

Author: Eric

Version: 9.0.1

- 一、引言
 - 1.1 环境不一致
 - 1.2 隔离性
 - 1.3 弹性伸缩
 - 1.4 学习成本
- 二、Docker介绍
 - 2.1 Docker的由来
 - 2.2 Docker的思想
- 三、Docker的安装
 - 3.1 下载Docker依赖的环境
 - 3.2 指定Docker镜像源
 - 3.3 安装Docker
 - 3.4 启动Docker并测试
- 四、Docker的中央仓库【重点】
- 五、镜像的操作【重点】
 - 5.1 拉取镜像
 - 5.2 查看本地全部镜像
 - 5.3 删除本地镜像
 - 5.4 镜像的导入导出
- 六、容器操作【重点】
 - 6.1 运行容器
 - 6.2 查看正在运行的容器
 - 6.3 查看容器日志
 - 6.4 进入容器内部
 - 6.5 复制内容到容器
 - 6.6 重启&启动&停止&删除容器
- 七、Docker应用
 - 7.1 Docker安装Tomcat
 - 7.2 Docker安装MySQL
 - 7.3 部署SSM工程
- 八、数据卷【重点】
 - 8.1 创建数据卷
 - 8.2 查看数据卷详情
 - 8.3 查看全部数据卷
 - 8.4 删除数据卷
 - 8.5 容器映射数据卷
- 九、Dockerfile自定义镜像【重点】
 - 9.1 Dockerfile
 - 9.2 通过Dockerfile制作镜像
- 十、Docker-Compose【重点】
 - 10.1 下载并安装Docker-Compose
 - 10.1.1 下载Docker-Compose
 - 10.1.2 设置权限
 - 10.1.3 配置环境变量
 - 10.1.4 测试
 - 10.2 Docker-Compose管理MySQL和Tomcat容器
 - 10.3 使用docker-compose命令管理容器
 - 10.4 docker-compose配合Dockerfile使用
 - 10.4.1 docker-compose文件
 - 10.4.2 Dockerfile文件
 - 10.4.3 运行
- 十一、Docker CI、CD
 - 11.1 CI、CD引言
 - 11.2 CI介绍
 - 11.3 搭建Gitlab服务器
 - 11.3.1 准备工作
 - 11.3.2 修改ssh的22端口
 - 11.3.3 编写docker-compose.yml
 - 11.4 搭建GitlabRunner
 - 11.5 整合项目入门测试
 - 11.5.1 创建项目

11.5.2 编写.gitlab-ci.yml
11.5.3 将maven工程推送到gitlab中
11.5.4 查看效果
11.6 完善项目配置
11.6.1 创建Dockerfile
11.6.2 创建docker-compose.yml
11.6.3 修改.gitlab-ci.yml
11.6.4 测试
11.7 CD介绍
11.8 安装Jenkins
11.8.1 编写docker-compose.yml
11.8.2 运行并访问Jenkins
11.9 配置Jenkins的目标服务器
11.9.1 点击左侧的系统设置
11.9.2 选中中间区域的系统设置
11.9.3 搜索Publish over SSH
11.9.4 点击上图新增
11.10 配置GitLab免密码登录
11.10.1 登录Jenkins容器内部
11.10.2 输入生成SSH秘钥命令
11.10.3 将秘钥复制到GitLab的SSH中
11.11 配置JDK和Maven
11.11.1 复制软件到data目录下
11.11.2 在监控界面中配置JDK和Maven
11.11.3 手动拉取gitlab项目
11.12 创建maven任务
11.12.1 创建maven工程，推送到GitLab中
11.12.2 Jenkins的监控页面中创建maven任务
11.12.3 执行maven任务
11.12.4 最终效果
11.13 实现持续交付持续部署
11.13.1 安装Persistent Parameter的插件
11.13.2 重新指定构建项目的方式
11.13.3 构建项目成功后，需要将内容发布到目标服务器
11.13.4 添加程序代码
11.13.5 测试

