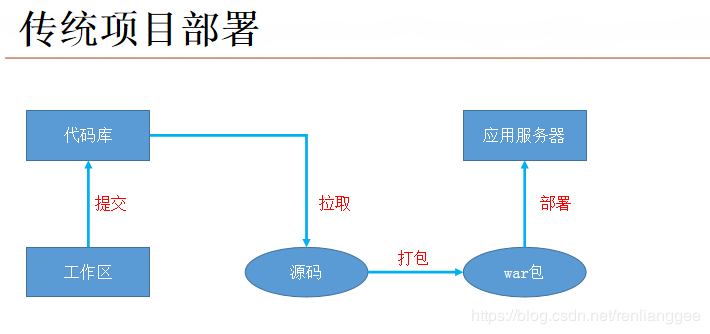
Jenkins

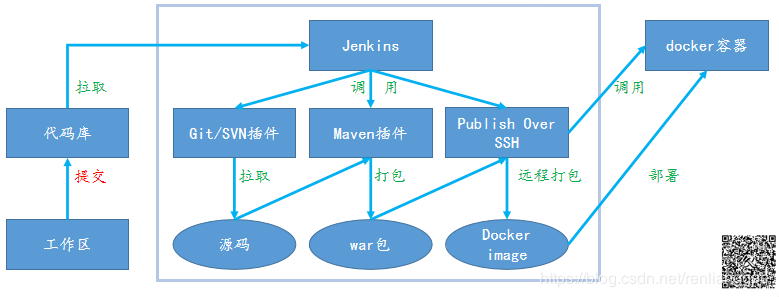
1. **概述**

传统项目部署



**持续集成概念**：随着软件开发复杂度的不断提高，团队开发成员间如何更好地协同工作以确保软件开发的质量已经慢慢成为开发过程中不可回避的问题。尤其是近些年来，敏捷（Agile） 在软件工程领域越来越红火，如何能再不断变化的需求中快速适应和保证软件的质量也显得尤其的重要。 持续集成CI(Continuous Integration)正是针对这一类问题的一种软件开发实践。它倡导团队开发成员必须经常集成他们的工作，甚至每天都可能发生多次集成。而每次的集成都是通过自动化的构建来验证，包括自动编译、发布和测试，从而尽快地发现集成错误，让团队能够更快的开发内聚的软件。

持续集成项目部署



**持续集成的价值**

1.持续集成中的任何一个环节都是自动完成的，无需太多的人工干预，有利于减少重复过程以节省时间、费用和工作量；

2.持续集成保障了每个时间点上团队成员提交的代码是能成功集成的。换言之，任何时间点都能第一时间发现软件的集成问题，使任意时间发布可部署的软件成为了可能；

3.持续集成还能利于软件本身的发展趋势，这点在需求不明确或是频繁性变更的情景中尤其重要，持续集成的质量能帮助团队进行有效决策。

**持续集成的原则**

1.需要版本控制软件保障团队成员提交的代码不会导致集成失败。常用的版本控制软件有 Git、CVS、Subversion 等；

2.开发人员必须及时向版本控制库中提交代码，也必须经常性地从版本控制库中更新代码到本地；

3.需要有专门的集成服务器来执行集成构建。根据项目的具体实际，集成构建可以被软件的修改来直接触发，也可以定时启动，如每半个小时构建一次。

1. **安装**

首先我们服务器分为Jenkins服务器，部署服务器

Jenkins服务器用于放置Jenkins相关应用，部署服务器放置我们需要部署的应用，目前Jenkins服务器和部署服务器都在同一台机器，部署服务器使用物理机本身，Jenkins服务器使用的是docker部署，因此也相当于是两台不同的机器。Gitlab也是通过docker部署

后面我会用**服务器A**代表Jenkins服务器，服务器B代表部署服务器

1. **需要准备的环境**

Jenkins

Jdk

Maven

Gitlab

Docker

1. **需要安装的Jenkins插件**

git client plugin( git的客户端插件)

gitlab plugin （gitlab插件，配置gitlab信息）

junit plugin（Java测试插件，测试Java代码）

maven intergration plugin（maven项目插件）

pipeline（流水线插件，项目构建流程插件）

publish over ssh（远程发布通过ssh）

subversion plug-in（svn插件，配置svn相关信息）

Timestamper（日期插件，让jenkins console日志带时间戳）

Workspace Cleanup Plugin（build开始前或build完成后清理workspace）

Mailer Plugin（邮件插件，可以构建完成后发送邮件）

docker-build-step（docker插件，可以配置docker相关信息）

1. **安装**

服务器A

我们是通过docker来安装Jenkins

步骤1：拉取Jenkins镜像，我们选择jenkins/jenkins最新镜像，因为人气比较旺

docker pull jenkins/jenkins

步骤2:由于该镜像只有jdk环境，不包含maven，不能直接使用，在此镜像基础上，我们通过dockerfile重新打包了一个包含maven的新镜像

dockerFile命令如下

|  |
| --- |
| FROM jenkins/jenkins  ADD apache-maven-3.6.3 /usr/local/ #将物理机的maven打入docker镜像  ENV  MAVEN\_HOME=/usr/local/apache-maven-3.6.3 #设置maven的home路径  ENV  PATH=$JAVA\_HOME/bin:$MAVEN\_HOME/bin:$PATH #设置环境变量  USER root #用户  RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers  USER jenkins |

服务器B

步骤3：gitlab安装

通过docker镜像拉取最新版本

docker pull gitlab/gitlab-ce

运行

|  |
| --- |
| docker run --detach --hostname gitlab.example.com --publish 5443:443 --publish 80:80 --publish 2222:22  --name gitlab  --restart always --volume /data/westone/gitlab/config:/etc/gitlab  --volume /data/westone/gitlab/logs:/var/log/gitlab --volume /data/westone/gitlab/data:/var/opt/gitlab  f2e48729e35c |

--hostname:设置gitlab主机名

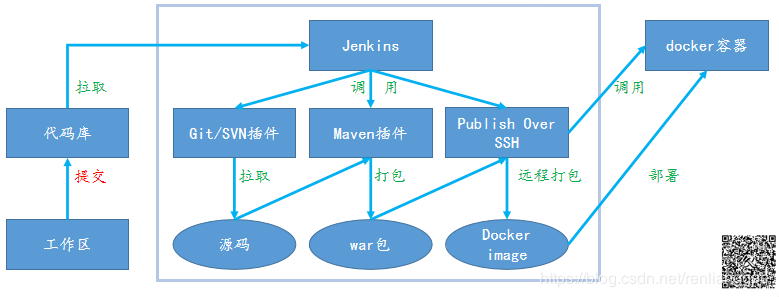
--publish：发布的端口号映射；80是HTTP访问gitlab的端口号，443是https访问端口，22是ssh端口

