## 一、持续集成CI

持续集成在敏捷开发中是一种非常推崇的软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误。

### 措施

**减少风险**

一天中进行多次的集成，并做了相应的测试，这样有利于检查缺陷，了解软件的健康状况，减少假定。

**减少重复过程**

减少重复的过程可以节省时间、费用和工作量。说起来简单，做起来难。这些浪费时间的重复劳动可能在我们的项目活动的任何一个环节发生，包括代码编译、数据库集成、测试、审查、部署及反馈。通过自动化的持续集成可以将这些重复的动作都变成自动化的，无需太多人工干预，让人们的时间更多的投入到动脑筋的、更高价值的事情上。

**任何时间、任何地点生成可部署的软件**

持续集成可以让您在任何时间发布可以部署的软件。从外界来看，这是持续集成最明显的好处，我们可以对改进软件品质和减少风险说起来滔滔不绝，但对于客户来说，可以部署的软件产品是最实际的资产。利用持续集成，您可以经常对[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81)进行一些小改动，并将这些改动和其他的代码进行集成。如果出现问题，项目成员马上就会被通知到，问题会第一时间被修复。不采用持续集成的情况下，这些问题有可能到交付前的[集成测试](https://baike.baidu.com/item/%E9%9B%86%E6%88%90%E6%B5%8B%E8%AF%95)的时候才发现，有可能会导致延迟发布产品，而在急于修复这些缺陷的时候又有可能引入新的缺陷，最终可能导致项目失败。

**增强项目的可见性**

持续集成让我们能够注意到趋势并进行有效的决策。如果没有真实或最新的数据提供支持，项目就会遇到麻烦，每个人都会提出他最好的猜测。通常，项目成员通过手工收集这些信息，增加了负担，也很耗时。持续集成可以带来两点积极效果：

(1)有效决策：持续集成系统为项目构建状态和品质指标提供了及时的信息，有些持续集成系统可以报告功能完成度和[缺陷率](https://baike.baidu.com/item/%E7%BC%BA%E9%99%B7%E7%8E%87" \t "_blank)。

(2)注意到趋势：由于经常集成，我们可以看到一些趋势，如构建成功或失败、总体品质以及其它的项目信息。

**建立团队对开发产品的信心**

持续集成可以建立开发团队对开发产品的信心，因为他们清楚的知道每一次构建的结果，他们知道他们对软件的改动造成了哪些影响，结果怎么样。[2] 

### 要素

1.统一的代码库

2.自动构建

3.自动测试

4.每个人每天都要向代码库主干提交代码

5.每次代码递交后都会在持续集成服务器上触发一次构建

6.保证快速构建

7.模拟生产环境的自动测试

8.每个人都可以很容易的获取最新可执行的应用程序

9.每个人都清楚正在发生的状况

10.自动化的部署[1] 

### 原则

1. 所有的开发人员需要在本地机器上做本地构建，然后再提交的版本控制库中，从而确保他们的变更不会导致持续集成失败。

2. 开发人员每天至少向版本控制库中提交一次代码。

3. 开发人员每天至少需要从版本控制库中更新一次代码到本地机器。

4. 需要有专门的集成服务器来执行集成构建,每天要执行多次构建。

5. 每次构建都要100%通过。

6. 每次构建都可以生成可发布的产品。

7. 修复失败的构建是优先级最高的事情。

8. 测试是未来，未来是测试[2]

### 工作流程

当从事变更时，开发者会从基础代码库复制以进行作业，其他开发者提交代码的变更至来源代码库，并透过副本的方式取代来源代码库的代码。不只变更目前（1998年）的代码库，新的代码也可以新增成为程序库、其它共享资源与潜在冲突。

当分支代码保持在取出状态时间越长，当分支代码开发者进行主线重新集成时，就愈容易遭遇集成多重冲突的风险以及失败。当开发者将代码提交到代码库时，首先必须更新代码以反映他们在代码库中的更改，因为他们拿到了副本。代码库包含的更改越多，开发人员在提交自己的更改前必须运行的工作越多。

终于，该程序库也许变成非常不同于开发者的目标代码，他们进入有时候被称为合并地狱或集成地狱的阶段，这时候开发者所花费的集成时间，将超过最初代码开发的时间。

持续性集成涉及预先集成与预先与经常性的集成，借此来避免踩到集成地狱的陷阱，实践的目标是减少重工、减少成本与时间。

持续性集成补充的实践是在提交成果之前，每个开发人员必须运行一个完整的构建与运行及通过所有的单元测试、集成测试，这些都是当持续性集成服务器侦测到代码有新的提交时，必须经常性与自动化的进行。[2]

参考资料

* 1.Booch, Grady. Object Oriented Design: With Applications. Benjamin Cummings. 1991: 209 [18 August 2014]. ISBN 9780805300918.
* 2.Grady Booch. Object-Oriented Analysis and Design with applications (2nd edition, 15th printing) (PDF). December 1998