一、引言

1.1 环境不一致

我本地运行没问题啊：由于环境不一致，导致相同的程序，运行结果却不一样。

1.2 隔离性

哪个哥们又写死循环了，怎么这么卡：在多用户的操作系统下，会因为其他用户的操作失误影响到你自己编些的程序。

1.3 弹性伸缩

淘宝在双11的时候，用户量暴增：需要很多很多的运维人员去增加部署的服务器，运维成本过高的问题。

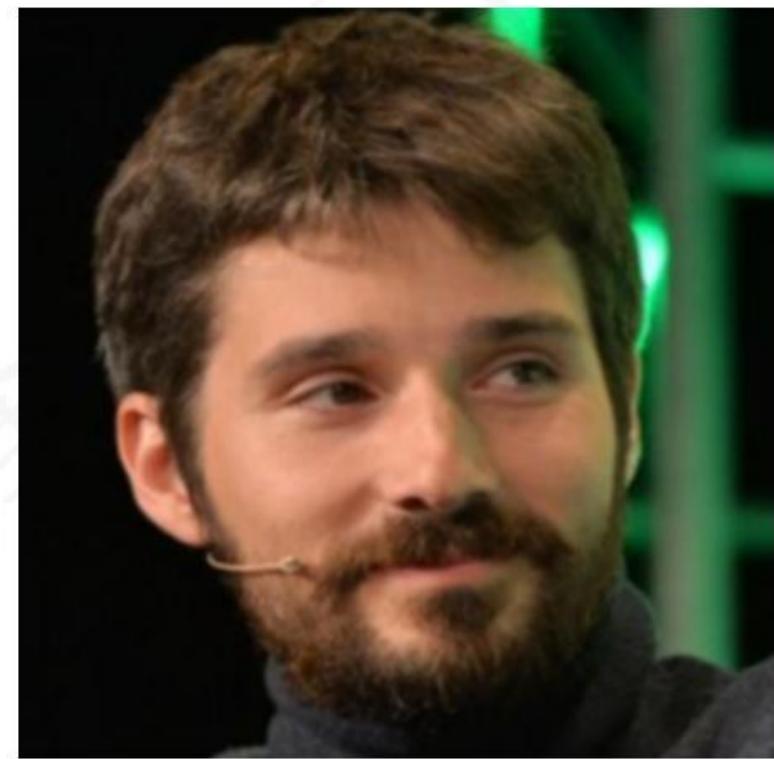
1.4 学习成本

学习一门技术，得先安装啊：学习每一门技术都要先安装响应的软件，但是还有他所依赖的各种环境，安装软件成本快高过学习成本啦。

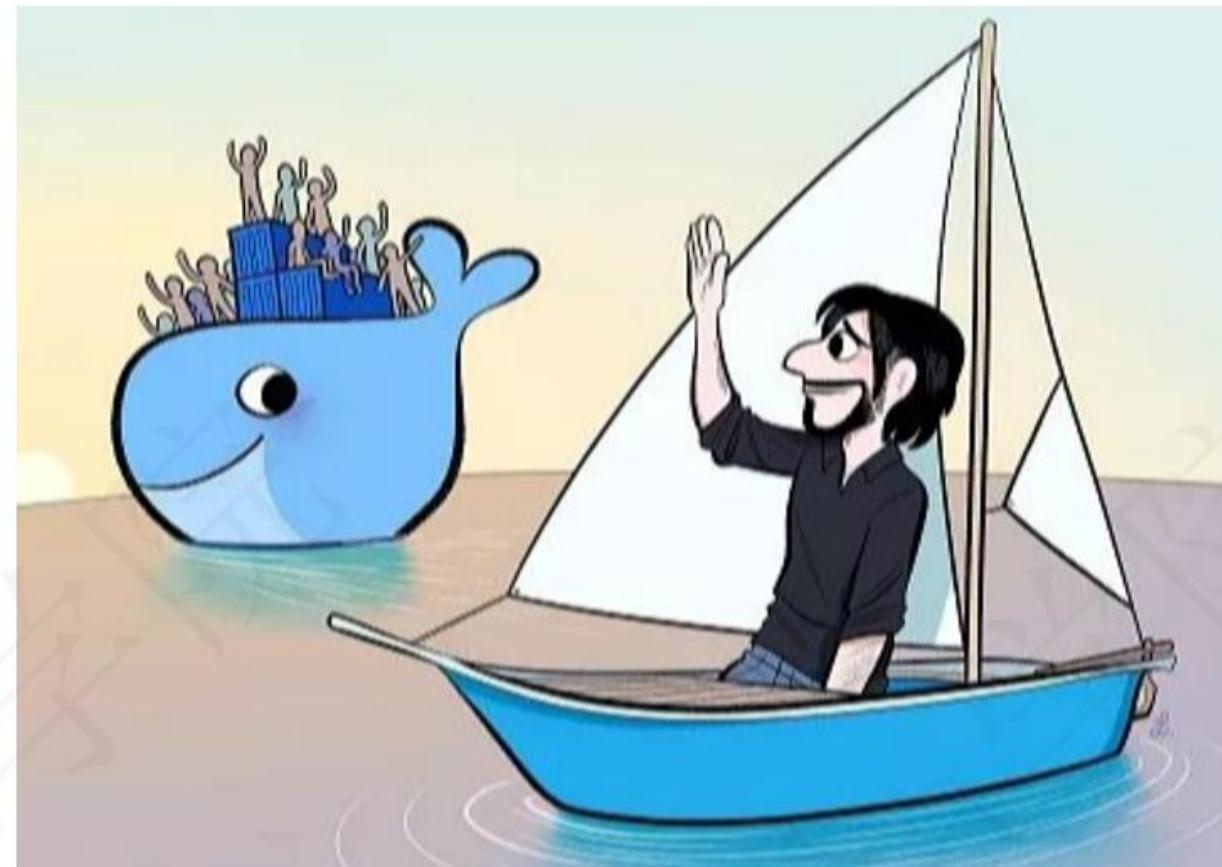
二、Docker介绍

2.1 Docker的由来

一帮年轻人创业，创办了一家公司，2010年的专门做PaaS平台。但是到了2013年的时候，像亚马逊，微软，Google都开始做PaaS平台。到了2013年，公司资金链断裂，不得不倒闭，于是将公司内的核心技术对外开源，核心技术就是Docker。由于开源了Docker，到了2014年的时候，得到了C轮的融资4000W，2015年的时候，得到了D轮的融资9500W。于是公司开始全神贯注的维护Docker。



Docker的作者已经离开了维护Docker的团队



2.2 Docker的思想

- 集装箱：会将所有需要的内容放到不同的集装箱中，谁需要这些环境就直接拿到这个集装箱就可以了。
- 标准化：
 - 运输的标准化：Docker有一个码头，所有上传的集装箱都放在了这个码头上，当谁需要某一个环境，就直接指派大海去搬运这个集装箱就可以了。
 - 命令的标准化：Docker提供了一些列的命令，帮助我们去获取集装箱等等操作。
 - 提供了REST的API：衍生出了很多的图形化界面，Rancher。