--restart always：自动重启

--volume：等同于-v挂载容器卷

--detach:等同于-d，保持后台运行

1. **流程**



1）、用户登陆Jenkins管理界面，点击构建按钮



2）、Jenkins通过配置的gitlab地址和账户信息，从gitlab服务器拉取项目代码到Jenkins的workspace空间

3）、Jenkins通过配置的maven信息，去workspace空间找到指定的项目的pom.xml文件，通过maven去编译打包。我们工程使用的是springboot项目，因此会打包成jar

4）、Jenkins再通过ssh，登陆远程服务器，也就是我们的服务器B，将打好的jar包传输到服务器B（这些信息是在Jenkins上配置的）

5）、Jenkins再执行服务器B上的脚本（脚本位置也是通过Jenkins配置的），脚本内容包含将jar包通过docker打包成docker镜像，然后并运行启动该镜像。

6）、前5步都没出错的话，Jenkins返回构建成功，此时的项目已经部署完成。

1. **jenkins配置部署**
2. **Jenkins启动**

1）、运行

|  |
| --- |
| docker run --name my-jenkins --user=root -p 8080:8080 -p 50000:50000 -v /opt/data/jenkins\_home:/var/jenkins\_home -d jenkins/jenkins:latest |

--name:容器名称

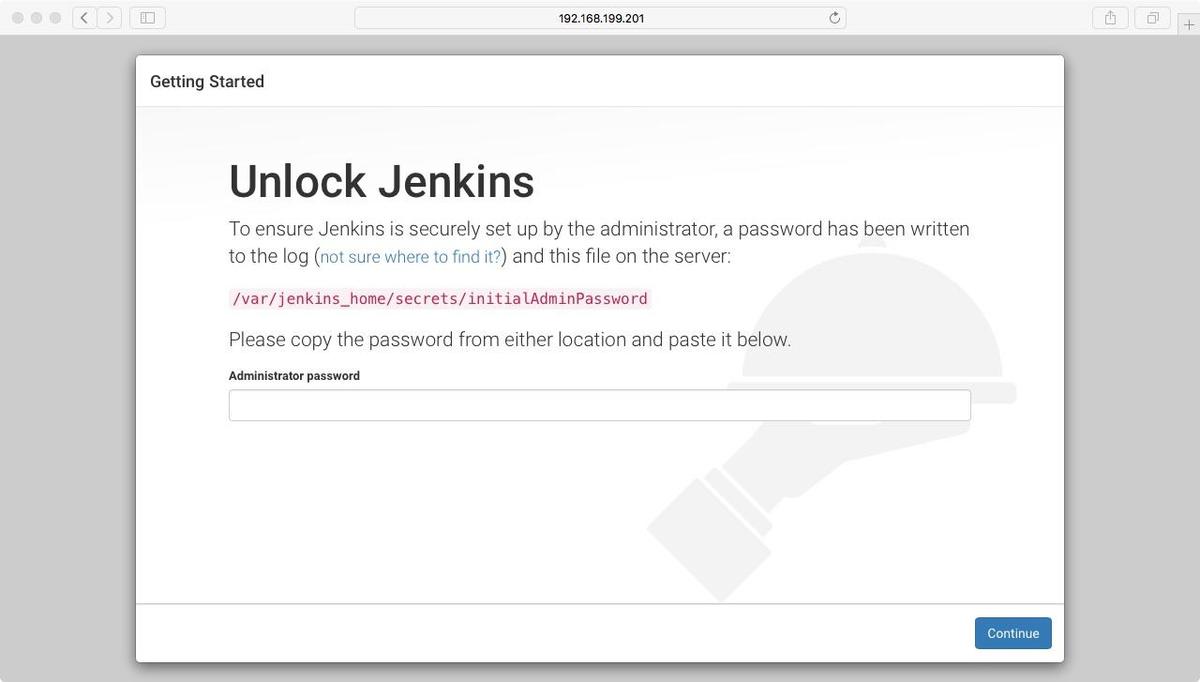
--user:容器用户

-p: 8080 jenkins访问端口 ；50000基于JNLP的Jenkins代理默认通过TCP端口50000与Jenkins主机通信

-v:挂载容器卷

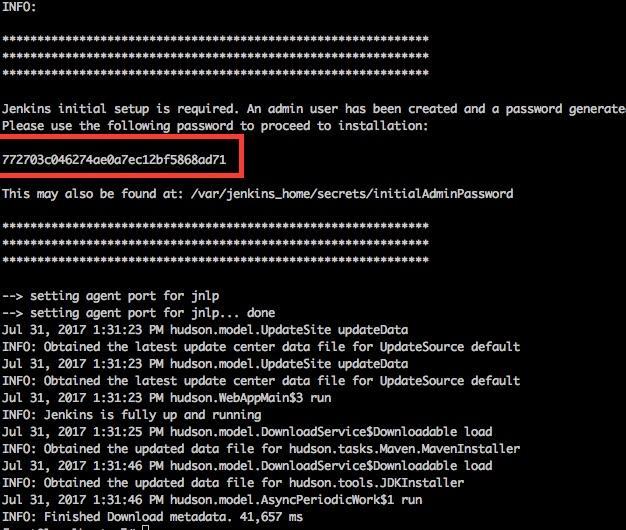
-d:后台运行

2）、通过端口8080访问Jenkins管理界面

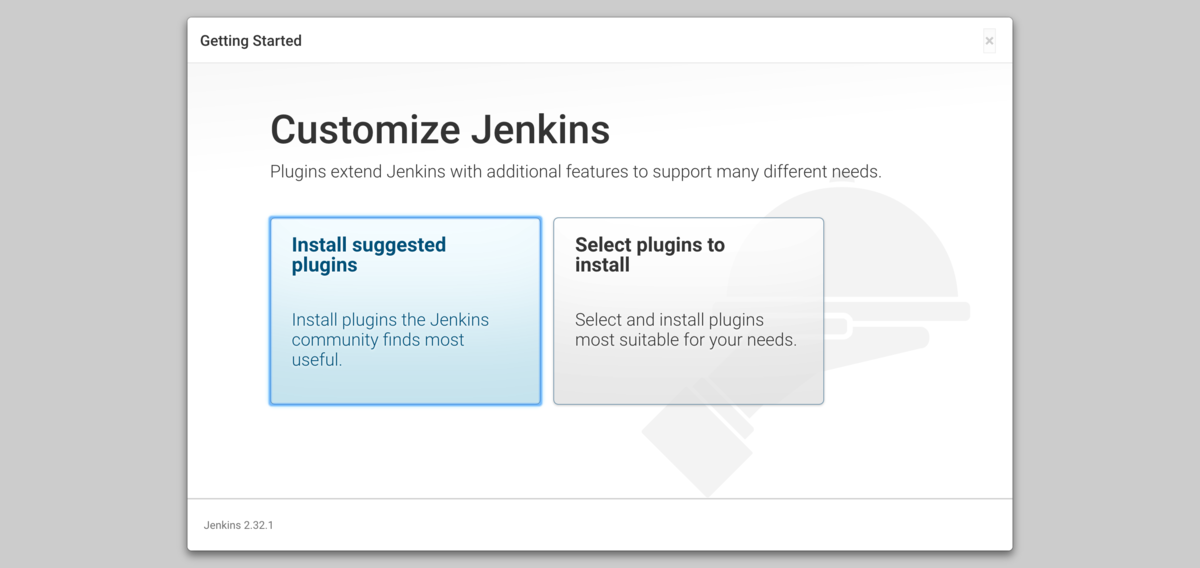


此处密码可以通过如下命令查看

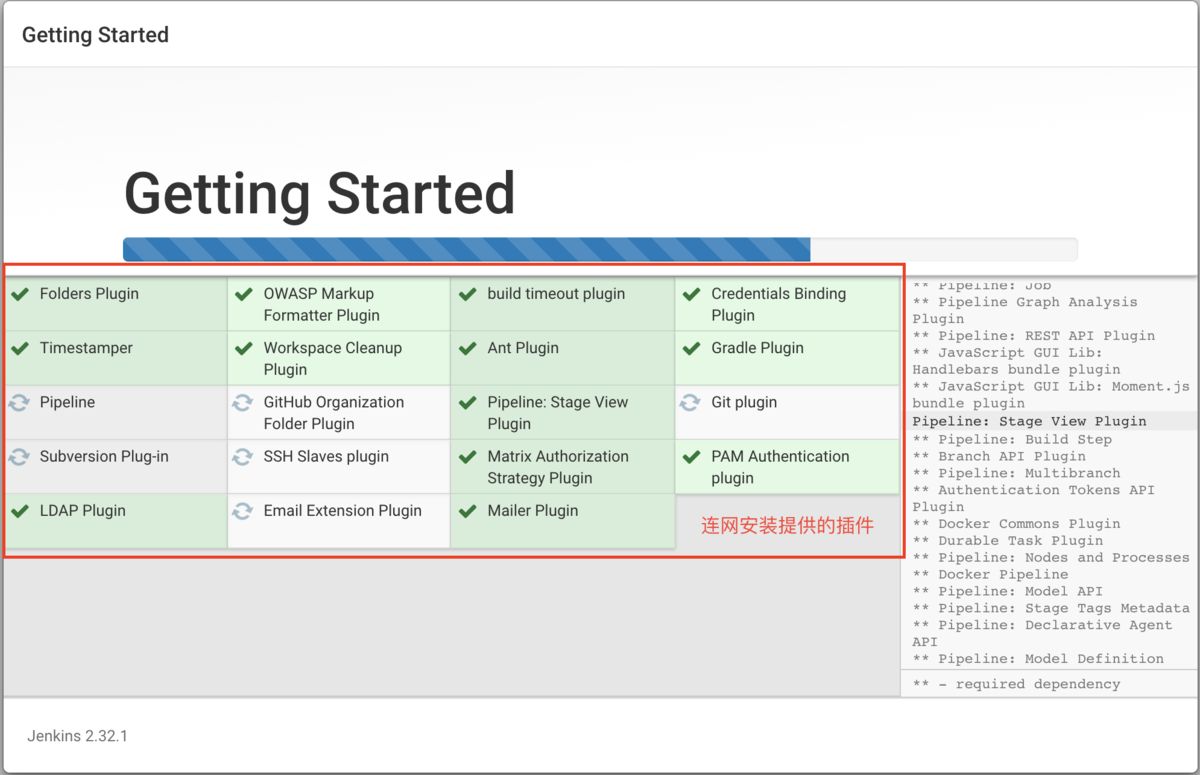
docker logs 容器id



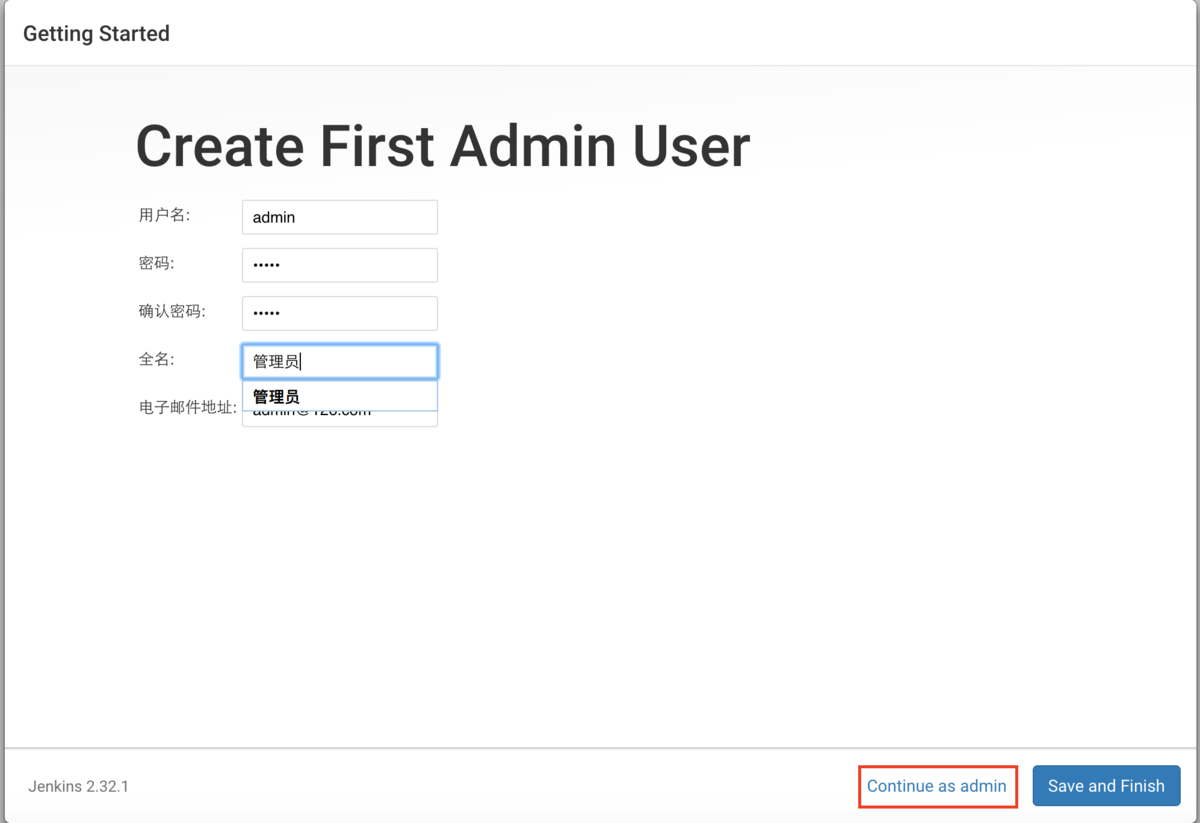
选择安装插件方式，这里我是默认第一个，此处安装插件很多时候都不会成功，首先可能与网络有关，其次就是插件地址有关。后面在插件安装条目中我会描述遇到插件安装不了，如何去解决。