## 二、Jenkins

Jenkins是一款开源 CI&CD 软件，用于自动化各种任务，包括构建、测试和部署软件。

Jenkins 支持各种运行方式，可通过系统包、Docker 或者通过一个独立的 Java 程序。

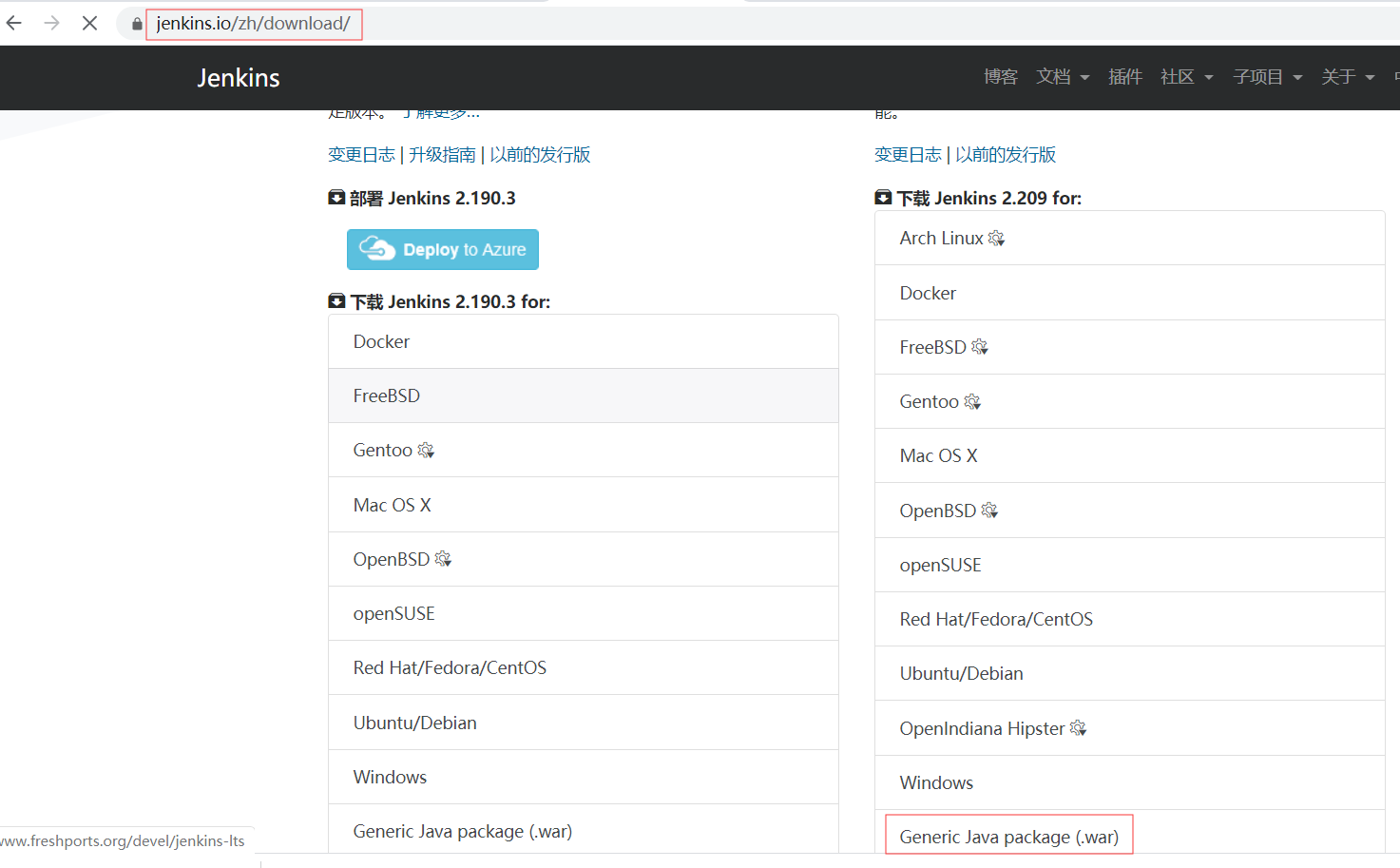


### 在CentOS下的下载及安装Jenkins

* 下载Jenkins

官网：jenkins.io

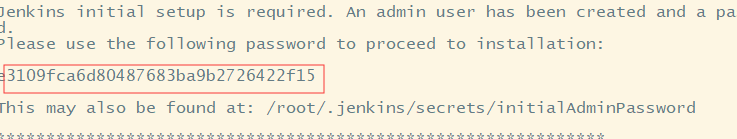
wget http://mirrors.jenkins.io/war/latest/jenkins.war



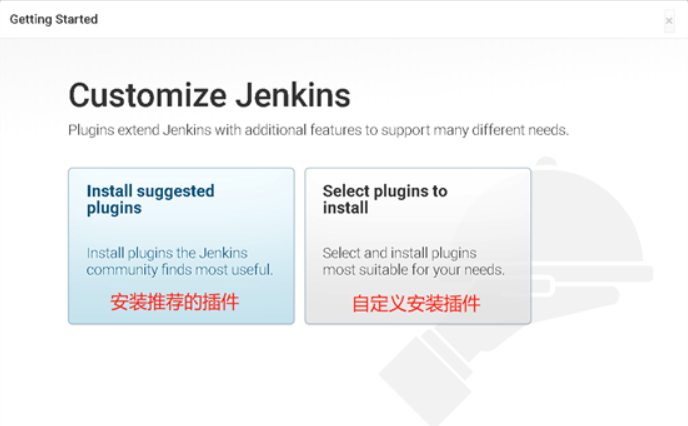
* 启动jenkins：java -jar jenkins.war
* 访问:http://localhost:8080通过浏览器访问，