- 隔离性：Docker在运行集装箱内的内容时，会在Linux的内核中，单独的开辟一片空间，这片空间不会影响到其他程序。
- 中央仓库|注册中心：超级码头，上面放的就是集装箱
- 镜像：就是集装箱
- 容器：运行起来的镜像

三、Docker的安装

3.1 下载Docker依赖的环境

想安装Docker，需要先将依赖的环境全部下载，就像Maven依赖JDK一样

```
yum -y install yum-utils device-mapper-persistent-data lvm2
```

3.2 指定Docker镜像源

默认下载Docker回去国外服务器下载，速度较慢，我们可以设置为阿里云镜像源，速度更快

```
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

3.3 安装Docker

依然采用yum的方式安装

```
yum makacache fast  
yum -y install docker-ce
```

3.4 启动Docker并测试

安装成功后，需要手动启动，设置为开机自启，并测试一下Docker

```
# 启动Docker服务  
systemctl start docker  
# 设置开机自动启动  
systemctl enable docker  
# 测试  
docker run hello-world
```

四、Docker的中央仓库【重点】

- Docker官方的中央仓库：这个仓库是镜像最全的，但是下载速度较慢。
<https://hub.docker.com/>
- 国内的镜像网站：网易蜂巢，daoCloud等，下载速度快，但是镜像相对不全。
<https://c.163yun.com/hub#/home>
<http://hub.daocloud.io/>（推荐使用）
- 在公司内部会采用私服的方式拉取镜像，需要添加配置，如下.....