进入插件安装界面，连网等待插件安装



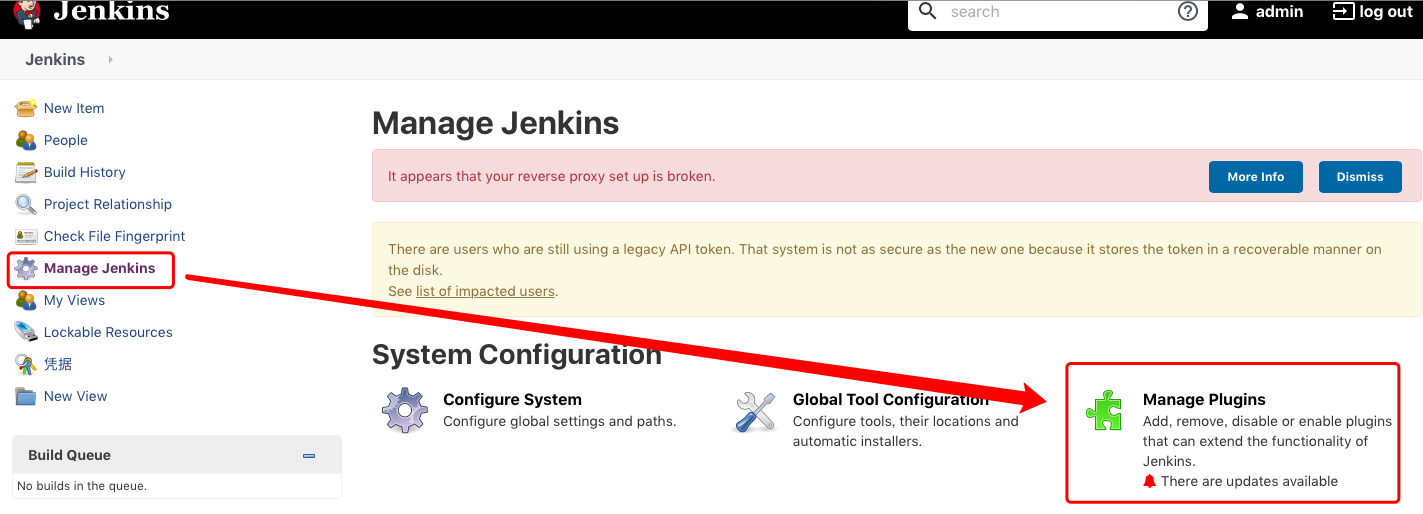
安装完插件后，进入创建管理员界面

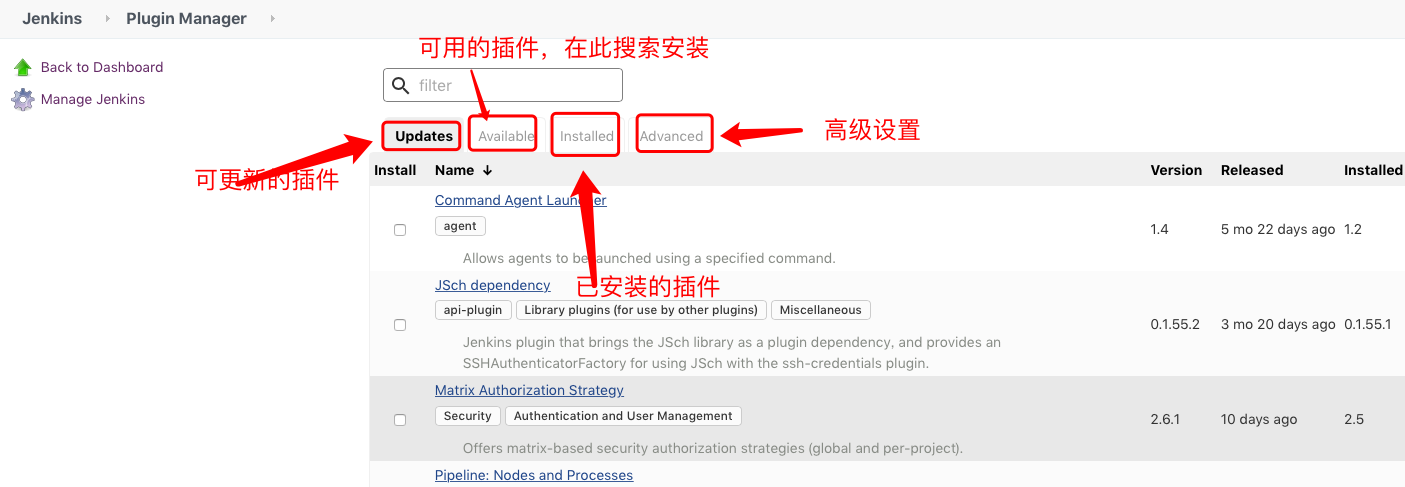


此处我设置的Jenkins账户密码是admin/admin，后面我会单独说明

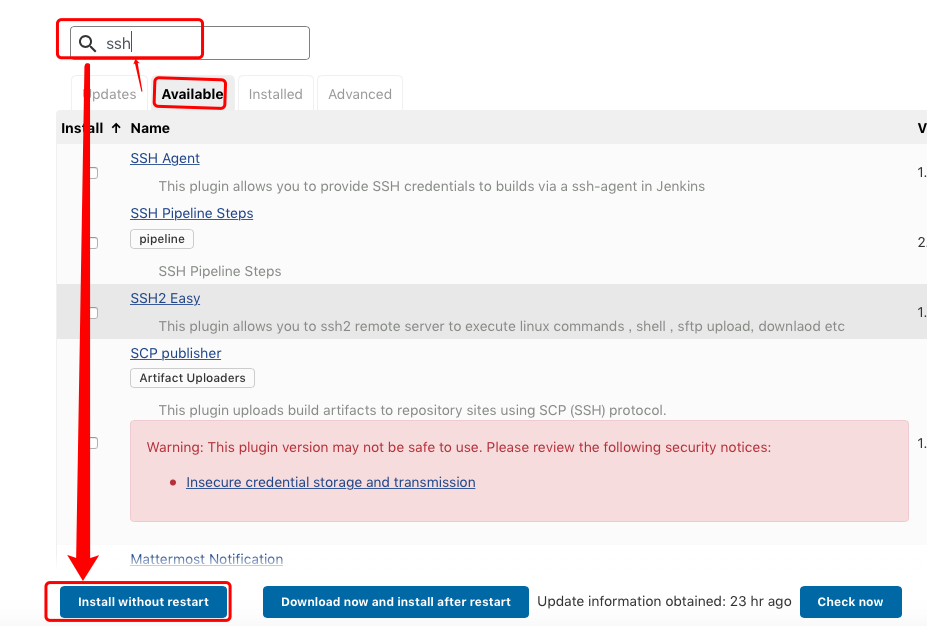
1. **插件安装**

选择Jenkins设置->插件管理





点击Available按钮



在搜索框输入需要安装的插件，在结果当中选中，点击install without restart按钮，进行安装

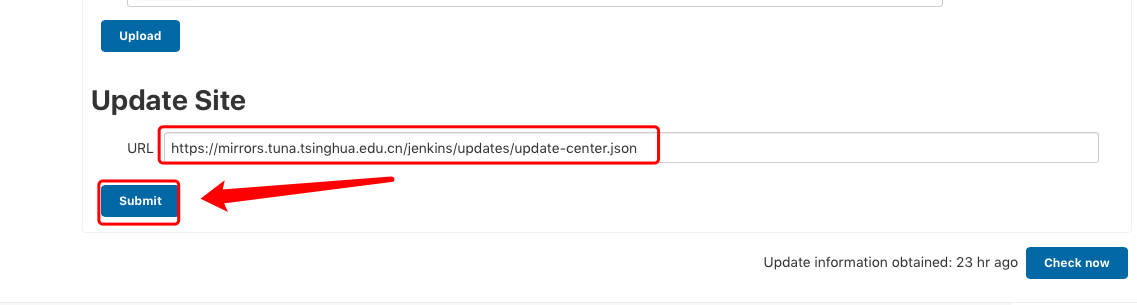
需要安装的插件前面有提到

很多时候安装插件都会报错，此时有几个办法可以解决

1）、再次点击安装按钮进行安装重试

2）、更换插件下载地址

点击Advanced按钮，进入高级设置，找到如下位置



更改插件下载地址为如下任意一个

|  |
| --- |