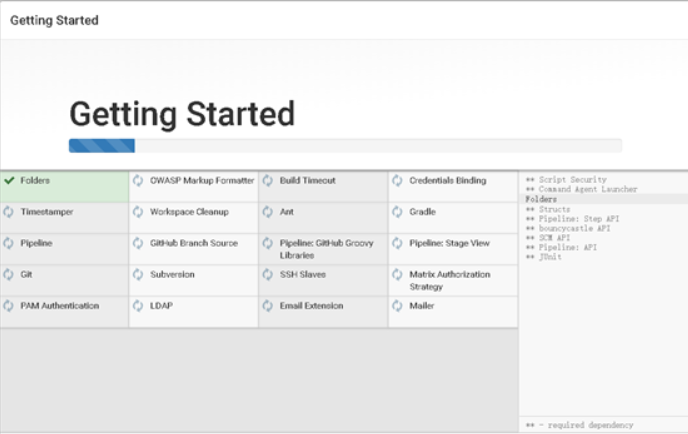
启动会生成初始密码进行解锁，按提示路径找到密码文件，复制内容提交解锁即可





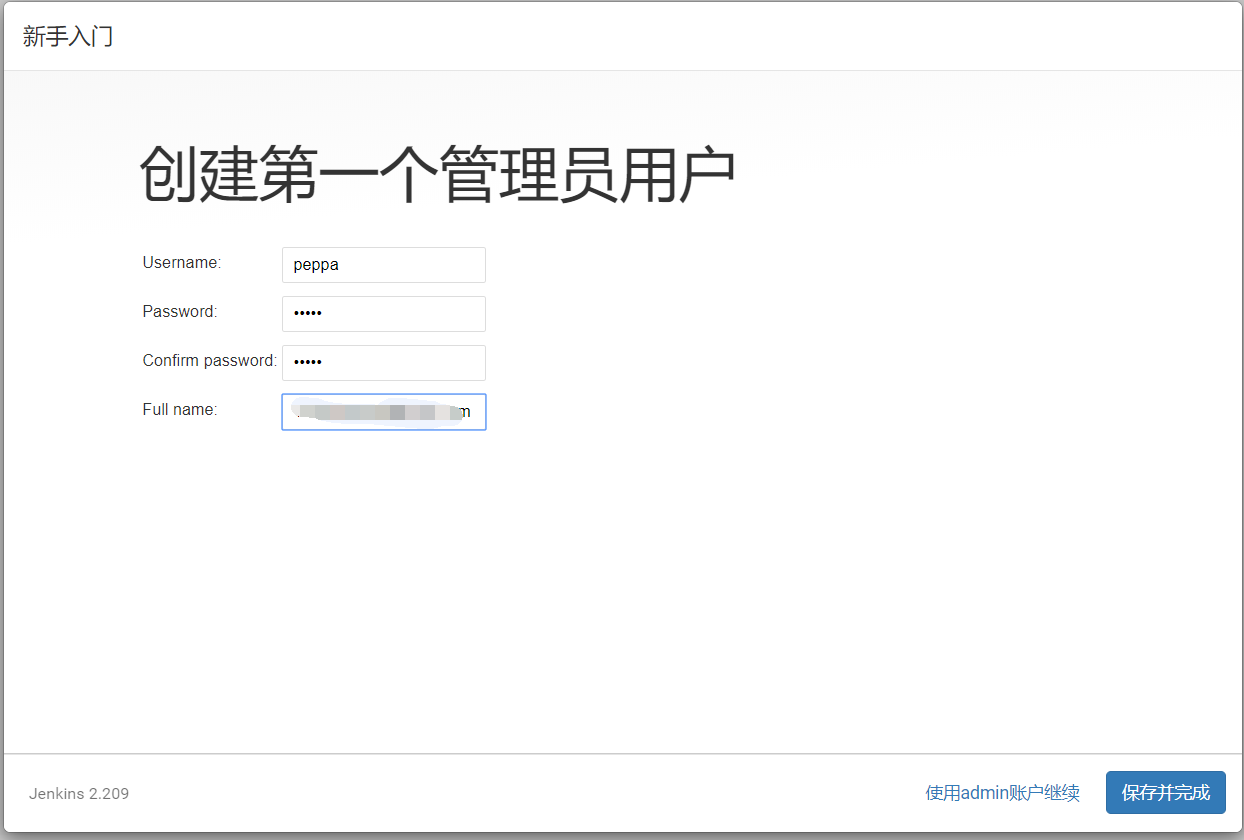
* Jenkins提供了很丰富的插件，为了使用方便，建议安装建议的插件





需要一定时间，有可能还会失败（网络等原因），也可以直接跳过

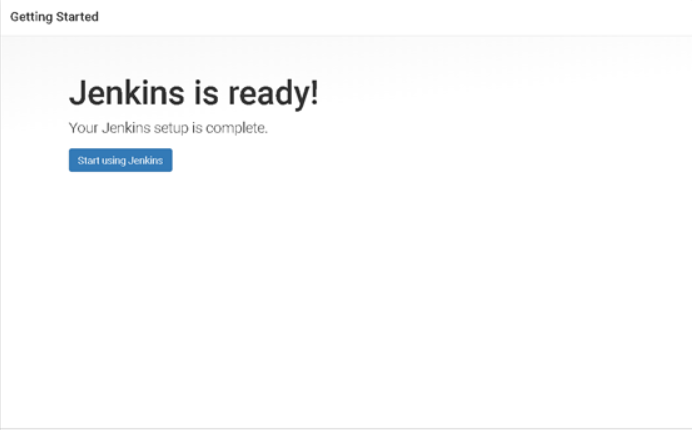
* 创建第一个管理员用户



* 配置Linux连接实例



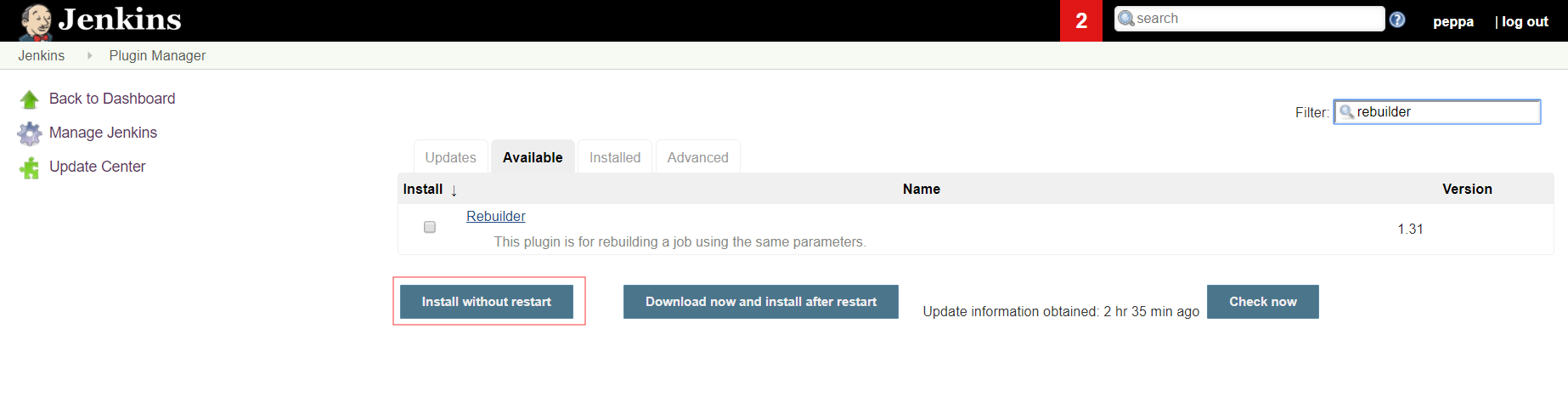
* 填写好用户信息，点击Save and Finish



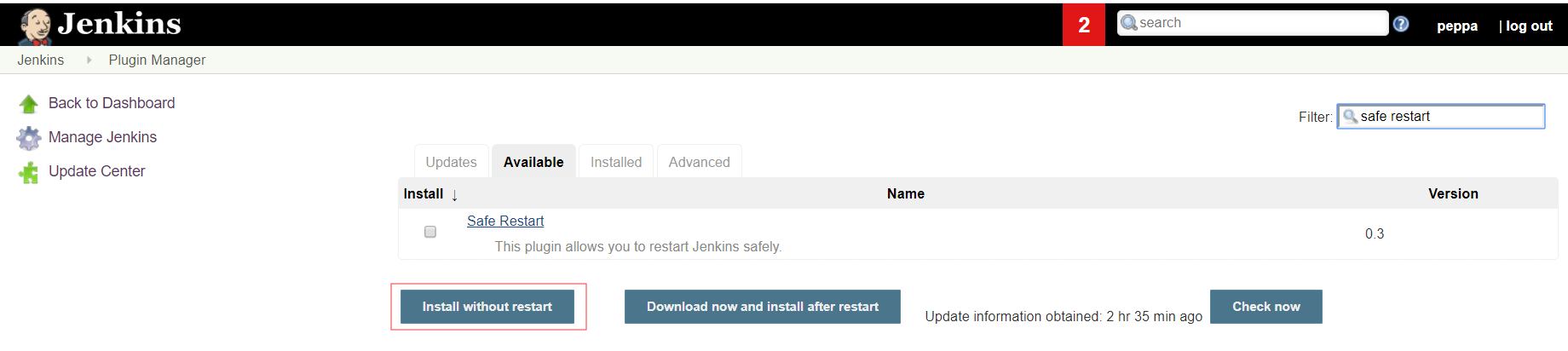
然后就可以开始Jenkins之旅了。

### 安装插件

**系统管理 --> 管理插件 --> 可选插件 --> 搜索rebuilder**  
Rebuilder插件用户重新构建，构建时输入很多参数，每次输入参数很麻烦，再次构建的时候使用这个插件会比较方便少输入很多参数



**系统管理 --> 管理插件 --> 可选插件 --> 搜索safe restart**  
这个插件是为了帮助你安全的重启Jenkins，Jenkins做了某些操作的时候需要重启



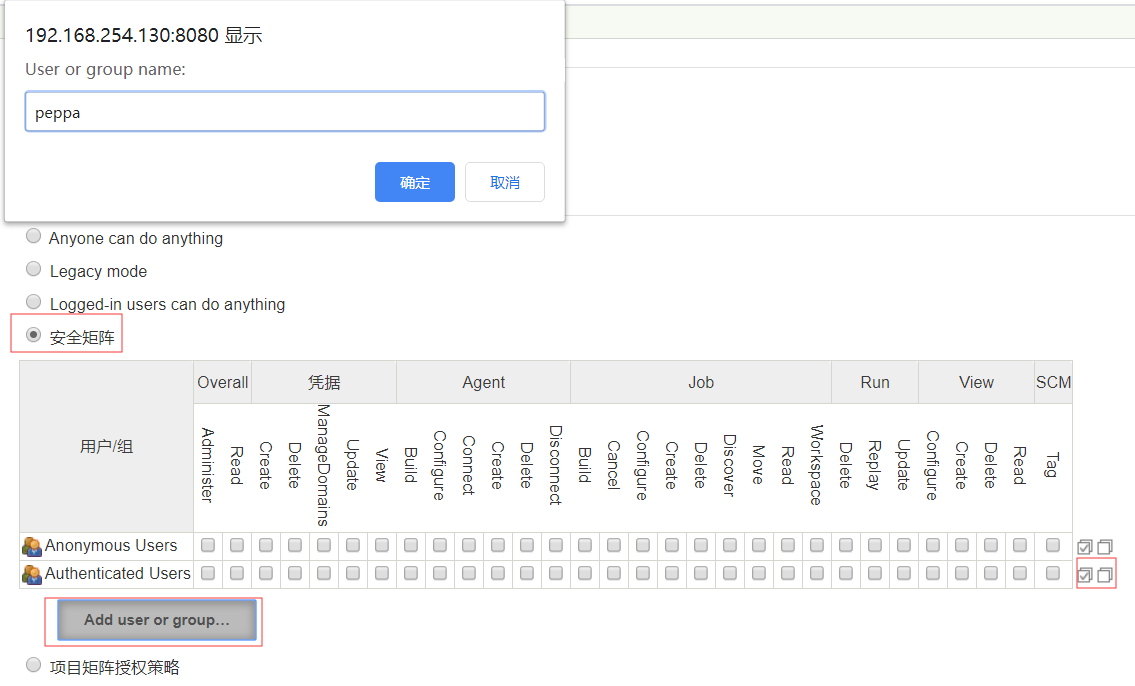
### Jenkins配置全局安全属性并添加自定义用户

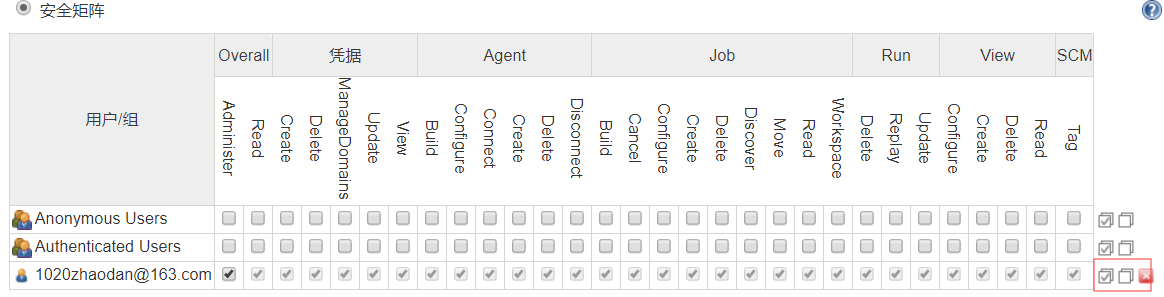
* 配置全局安全属性

**在Jenkins中默认中全局安全性是用登录之后可以做任何事情，拥有全部的权限，这种配置不是很安全，可以设置安全组**在系统管理 --> 全局安装配置



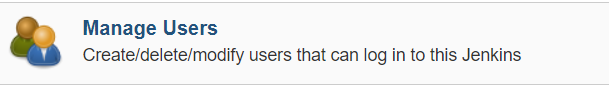
选择安全矩阵 --> Add user or group



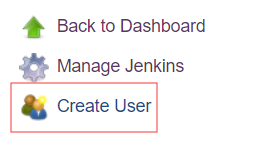


* 添加新用户

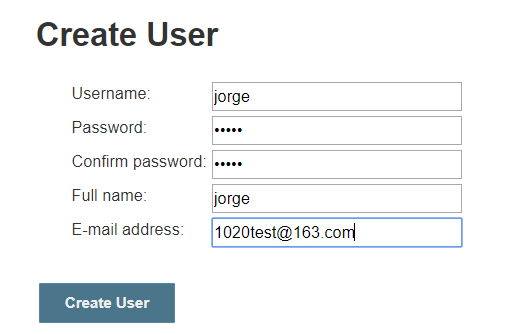
**在系统管理 --> 添加用户，可以看到当前用户只有peppa用户一个**



新建用户



新建用户成功之后



