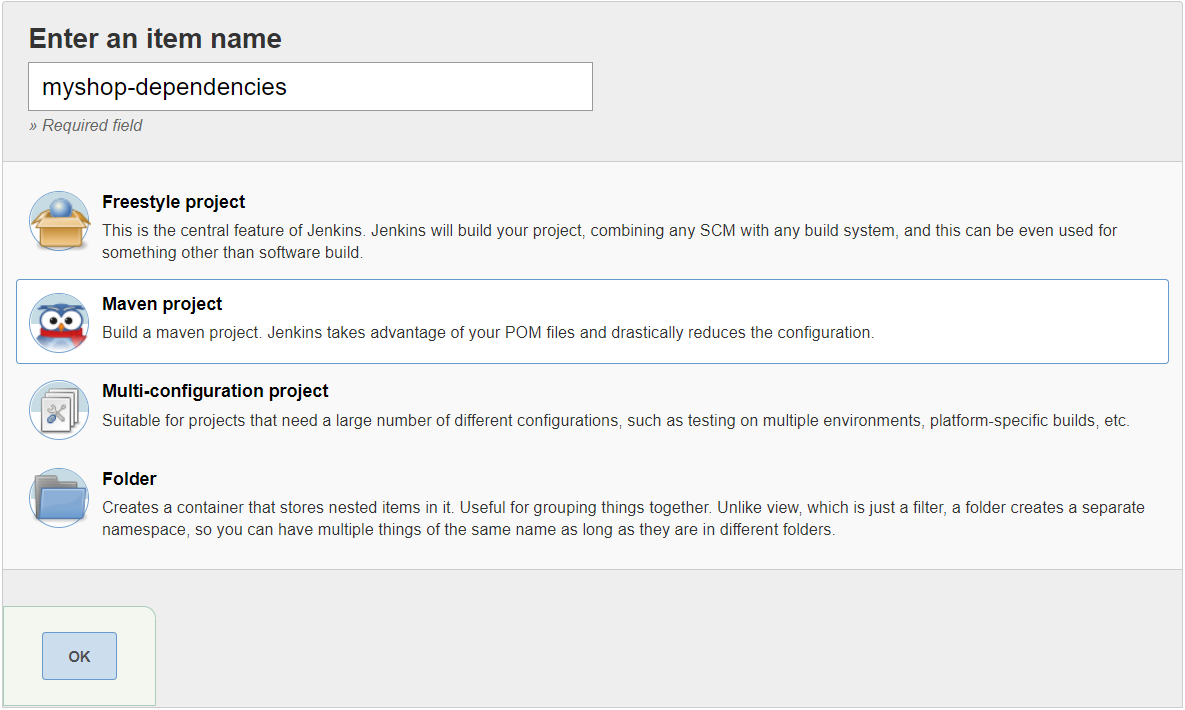




### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "创建-maven-project)**创建 Maven Project或Freestyle Project**