| https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/update-center.json  http://mirror.xmission.com/jenkins/updates/update-center.json  http://mirrors.shu.edu.cn/jenkins/updates/current/update-center.json  https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/update-center.json |

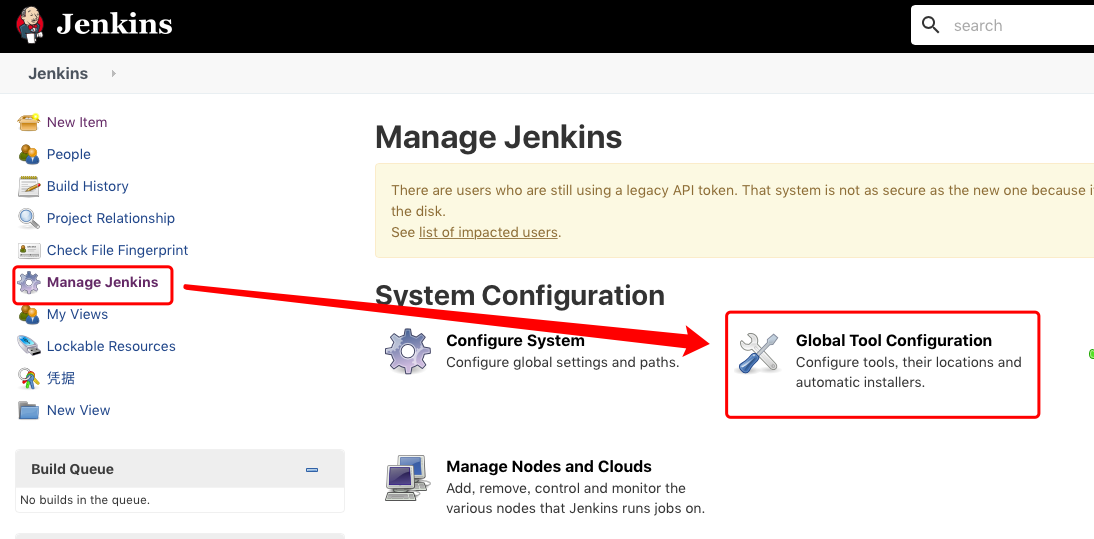
点击submit按钮，最好是重启Jenkins

3）、切换网络

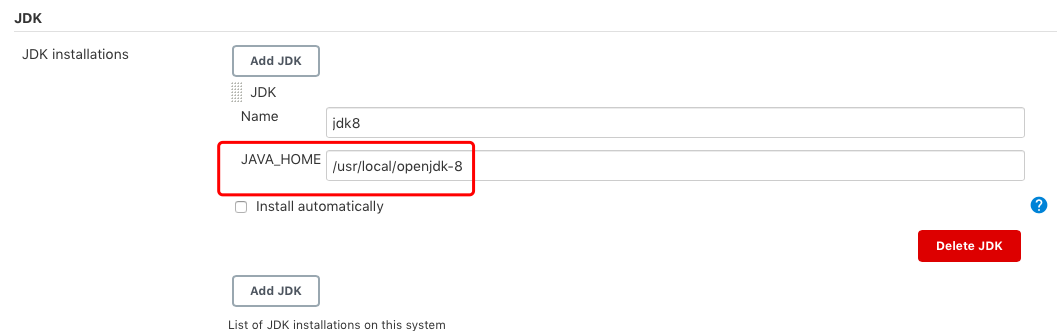
4）、重启

1. **全局配置**

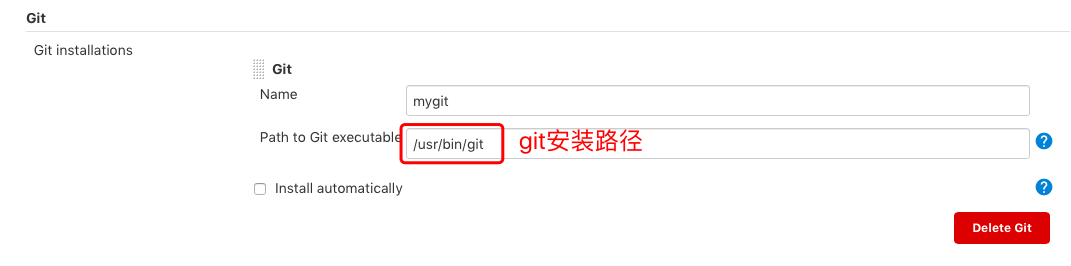
点击Manage Jenkins->Global Tool Configuration