```
# 需要创建/etc/docker/daemon.json，并添加如下内容  
{  
    "registry-mirrors": [ "https://registry.docker-cn.com" ],  
    "insecure-registries": [ "ip:port" ]  
}  
# 重启两个服务  
systemctl daemon-reload  
systemctl restart docker
```

五、镜像的操作【重点】

5.1 拉取镜像

从中央仓库拉取镜像到本地

```
docker pull 镜像名称[:tag]  
# 举个栗子: docker pull daocloud.io/library/tomcat:8.5.15-jre8
```

5.2 查看本地全部镜像

查看本地已经安装过的镜像信息，包含标识，名称，版本，更新时间，大小

```
docker images
```

5.3 删除本地镜像

镜像会占用磁盘空间，可以直接手动删除，表示通过查看获取

```
docker rmi 镜像的标识
```

5.4 镜像的导入导出

如果因为网络原因可以通过硬盘的方式传输镜像，虽然不规范，但是有效，但是这种方式导出的镜像名称和版本都是null，需要手动修改

```
# 将本地的镜像导出  
docker save -o 导出的路径 镜像id  
# 加载本地的镜像文件  
docker load -i 镜像文件  
# 修改镜像名称  
docker tag 镜像id 新镜像名称:版本
```

六、容器操作【重点】

6.1 运行容器

运行容器需要制定具体镜像，如果镜像不存在，会直接下载

```
# 简单操作  
docker run 镜像的标识|镜像名称[:tag]  
  
# 常用的参数  
docker run -d -p 宿主机端口:容器端口 --name 容器名称 镜像的标识|镜像名称[:tag]  
# -d: 代表后台运行容器  
# -p 宿主机端口:容器端口:为了映射当前Linux的端口和容器的端口  
# --name 容器名称: 指定容器的名称
```

6.2 查看正在运行的容器

查看全部正在运行的容器信息

```
docker ps [-qa]  
# -a: 查看全部的容器，包括没有运行  
# -q: 只查看容器的标识
```

6.3 查看容器日志

查看容器日志，以查看容器运行的信息

```
docker logs -f 容器id  
# -f: 可以滚动查看日志的最后几行
```

6.4 进入容器内部

可以进入容器内部进行操作

```
docker exec -it 容器id bash
```

6.5 复制内容到容器

将宿主机的文件复制到容器内部的指定目录

```
docker cp 文件名称 容器id:容器内部路径
```

6.6 重启&启动&停止&删除容器

容器的启动，停止，删除等操作，后续经常会使用到

```
# 重新启动容器  
docker restart 容器id  
  
# 启动停止运行的容器  
docker start 容器id  
  
# 停止指定的容器（删除容器前，需要先停止容器）  
docker stop 容器id  
# 停止全部容器  
docker stop $(docker ps -qa)  
  
# 删除指定容器  
docker rm 容器id  
# 删除全部容器  
docker rm $(docker ps -qa)
```

七、Docker应用

7.1 Docker安装Tomcat

运行Tomcat容器，为部署SSM工程做准备

```
docker run -d -p 8080:8080 --name tomcat daocloud.io/library/tomcat:8.5.15-jre8
```

7.2 Docker安装MySQL

运行MySQL容器，为部署SSM工程做准备

```
docker run -d -p 3306:3306 --name mysql -e MYSQL_ROOT_PASSWORD=root daocloud.io/library/mysql:5.7.4
```

7.3 部署SSM工程

- 修改SSM工程环境，设置为Linux中Docker容器的信息
- 通过Maven的package重新打成war包
- 将Windows下的war包复制到Linux中
- 通过docker命令将宿主机的war包复制到容器内部
- 测试访问SSM工程

八、数据卷【重点】

为了部署SSM的工程，需要使用到cp的命令将宿主机内的ssm.war文件复制到容器内部。

数据卷：将宿主机的一个目录映射到容器的一个目录中。

可以在宿主机中操作目录中的内容，那么容器内部映射的文件，也会跟着一起改变。

8.1 创建数据卷

创建数据卷之后，默认会存放在一个目录下 /var/lib/docker/volumes/数据卷名称/_data