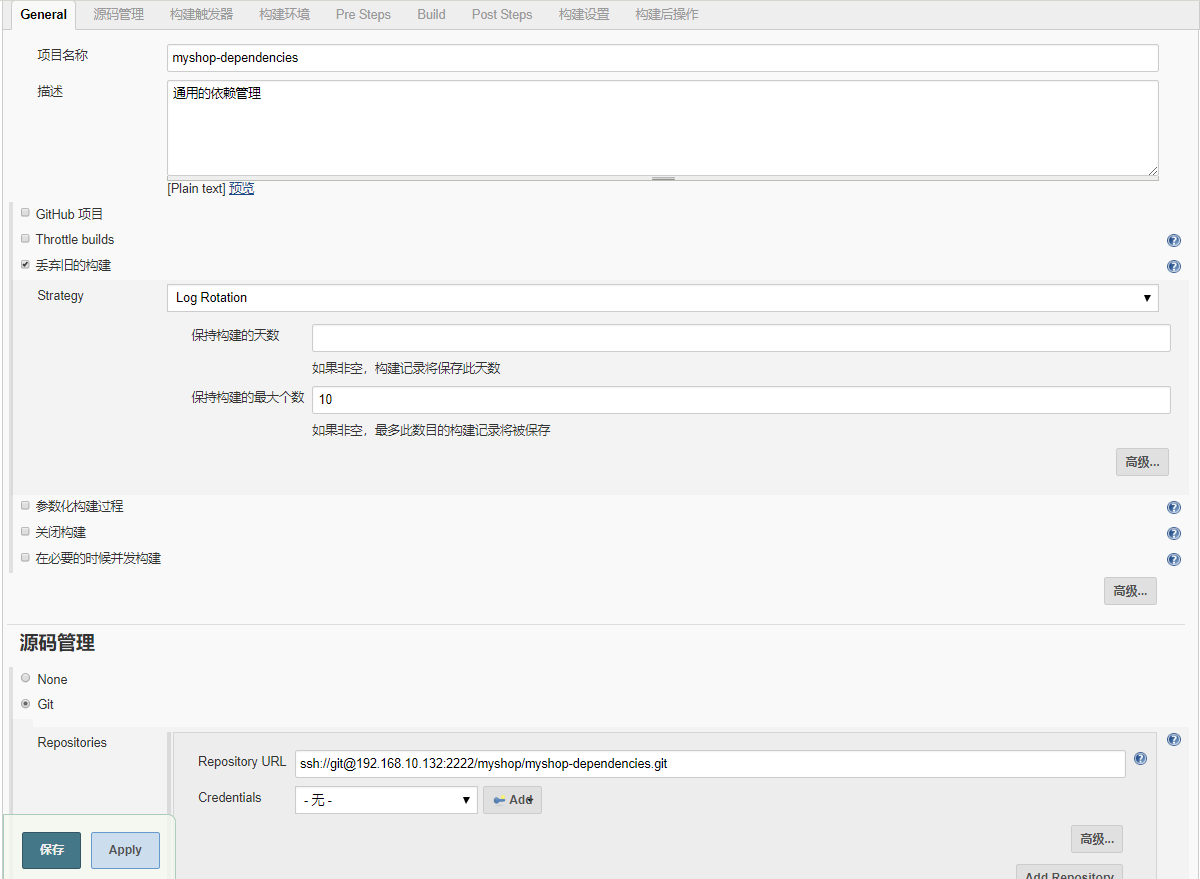
在 Jenkins 中创建一个基于 Maven 的任务





### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "配置-maven-project)**配置 Maven Project**

#### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "配置第一次构建)**配置第一次构建**

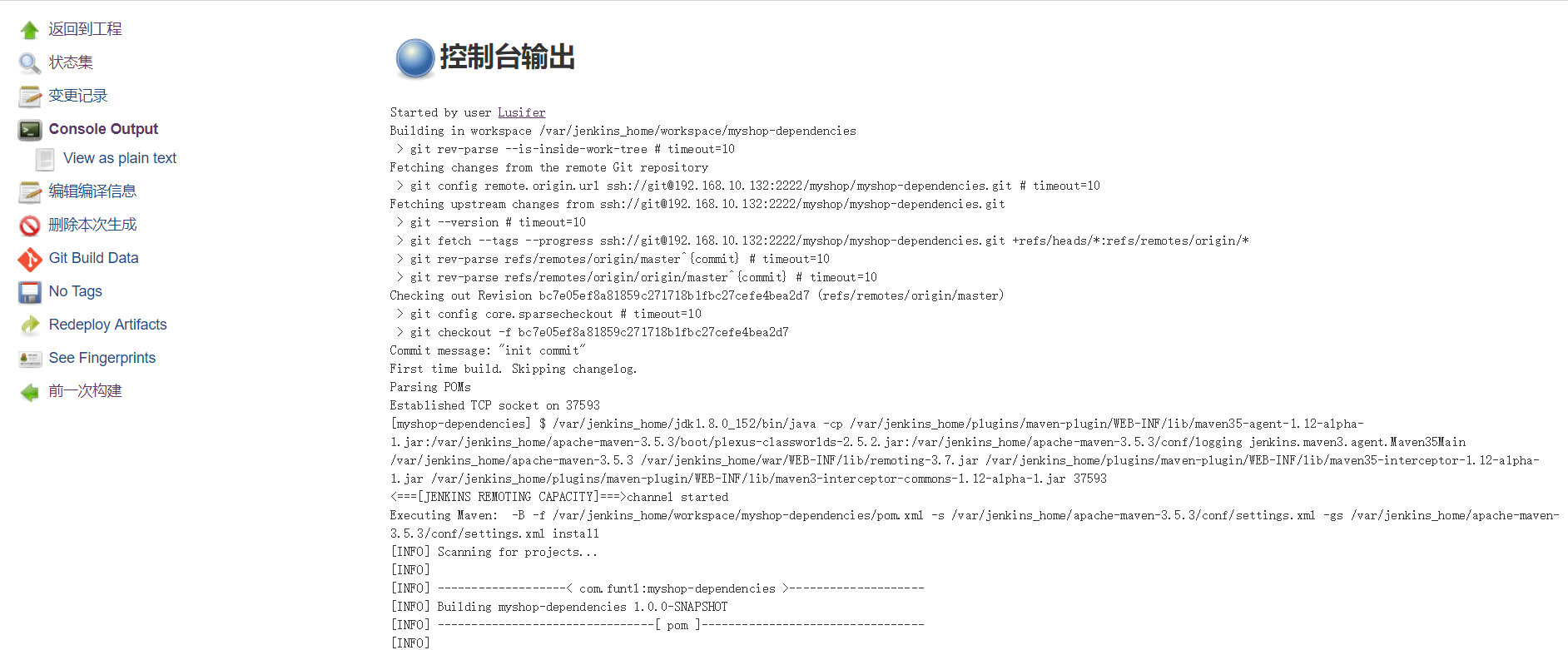


* 构建项目



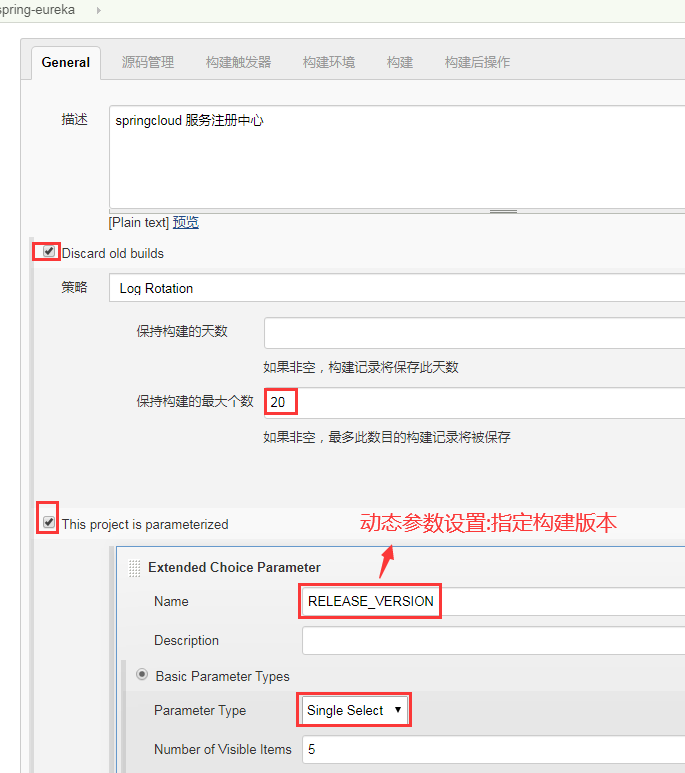
* 查看构建日志

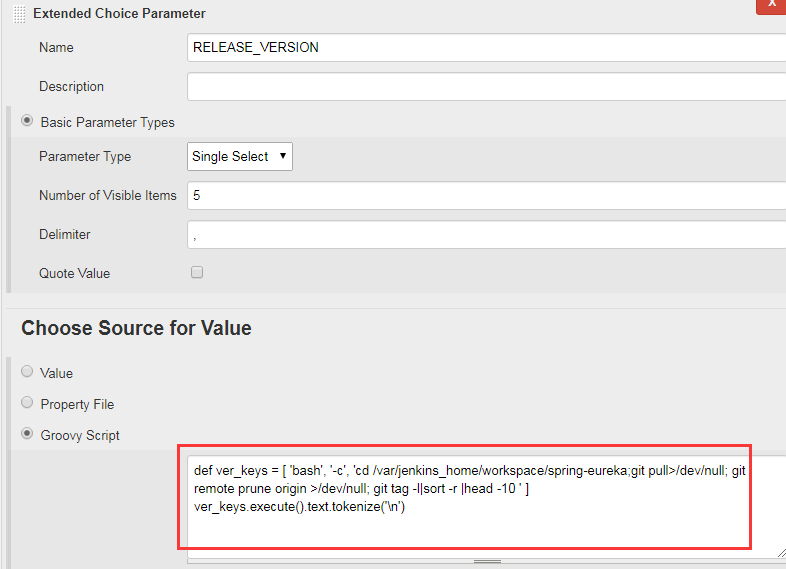




#### [#](https://www.funtl.com/zh/apache-dubbo-ci/%E6%8C%81%E7%BB%AD%E4%BA%A4%E4%BB%98%E5%AE%9E%E6%88%98%E7%94%A8%E6%88%B7%E7%AE%A1%E7%90%86%E6%9C%8D%E5%8A%A1.html" \l "配置正式构建)**配置正式构建**

* 增加参数化构建过程