配置jdk



配置git ，可以通过which git 或 whereis git命令查看安装路径

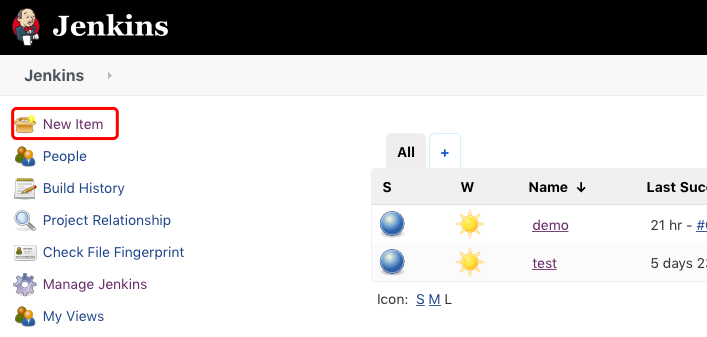


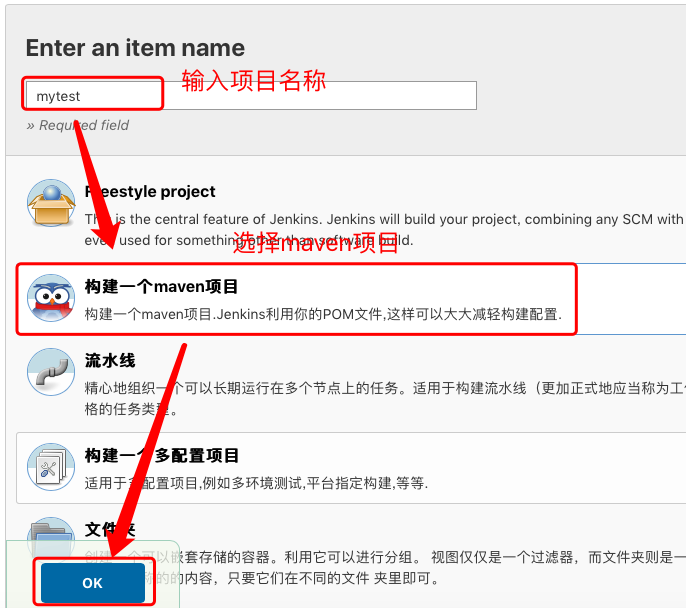
配置maven



1. **创建项目**

点击 New Item按钮



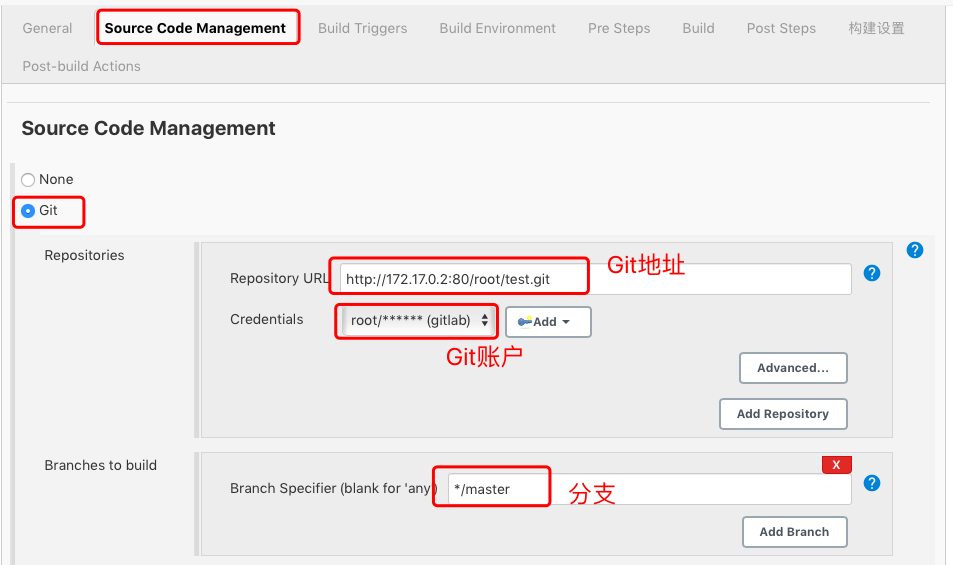


点击OK按钮后会进入到工程的相关配置中

1. **gitlab配置**

gitlab分为http方式配置和ssh方式

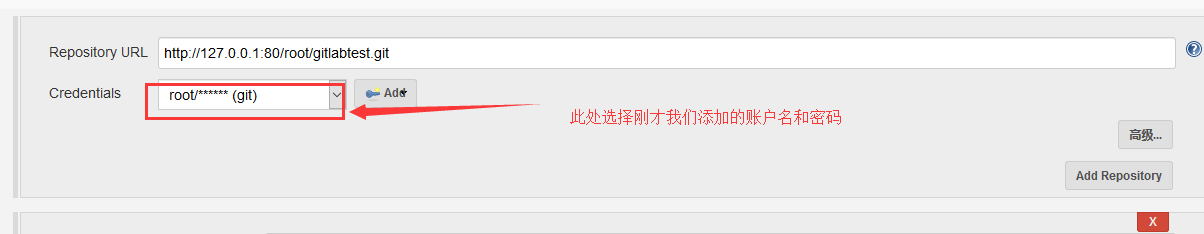
1）、http方式



此处gitlab地址用的是ip，因为我没有配置域名，可以通过设置/etc/hosts来设置域名

配置git账户

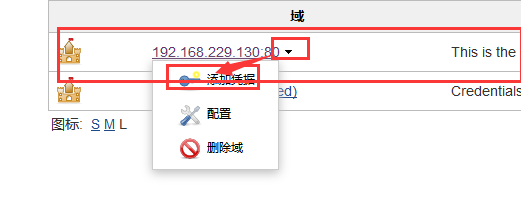


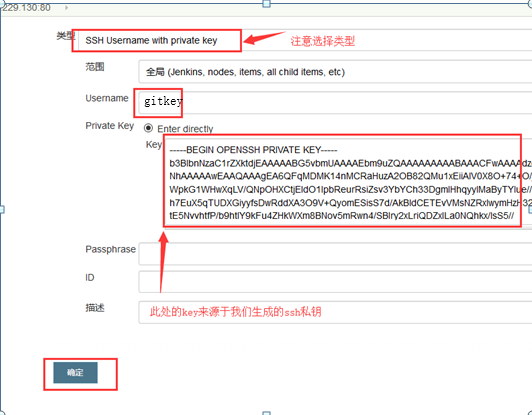


2）、ssh方式









Username此处可以随便取

此处多说一点private key,来自Jenkins服务器执行ssh命令

ssh-keygen -o -t rsa -C "982452937@qq.com" -b 4096

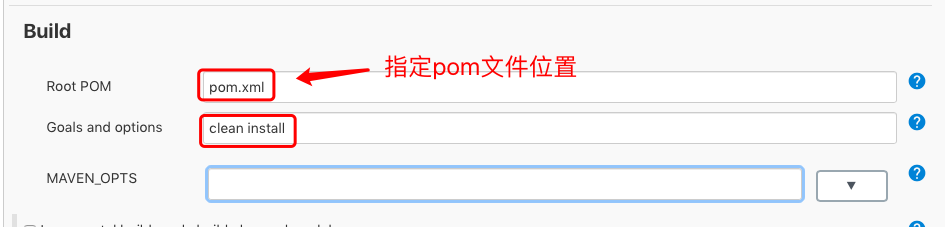
这个命令在Ubuntu系统下会在/root/.ssh/文件夹下生成两个文件

id\_rsa：私钥

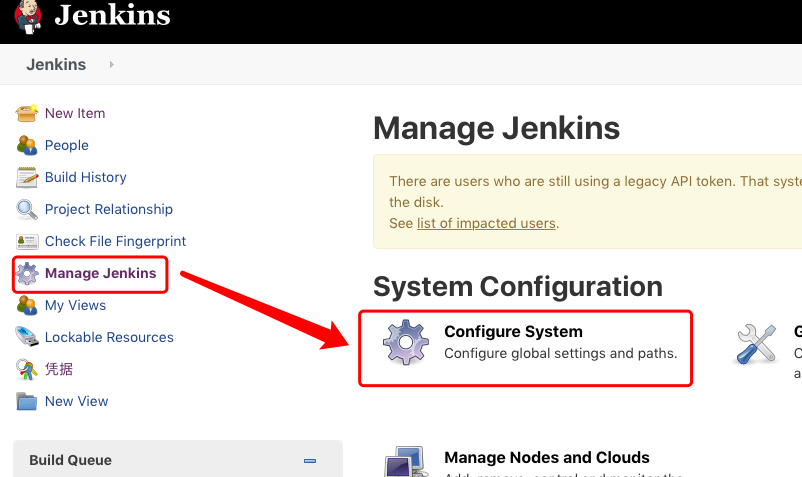
id\_rsa.pub：公钥

上面添加凭据的时候，是把私钥复制到凭据里，这里需要注意

1. **maven配置**



1. **ssh配置**



找到Publish over SSH，如果没有，则需要安装该插件，安装插件的方法上面有提到