然后配置其除了管理员以外的权限



这样jorge用户就拥有了除了系统管理之外的所有权限  
使用刚刚创建的jorge用户重新登录Jenkins，可以看到左侧功能栏就没有系统管理功能了

## 三、应用部署服务器准备

### 1.Linux系统

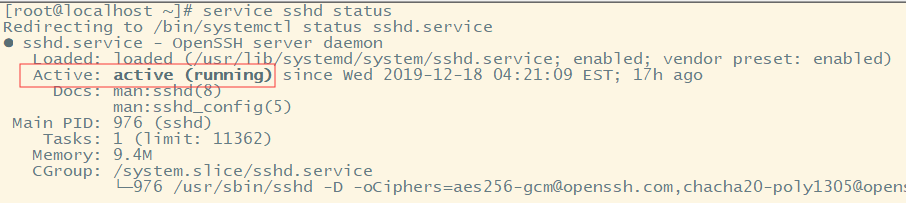
查看资料安装虚拟机，安装习惯使用的Linux版本

2查看ip：192.168.154.130

准备用户名和密码（root）

3.ssh协议开启

查看：**service sshd status**



如没有开启，运行**service sshd start**开启

开放指定端口：22 80 8080 3306

或者关闭防火墙 service iptables stop



### 2. 安装JDK

Oracle官网下载Linux对应jdk版本，推荐下载JDK11.

<https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>

下载前要同意其声明

## 

* 创建文件夹用于存放java环境:

mkdir -p /usr/local/java

cd /usr/local/java

通过远程终端上传到该文件夹，解压: tar -zxvf jdk-11.0.5\_linux-x64\_bin.tar.gz

* 配置环境变量

vi /etc/profile

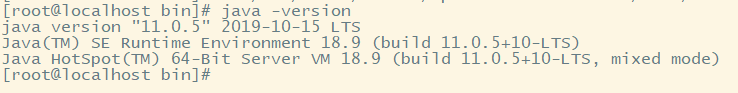
export JAVA\_HOME=/usr/local/java/jdk-11

export PATH $JAVA\_HOME/bin:$PATH

* 重新加载环境变量

source /etc/profile

* 输入 java -version验证java环境是否正确配置



### 3.安装Git并配置

* 安装git：

yum install git

* Git配置

git config –global user.name “你的用户名”

git config –global user.email “你的邮箱

* 创建Linux服务器的SSH Key

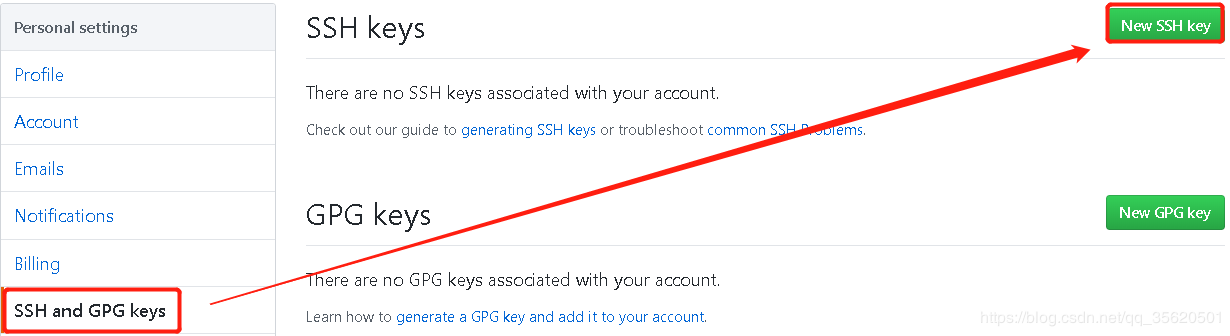
ssh-keygen -t rsa -C “email”