```
docker volume create 数据卷名称
```

8.2 查看数据卷详情

查看数据卷的详细信息，可以查询到存放路径，创建时间等等

```
docker volume inspect 数据卷名称
```

8.3 查看全部数据卷

查看全部数据卷信息

```
docker volume ls
```

8.4 删除数据卷

删除指定数据卷

```
docker volume rm 数据卷名称
```

8.5 容器映射数据卷

映射有两种方式：

- 通过数据卷名称映射，如果数据卷不存在。Docker会帮你自动创建，会将容器内部自带的文件，存储在默认的存放路径中。
- 通过路径映射数据卷，直接指定一个路径作为数据卷的存放位置。但是这个路径下是空的。

```
# 通过数据卷名称映射  
docker run -v 数据卷名称:容器内部的路径 镜像id  
# 通过路径映射数据卷  
docker run -v 路径:容器内部的路径 镜像id
```

九、Dockerfile自定义镜像【重点】

我们可以从中央仓库下载一个镜像，也可以自己手动去制作一个镜像，需要通过Dockerfile去指定自定义镜像的信息

9.1 Dockerfile

创建自定义镜像就需要创建一个Dockerfile，如下为Dockerfile的语言

```
from: 指定当前自定义镜像依赖的环境  
copy: 将相对路径下的内容复制到自定义镜像中  
workdir: 声明镜像的默认工作目录  
run: 执行的命令，可以编写多个  
cmd: 需要执行的命令（在workdir下执行的，cmd可以写多个，只以最后一个为准）
```

```
# 举个例子，制作SSM容器镜像，而且ssm.war要放在Dockerfile的同级目录下  
from daocloud.io/library/tomcat:8.5.15-jre8  
copy ssm.war /usr/local/tomcat/webapps
```

9.2 通过Dockerfile制作镜像

编写完Dockerfile后需要通过命令将其制作成镜像，并且要在Dockerfile的当前目录下，之后即可在镜像中查看到指定的镜像信息，注意最后的`.`

```
docker build -t 镜像名称[:tag] .
```

十、Docker-Compose【重点】

之前运行一个镜像，需要添加大量的参数，可以通过Docker-Compose编写这些参数。而且Docker-Compose可以帮助我们批量的管理容器。这些信息只需要通过一个`docker-compose.yml`文件去维护即可。

10.1 下载并安装Docker-Compose

10.1.1 下载Docker-Compose

去github官网搜索`docker-compose`，下载1.24.1版本的Docker-Compose

下载路径：https://github.com/docker/compose/releases/download/1.24.1/docker-compose-Linux-x86_64

10.1.2 设置权限

需要将`DockerCompose`文件的名称修改一下，给予`DockerCompose`文件一个可执行的权限

```
mv docker-compose-Linux-x86_64 docker-compose  
chmod 777 docker-compose
```

10.1.3 配置环境变量

方便后期操作，配置一个环境变量

将`docker-compose`文件移动到了`/usr/local/bin`，修改了`/etc/profile`文件，给`/usr/local/bin`配置到了`PATH`中

```
mv docker-compose /usr/local/bin  
  
vi /etc/profile  
# 添加内容: export PATH=$JAVA_HOME:/usr/local/bin:$PATH  
  
source /etc/profile
```

10.1.4 测试

在任意目录下输入`docker-compose`

测试效果

```
[root@localhost ~]# docker-compose
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE           Specify an alternate compose file
                            (default: docker-compose.yml)
  -p, --project-name NAME   Specify an alternate project name
```

10.2 Docker-Compose管理MySQL和Tomcat容器

yml文件以key: value方式来指定配置信息
多个配置信息以换行+缩进的方式来区分
在docker-compose.yml文件中，不要使用制表符

```
version: '3.1'
services:
  mysql:          # 服务的名称
    restart: always # 代表只要docker启动，那么这个容器就跟着一起启动
    image: daocloud.io/library/mysql:5.7.4 # 指定镜像路径
    container_name: mysql # 指定容器名称
    ports:
      - 3306:3306 # 指定端口号的映射
    environment:
      MYSQL_ROOT_PASSWORD: root # 指定MySQL的ROOT用户登录密码
      TZ: Asia/Shanghai # 指定时区
    volumes:
      - /opt/docker_mysql_tomcat/mysql_data:/var/lib/mysql # 映射数据卷
  tomcat:
    restart: always
    image: daocloud.io/library/tomcat:8.5.15-jre8
    container_name: tomcat
    ports:
      - 8080:8080
    environment:
      TZ: Asia/Shanghai
    volumes:
      - /opt/docker_mysql_tomcat/tomcat_webapps:/usr/local/tomcat/webapps
      - /opt/docker_mysql_tomcat/tomcat_logs:/usr/local/tomcat/logs
```

10.3 使用docker-compose命令管理容器

在使用docker-compose的命令时， 默认会在当前目录下找docker-compose.yml文件