**参数说明**：

Passphrase：SSH的密码，创建ssh key时指定，本文创建ssh key时默认为空。故，此处为空，不填写。

Path to key：SSH私钥的文件路径，私钥文件的路径，可以是绝对路径，也可以是相对$JENKINS\_HOME的相对路径

Key：私钥，私钥导出后的文本内容

如果“Key”和“Path to key”都设置，则“Key”的优先级较高



关于key

1）、私钥需要在Jenkins服务器A生成

|  |
| --- |
| ssh-keygen -t rsa -P '' |

2）、使用cat id\_rsa.pub（公钥） 查看内容，并复制此条命令。

3）、在部署服务器B上，进入目录：/root/.ssh，并执行如下命令：

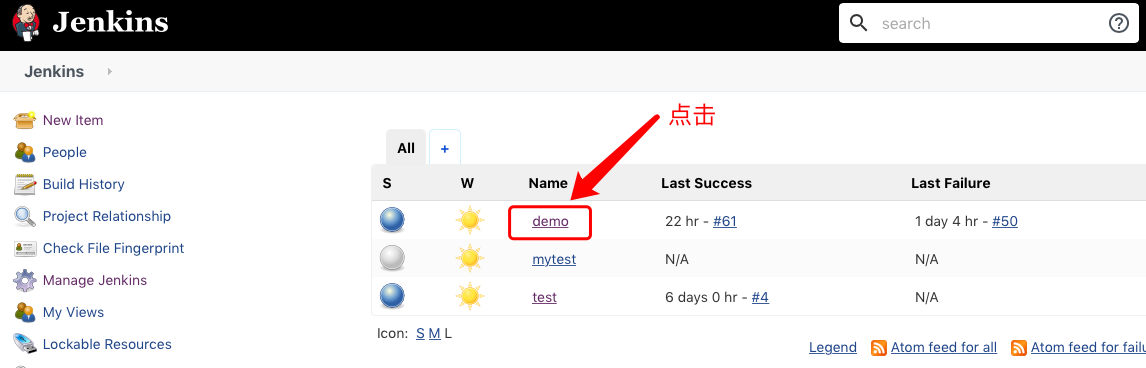
|  |
| --- |
| echo 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC+9kEHx54UKNPl6oc0lU5XCwb1kwCtRC0++KKQtHly3jbJATUoagcAL2IUnnH2tEnZq0p0gvDpOJSAiug4kS4hNremsUmQJ+ew9ib2/o+dgLCBufOxfu+YnbX+3xnGuPfqkwKY6HW9hOUBAmLPOyjhwpcvmjd92W2stKwqn1w1Jd0ikpbDfTili2c8Mie+VJzLX/ywowiRpo6OM3JBd0cINQcLmBA6IQRXj6WVABNGrtrYPYdQ0A67Vgdwu3CywJleZxRFp1jqQzClR9dtia5MlQXcih1HcehtNgnfmQI2xeHWlK4+kl1CvUDCOUigzSomlFzEFTa5p5vKzDbNIrH1 root@b958c6715350' >> authorized\_keys |