（过程中会要求你填写密码等，可以为空）

* 将SSH Key配置到GitHub/码云/coding上，保证Linux服务器与GitHub等之间可以连通

进入GitHub

登录GitHub --> 点击右上角的头像 --> Settings --> SSH and GPG keys --> New SSH Key



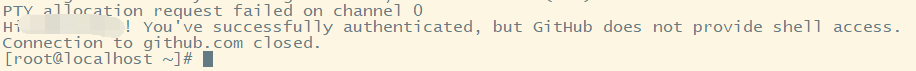
填写名称和SSH公钥

首先先确认在/root/.ssh/下有id\_rsa和id\_rsa.pub，然后将id\_rsa.pub的内容复制到Key中

* 测试GitHub和Linux是否连通

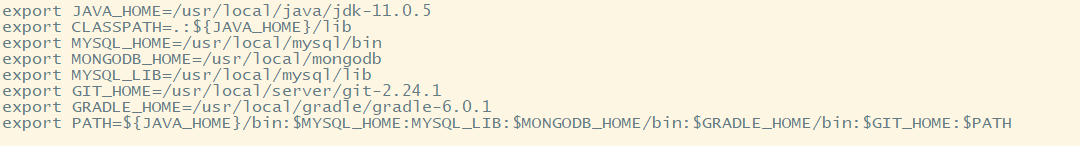
ssh git@github.com

出现You’ve successfully authenticated表示连通成功



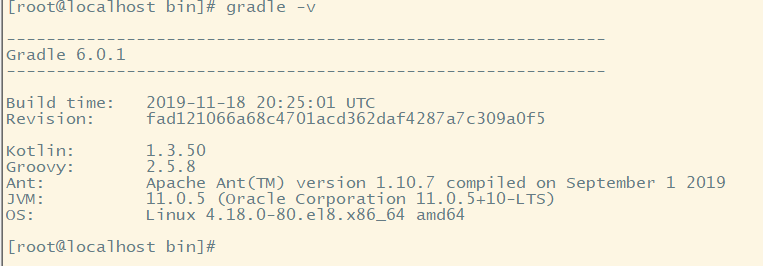
### 4.构建工具Gradle安装

下载Gradle，上传至定义文件夹，解压，参照JDK配置环境变量



重新加载配置：source /etc/profile

gradle -v验证



### 5.安装应用服务器Tomcat

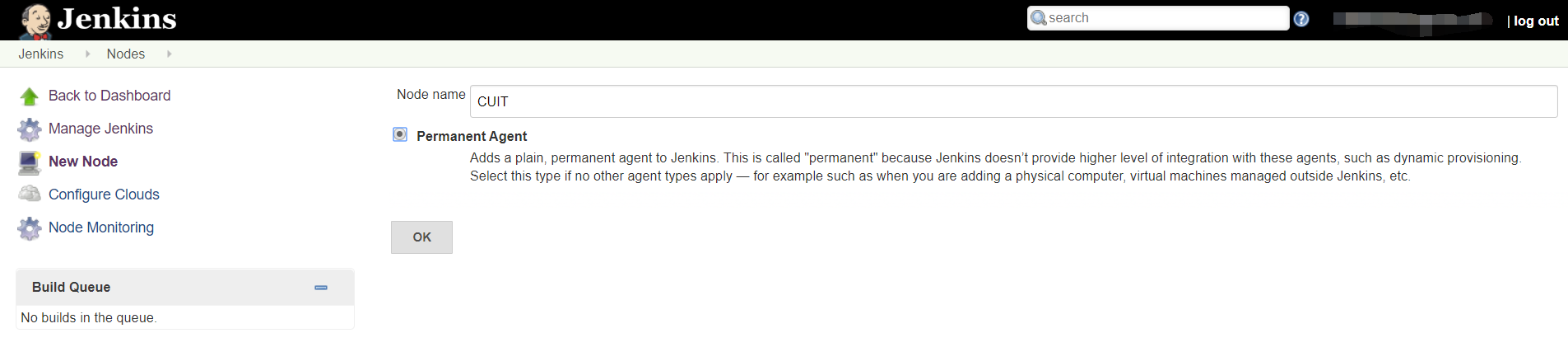
下载，上传，解压，修改默认端口号，避免和jenkins冲突



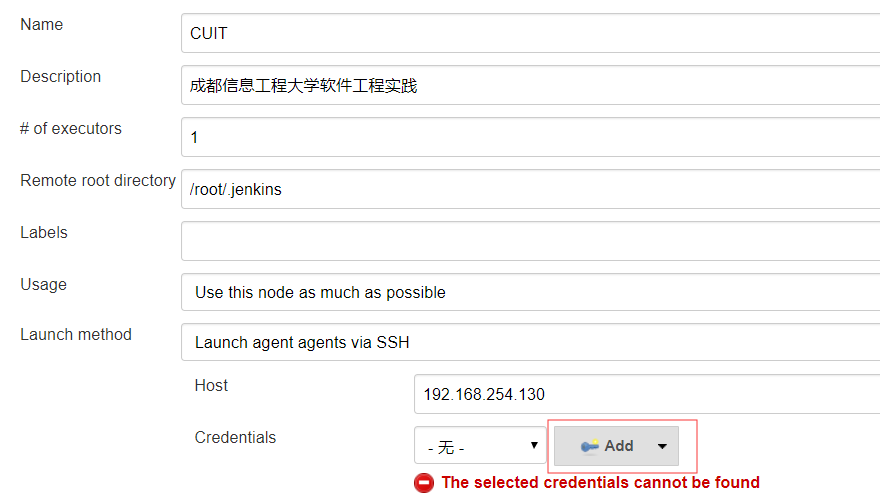
### 6．将Linux服务器注册到Jenkins节点中

* 新建节点

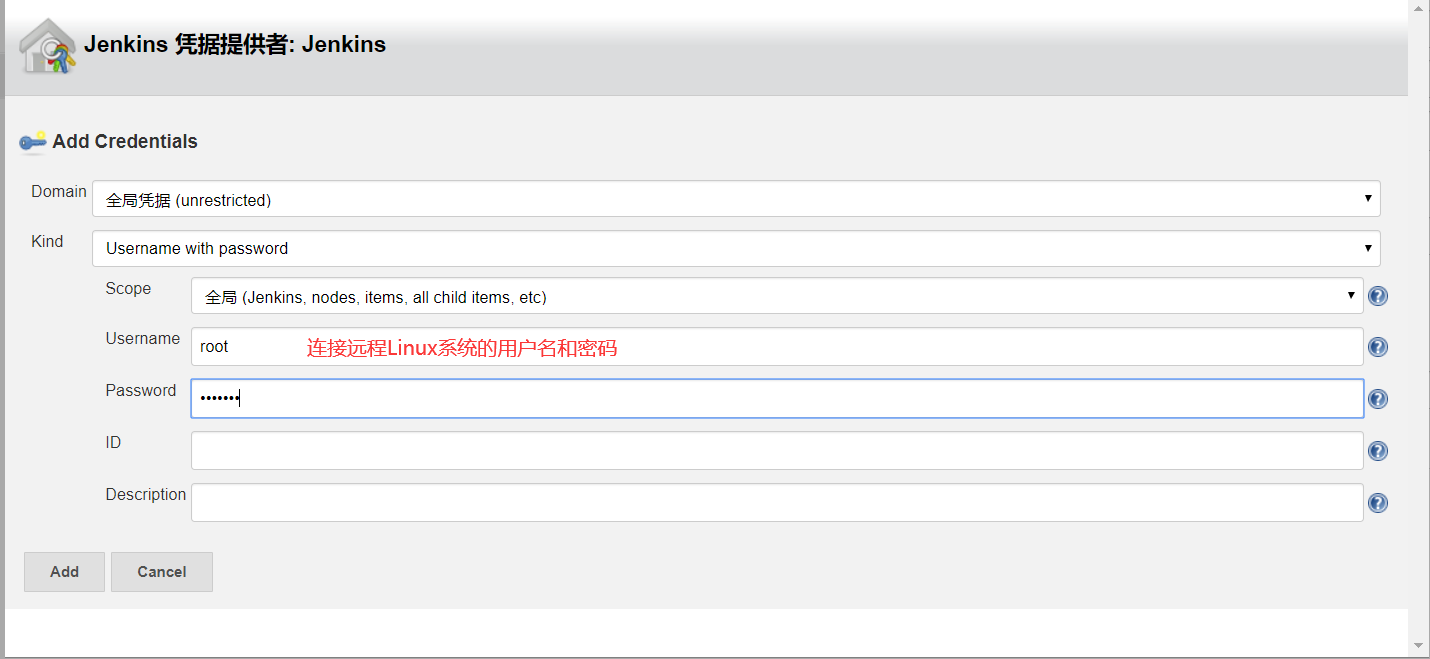
系统管理 --> 管理节点 --> 新建节点