```
# 1. 基于docker-compose.yml启动管理的容器
docker-compose up -d

# 2. 关闭并删除容器
docker-compose down

# 3. 开启|关闭|重启已经存在的由docker-compose维护的容器
docker-compose start|stop|restart

# 4. 查看由docker-compose管理的容器
docker-compose ps

# 5. 查看日志
docker-compose logs -f
```

10.4 docker-compose配合Dockerfile使用

使用docker-compose.yml文件以及Dockerfile文件在生成自定义镜像的同时启动当前镜像，并且由docker-compose去管理容器

10.4.1 docker-compose文件

编写docker-compose.yml文件

```
# yml文件
version: '3.1'
services:
  ssm:
    restart: always
```

```
build:          # 构建自定义镜像
  context: ...   # 指定dockerfile文件的所在路径
  dockerfile: Dockerfile  # 指定Dockerfile文件名称
image: ssm:1.0.1
container_name: ssm
ports:
  - 8081:8080
environment:
  TZ: Asia/Shanghai
```

10.4.2 Dockerfile文件

编写Dockerfile文件

```
from daocloud.io/library/tomcat:8.5.15-jre8
copy ssm.war /usr/local/tomcat/webapps
```

10.4.3 运行

测试效果

```
# 可以直接启动基于docker-compose.yml以及Dockerfile文件构建的自定义镜像
docker-compose up -d
# 如果自定义镜像不存在，会帮助我们构建出自定义镜像，如果自定义镜像已经存在，会直接运行这个自定义镜像
# 重新构建的话。
# 重新构建自定义镜像
docker-compose build
# 运行当前内容，并重新构建
docker-compose up -d --build
```

十一. Docker CI、CD

11.1 CI、CD引言

项目部署

- 将项目通过maven进行编译打包
- 将文件上传到指定的服务器中
- 将war包放到tomcat的目录中
- 通过Dockerfile将Tomcat和war包转成一个镜像，由DockerCompose去运行容器

项目更新后，需要将上述流程再次的从头到尾的执行一次，如果每次更新一次都执行一次上述操作，很费时，费力。我们就可以通过CI、CD帮助我们实现持续集成，持续交付和部署。

11.2 CI介绍

CI (continuous integration) 持续集成

持续集成：编写代码时，完成了一个功能后，立即提交代码到Git仓库中，将项目重新的构建并且测试。

- 快速发现错误。
- 防止代码偏离主分支。

11.3 搭建Gitlab服务器

实现CI，需要使用到Gitlab远程仓库，先通过Docker搭建Gitlab

11.3.1 准备工作

- 创建一个全新的虚拟机，并且至少指定4G的运行内存，4G运行内存是Gitlab推荐的内存大小。
- 并且安装Docker以及Docker-Compose

11.3.2 修改ssh的22端口

将ssh的默认22端口，修改为60022端口，因为Gitlab需要占用22端口

```
vi /etc/ssh/sshd_config
PORT 22 -> 60022
systemctl restart sshd
```

11.3.3 编写docker-compose.yml

docker-compose.yml文件去安装gitlab（下载和运行的时间比较长的）

```
version: '3.1'
services:
  gitlab:
    image: 'twang2218/gitlab-ce-zh:11.1.4'
    container_name: "gitlab"
    restart: always
    privileged: true
    hostname: 'gitlab'
    environment:
      TZ: 'Asia/Shanghai'
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://192.168.199.110'
        gitlab_rails['time_zone'] = 'Asia/Shanghai'
        gitlab_rails['smtp_enable'] = true
        gitlab_rails['gitlab_shell_ssh_port'] = 22
    ports:
      - '80:80'
      - '443:443'
      - '22:22'
    volumes:
      - /opt/docker_gitlab/config:/etc/gitlab
      - /opt/docker_gitlab/data:/var/opt/gitlab
      - /opt/docker_gitlab/logs:/var/log/gitlab
```

11.4 搭建GitlabRunner

查看资料中的gitlab-runner文件即可安装

11.5 整合项目入门测试

11.5.1 创建项目

创建maven工程，添加web.xml文件，编写html页面

11.5.2 编写.gitlab-ci.yml

编写 .gitlab-ci.yml 文件