验证如上是否配置成功，在Jenkins服务器A上输入命令

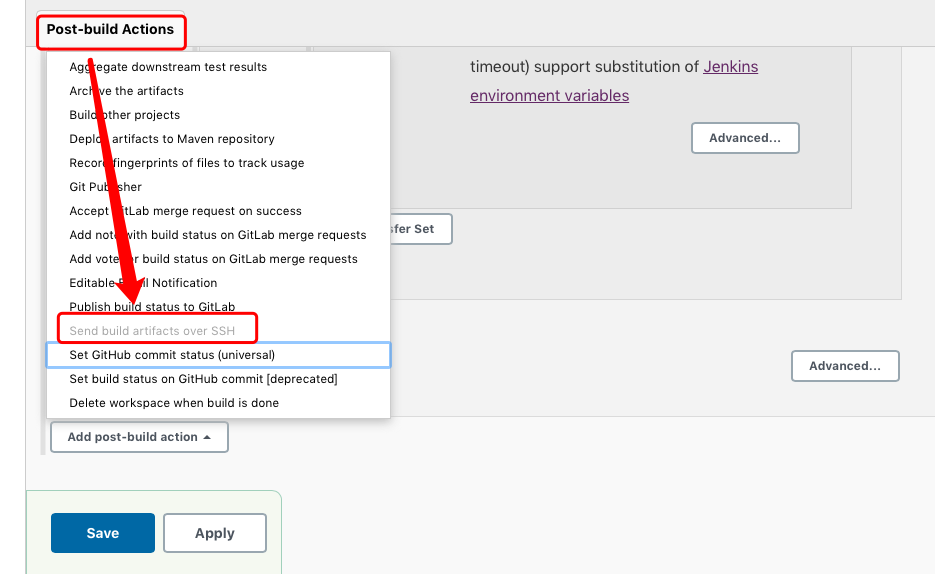
ssh [root@172.17.0.1](mailto:root@172.17.0.1) ，可以免密登陆则配置成功

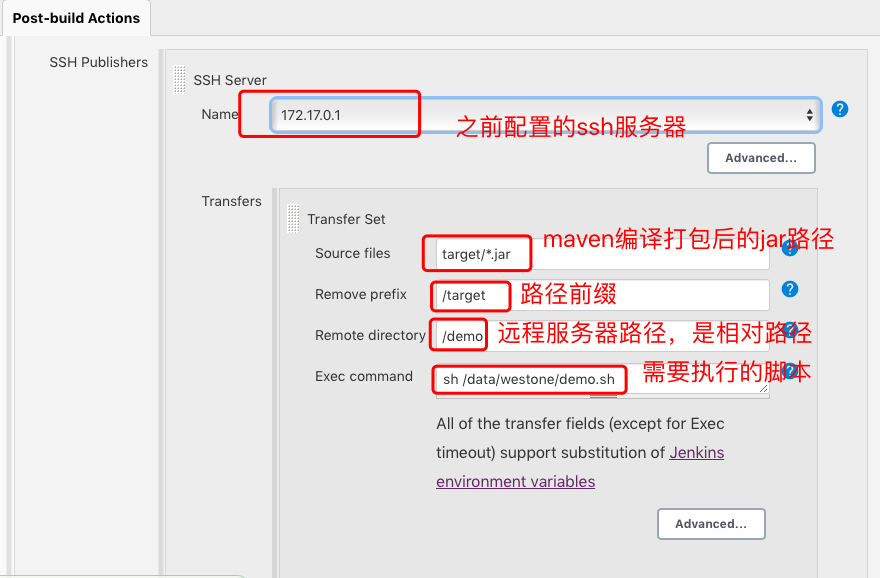
172.17.0.1：是部署服务器IP，root是用户名，此处最好使用root用户，要不然会涉及很多权限问题。

如上配置成功后，点击具体项目









参数说明

Name：下拉选择一个 系统设置已配置的名字。

Transfer Set Source files：需要上传的文件（注意：相对于工作空间根目录。）

Remove prefix：Source files”配置的路径中要移除的前缀

Remote directory：远程目录，此目录是相对于“SSH Server”中的“Remote directory”的，如果不存在将会自动创建。

Exec command：通过ssh连接远程服务器后，在远程服务器上执行的脚本。

1. **dockerfile编写**

dockerfile目的是为了将maven打包好的jar，通过docker再打包成docker镜像.

编写Dockerfile文件，内容如下

|  |
| --- |
| FROM openjdk:8-jdk-alpine  #FROM hub.c.163.com/dwyane/openjdk:8  ADD demo-0.0.1-SNAPSHOT.jar demo.jar  ENTRYPOINT ["java","-jar","/demo.jar","-c"] |

将Dockerfile文件放到部署服务器/data/westone/demo目录下，这个目录不能随意放置，是之前在Jenkins上配置好的目录，要不然docker会找不到文件。

1. **脚本编写**

该脚本是Jenkins远程ssh时，需要执行的脚本，用于将jar包打包成docker镜像，然后再运行镜像，启动我们需要的工程服务。

|  |
| --- |
| #!/bin/bash  # Variables  DOCKERFILE\_HOME='/data/westone/demo/'  REPOSITORIES='demo'  # Copy the newest war to docker-file directory.  #\cp -f ${JENKINS\_WAR\_HOME}/godseye-container-wisedu.war ${DOCKERFILE\_HOME}/godseye.war  IMAGE\_ID=`sudo docker images | grep ${REPOSITORIES} | awk '{print $3}'`  CONTAINER\_ID=`docker ps | grep "demo" | awk '{print $1}'`  if [ -n "$CONTAINER\_ID" ]; then      docker stop $CONTAINER\_ID      docker rm $CONTAINER\_ID  else #如果容器启动时失败了，就需要docker ps -a才能找到那个容器      CONTAINER\_ID=`docker ps -a | grep "demo" | awk '{print $1}'`      if [ -n "$CONTAINER\_ID" ]; then  # 如果是第一次在这台机器上拉取运行容器，那么docker ps -a也是找不到这个容器的          docker rm $CONTAINER\_ID      fi  fi  if [ -n "${IMAGE\_ID}" ];then      sudo docker rmi ${IMAGE\_ID}  fi  # Build image.  cd ${DOCKERFILE\_HOME}  TAG=`date +%Y%m%d-%H%M%S`  sudo docker build -t ${REPOSITORIES}:${TAG} .  # Run.  docker run -d --name demo -p 18080:18080 ${REPOSITORIES}:${TAG} |

如上的脚本主要任务是将之前已有的docker容器和镜像先删除，然后再生成新的镜像，并运行新的镜像，镜像名称是固定名称+时间。

1. **功能**
2. **Multi-configuration多配置项目**

网址：<https://blog.csdn.net/qq_35081227/article/details/81189116>

1. **分布式构建**

网址：<https://blog.csdn.net/u012060033/article/details/87259981>

1. **代码质量检测**

<https://www.cnblogs.com/gcgc/p/10829948.html>

1. **参数化构建**

<https://www.mgchen.com/203.html>

1. **aa**