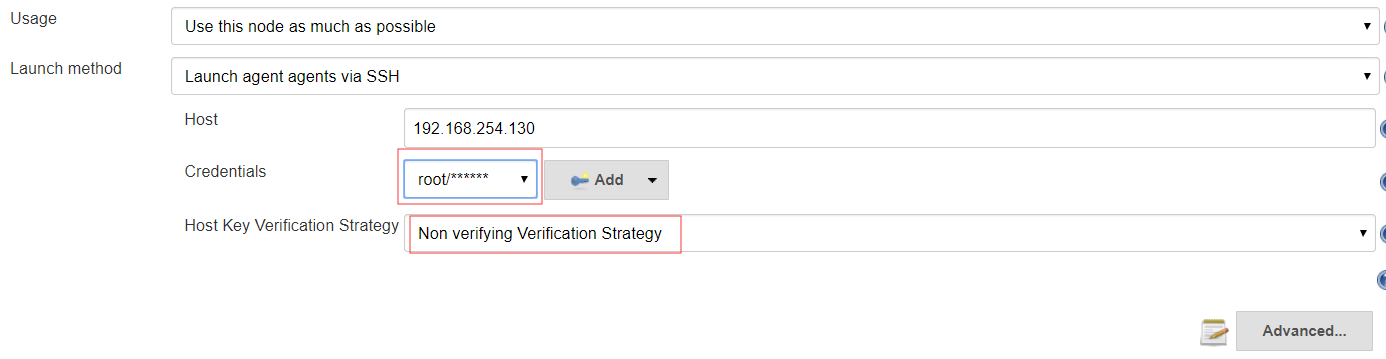
然后填写节点信息



然后点击Add --> Jenkins



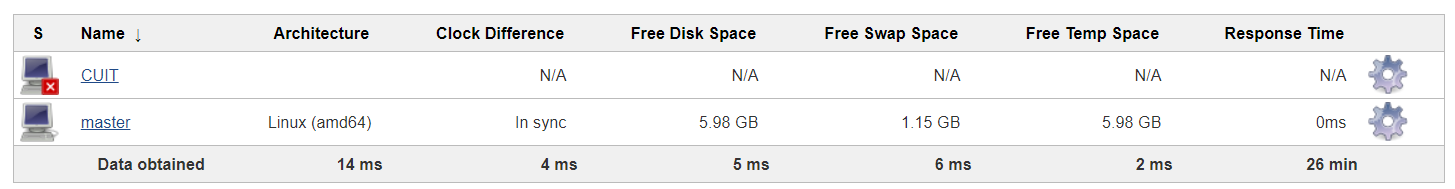
继续填写信息，在高级中配置Linux服务器java的路径，然后点击保存



高级中配置java

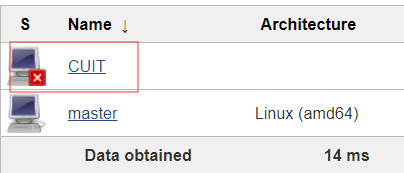


保存成功之后

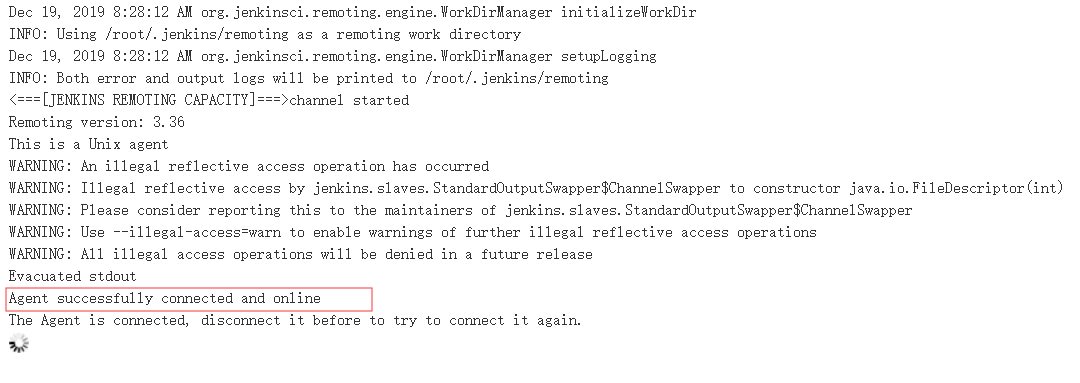


* 测试服务器注册到Jenkins上

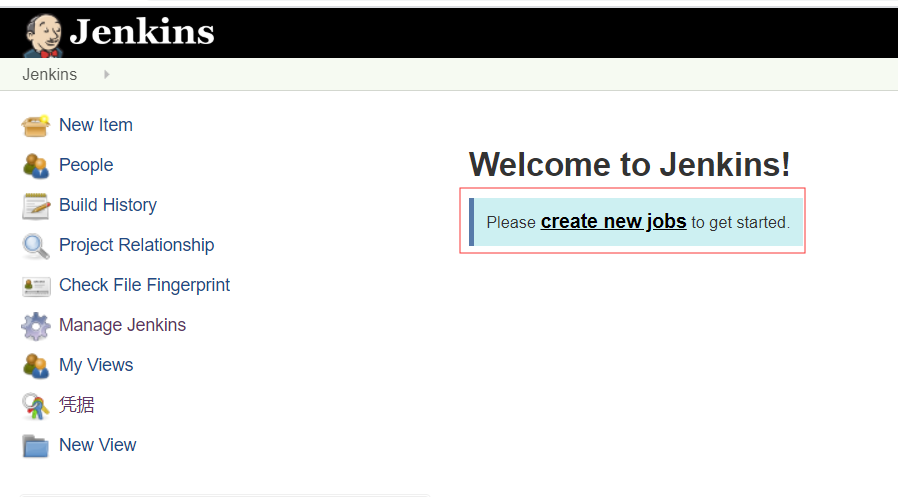
点击CUIT



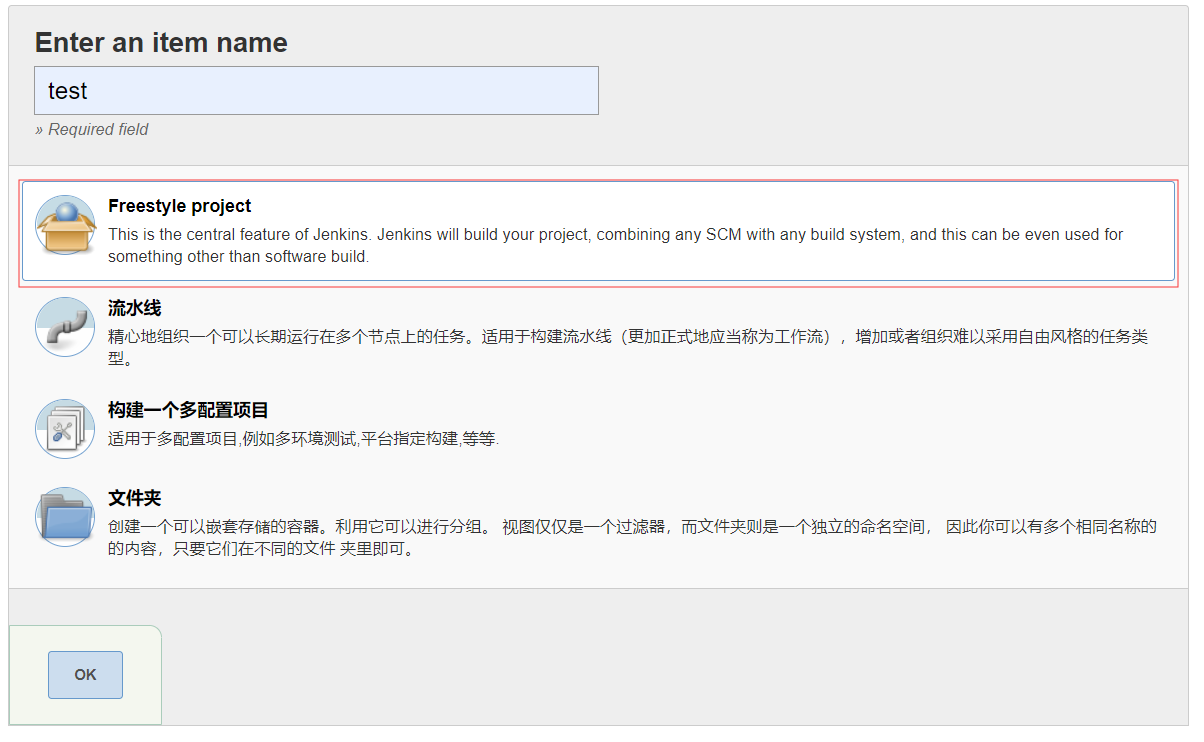
点击launch agent



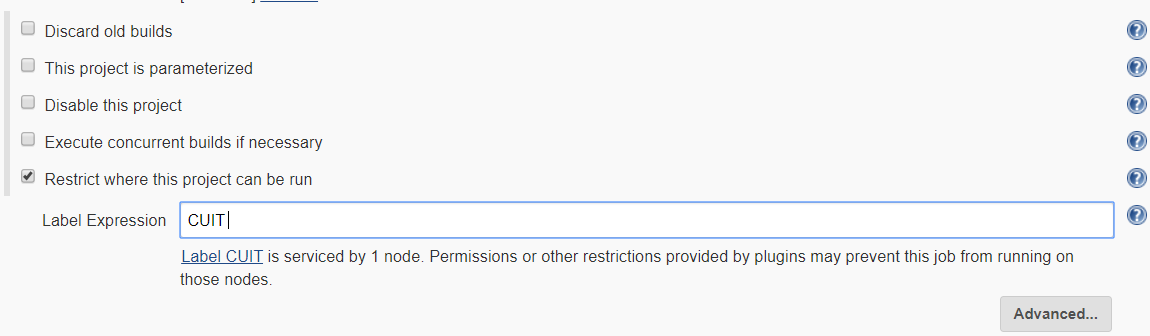
创建一个新任务



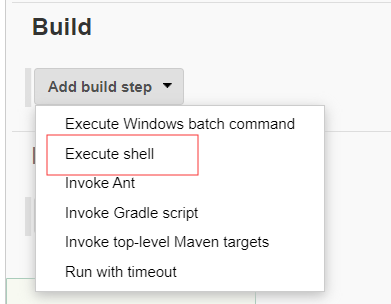
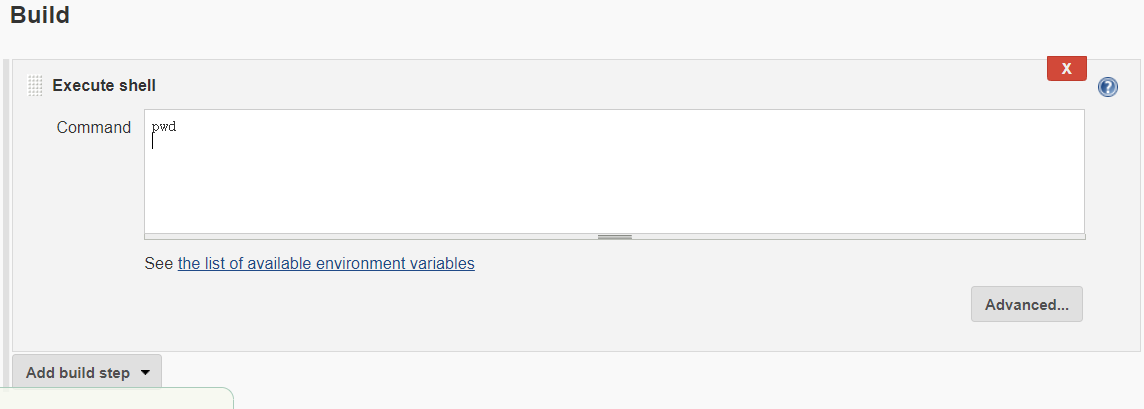
输入任务名称和任务类型

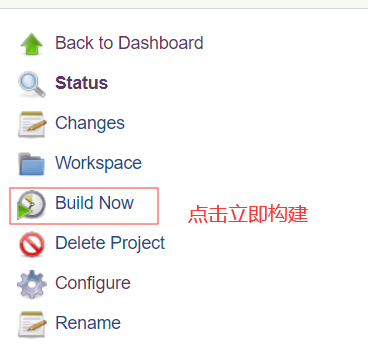


设置任务可以在哪个服务器上运行



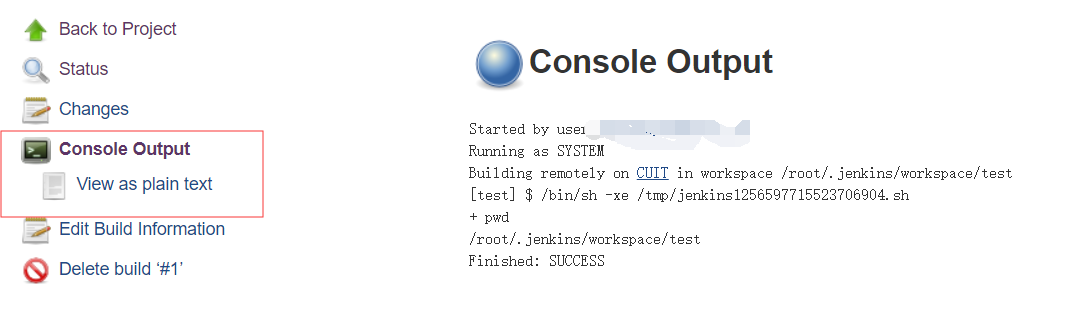
在构建中选中Execute shell  
在Command中输入ifconfig（这是用来输出服务器的网络信息的命令），然后保存



查看任务控制台输出日志





### 创建Jenkins自动化部署任务(将项目运行在Tomcat中)

* 新建任务

新建  
  
输入任务名称和任务类型