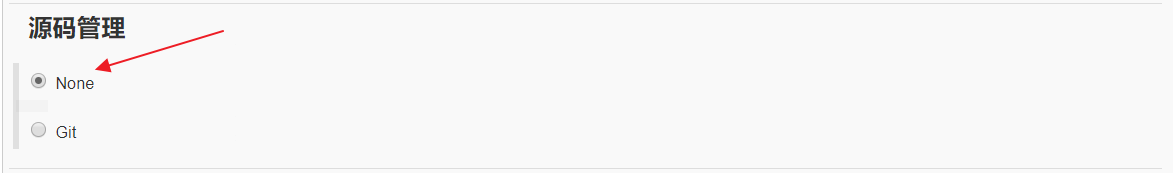


这里使用了 Groovy 脚本来查询最近的 tags 版本，代码如下：/var/jenkins\_home/workspace jenkins容器内的工作空间,spring-eureka是第一次构建任务时git拉取的spring-eureka项目的代码。

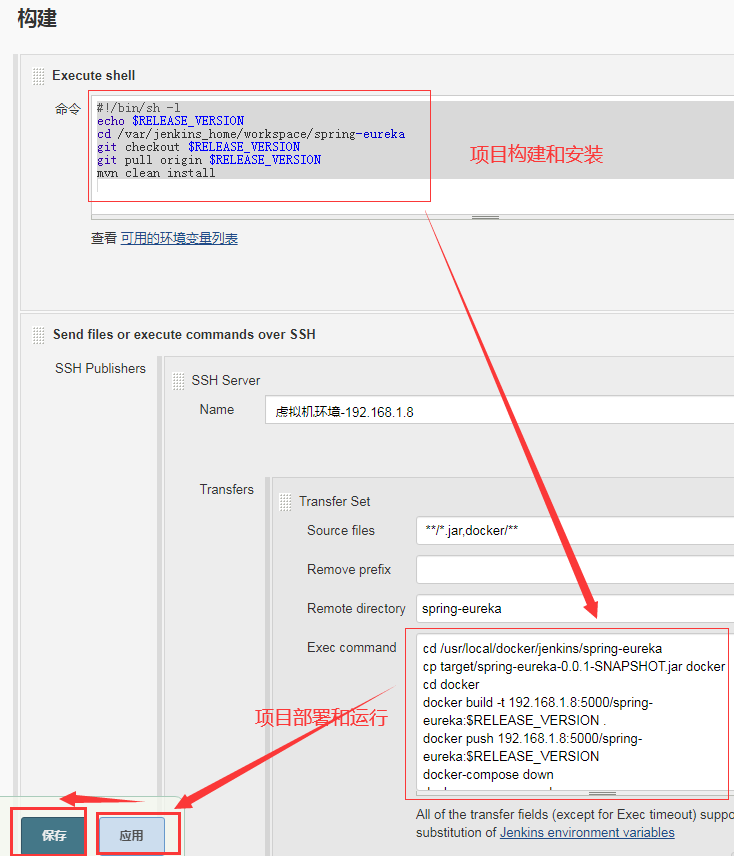
def ver\_keys = [ 'bash', '-c', 'cd /var/jenkins\_home/workspace/spring-eureka;git pull>/dev/null; git remote prune origin >/dev/null; git tag -l|sort -r |head -10 ' ]

ver\_keys.execute().text.tokenize('\n')