填写任务参数

配置Git参数

然后在构建中添加shell脚本

# 部署完server的任务，server这个进程不会被Jenkins杀掉

BUILD\_ID=DONTKILLME

# 加载配置参数

source /etc/profile

# 配置运行参数

export PROJ\_PATH=`pwd`

export TOMCAT\_APP\_PATH=/usr/local/server/apache-tomcat-9.0/webapps

# bash函数

killTomcat()

{

pid=`ps -ef|grep tomcat|grep java|awk '{print $2}'`

echo " tomcat Id list :$pid"

if [ "$pid" = "" ]

then

echo "no tomcat pid alive"

else

kill -9 $pid

fi

}

# 切换到任务下的指定目录

cd $PROJ\_PATH/book-1.0-SNAPSHOT

# 将从GitHub拉取的项目进行清理并安装在本地

gradle build

# 杀死原有的tomcat进程

killTomcat

# 删除原有工程

rm -rf $TOMCAT\_APP\_PATH/webapps/ROOT

rm -f $TOMCAT\_APP\_PATH/webapps/ROOT.war

rm -f $TOMCAT\_APP\_PATH/webapps/book-1.0-SNAPSHOT.war

# 复制新的工程

cp $PROJ\_PATH/book-1.0-SNAPSHOT/target/book-1.0-SNAPSHOT.war $TOMCAT\_APP\_PATH/webapps/

# 切换到指定目录

cd $TOMCAT\_APP\_PATH/webapps/

# 为war包重命名

mv book-1.0-SNAPSHOT.war ROOT.war

# 启动tomcat服务器

cd $TOMCAT\_APP\_PATH/bin

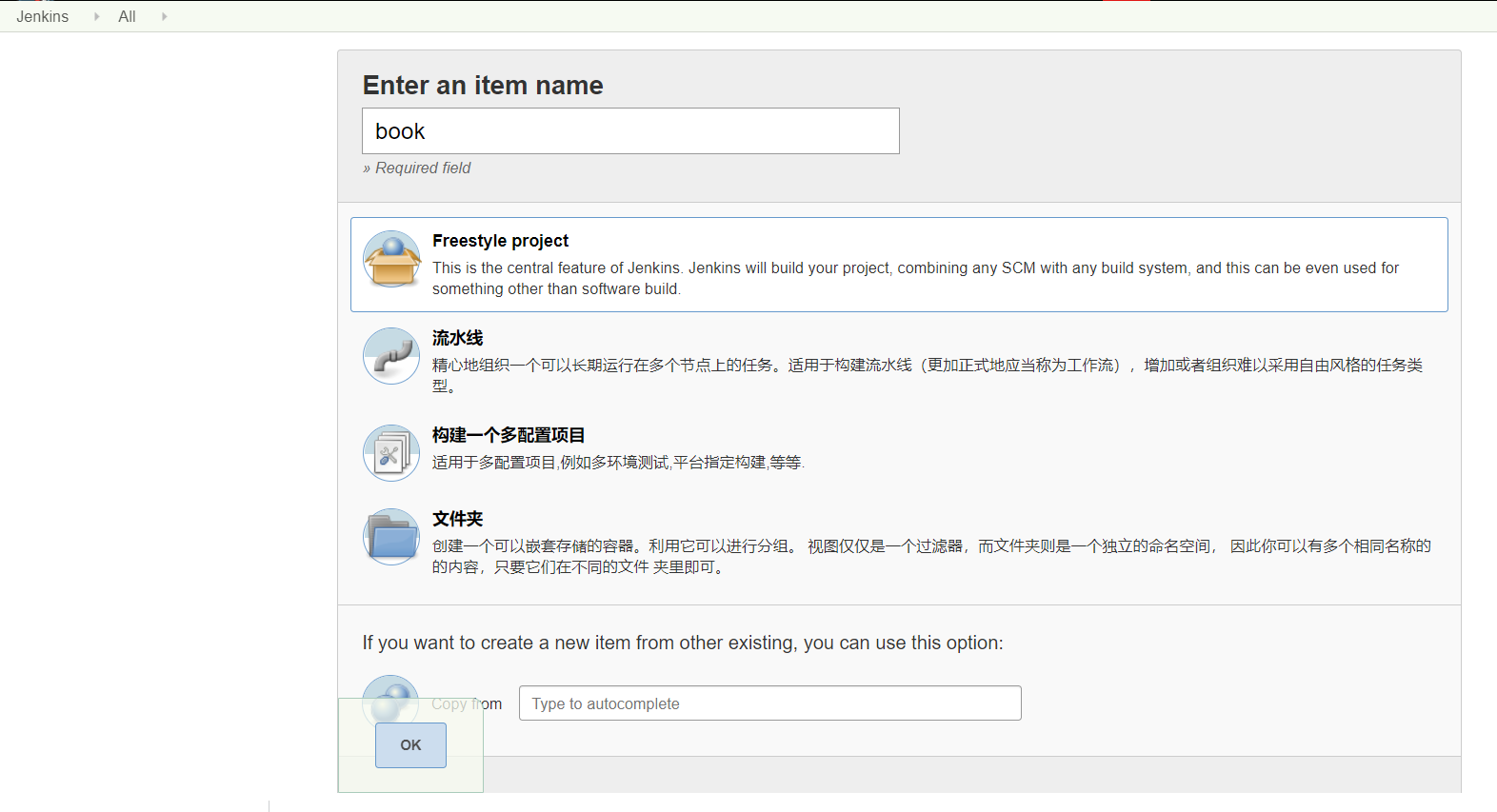
sh startup.sh

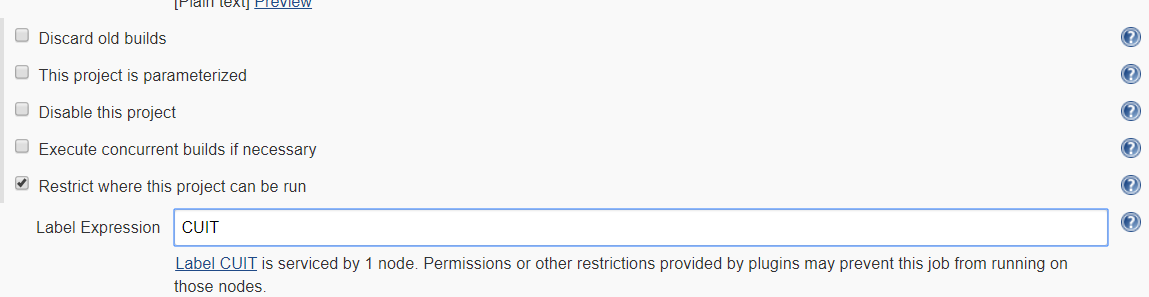
* 构建任务测试是否配置成功

立即构建该任务，然后查看控制日志输出

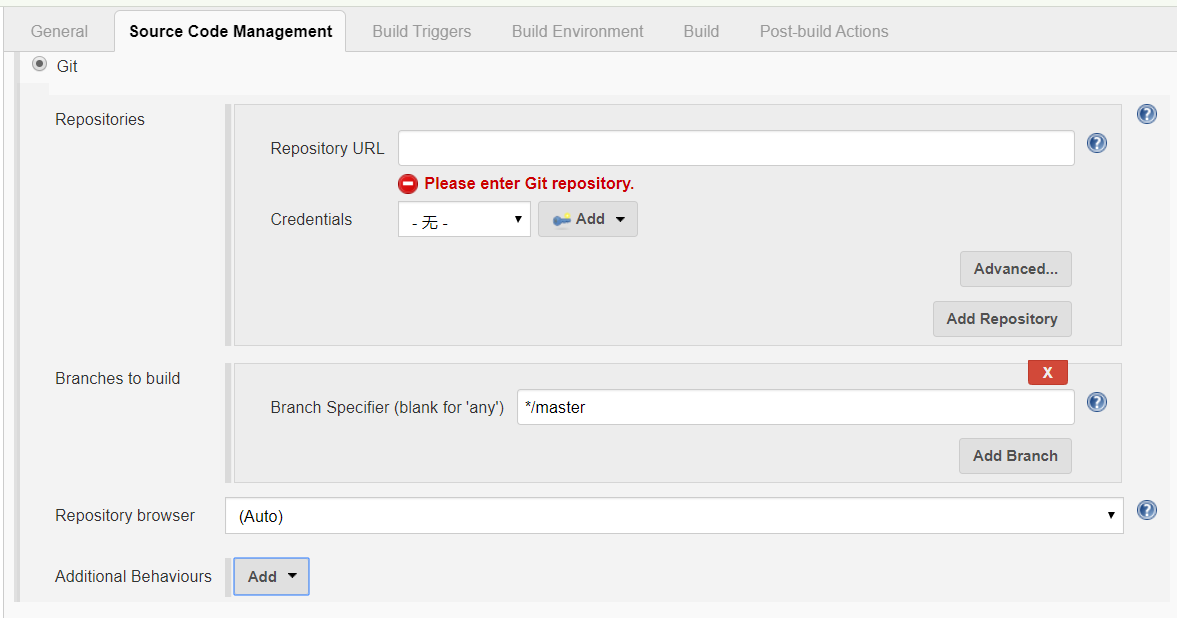
SpringBoot项目创建Jenkins自动化部署任务)

* 新建任务（输入任务名称和任务类型）

  
填写任务参数



配置Git参数



然后在构建中添加shell脚本

下面是完整的shell脚本，需要更改的参数有：

1、BOOK\_PATH表示你使用的服务器中指定的jar存放路径

2、book-1.0-SNAPSHOT是我这个项目存放的文件夹的名称（在上面Local subdirectory for repo中已经设置为book-1.0-SNAPSHOT），需要换成你们的文件夹名称

# 部署完server的任务，server这个进程不会被Jenkins杀掉

BUILD\_ID=DONTKILLME

# 加载配置参数

. /etc/profile

# 配置运行参数

export PROJ\_PATH=`pwd`

export BOOK\_PATH=/data/book

# base函数

kill()

{

pid=`ps -ef|grep book|grep java|awk '{print $2}'`

echo "book Id list :$pid"

if [ "$pid" = "" ]

then

echo "no book pid alive"

else

kill -9 $pid

fi

}

cd $PROJ\_PATH/book-1.0-SNAPSHOT

gradle build

# 杀死原有的book进程

kill

# 删除原有工程

rm -f $BOOK\_PATH/book-1.0-SNAPSHOT.war

# 复制新的工程到指定目录

cp $PROJ\_PATH/ book-1.0-SNAPSHOT/target/ book-1.0-SNAPSHOT.jar $BOOK\_PATH/

# 切换到指定目录

cd $ BOOK \_PATH/

# springboot项目

nohup java -jar book-1.0-SNAPSHOT.jar > nohup.out &  
构建任务测试是否配置成功

立即构建该任务，然后查看控制日志输出