* 关闭源码管理:因为第一次也完成构建



* 增加构建步骤



* 使用参数化构建项目

命令:

#!/bin/sh -l

echo $RELEASE\_VERSION

cd /var/jenkins\_home/workspace/spring-eureka

git checkout $RELEASE\_VERSION

git pull origin $RELEASE\_VERSION

mvn clean install

Ecec command：

cd /usr/local/docker/jenkins/spring-eureka

cp target/spring-eureka-0.0.1-SNAPSHOT.jar docker

cd docker

docker build -t 192.168.1.8:5000/spring-eureka:$RELEASE\_VERSION .

docker push 192.168.1.8:5000/spring-eureka:$RELEASE\_VERSION

docker-compose down

docker-compose up -d

docker image prune -f

命令解释: 部署项目到远程机器并运行,此处就部署到jenkins所在机器做测试

#进入到要部署项目的目录

cd /usr/local/docker/jenkins/spring-eureka

#拷贝项目jar包到docker目录(包含docker-compose.yml编排文件)

cp target/spring-eureka-0.0.1-SNAPSHOT.jar docker

cd docker

#把项目构建成docker镜像

docker build -t 192.168.1.8:5000/spring-eureka:$RELEASE\_VERSION .

#把构建的镜想推送到docker镜像仓库

docker push 192.168.1.8:5000/spring-eureka:$RELEASE\_VERSION

docker-compose down

#运行容器启动项目,就可以访问了

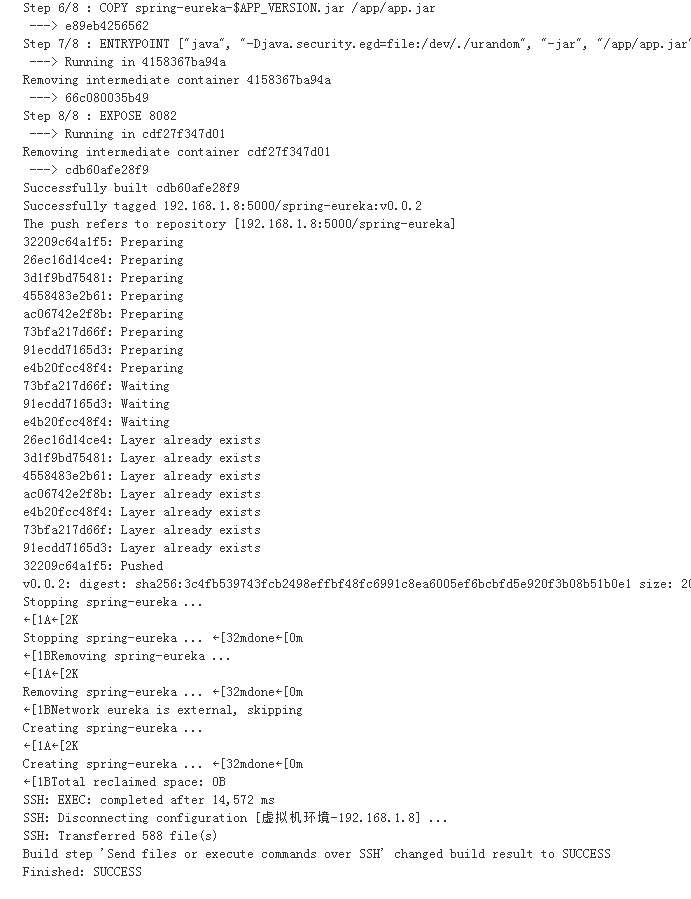
docker-compose up -d

#删除虚悬镜像

docker image prune -f

查看构建日志:





浏览器访问项目:部署机器ip:port/path，比如192.168.1.6:8082启动eureka成功!!!