```
stages:
  - test

test:
  stage: test
  script:
    - echo first test ci # 输入的命令
```

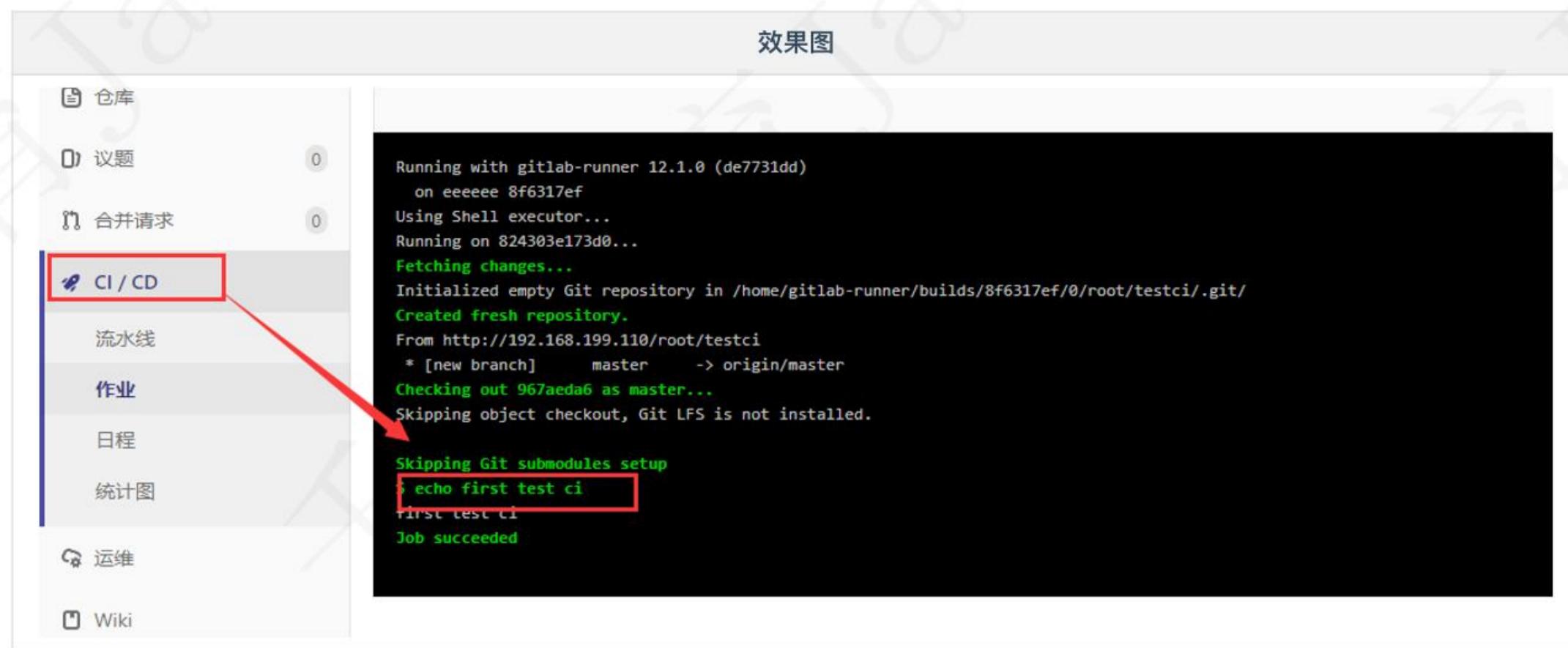
11.5.3 将maven工程推送到gitlab中

执行git命令推送到Gitlab

```
git push origin master
```

11.5.4 查看效果

可以在gitlab中查看到gitlab-ci.yml编写的内容



11.6 完善项目配置

添加Dockerfile以及docker-compose.yml，并修改.gitlab-ci.yml文件

11.6.1 创建Dockerfile

```
# Dockerfile
FROM daocloud.io/library/tomcat:8.5.15-jre8
COPY testci.war /usr/local/tomcat/webapps
```

11.6.2 创建docker-compose.yml

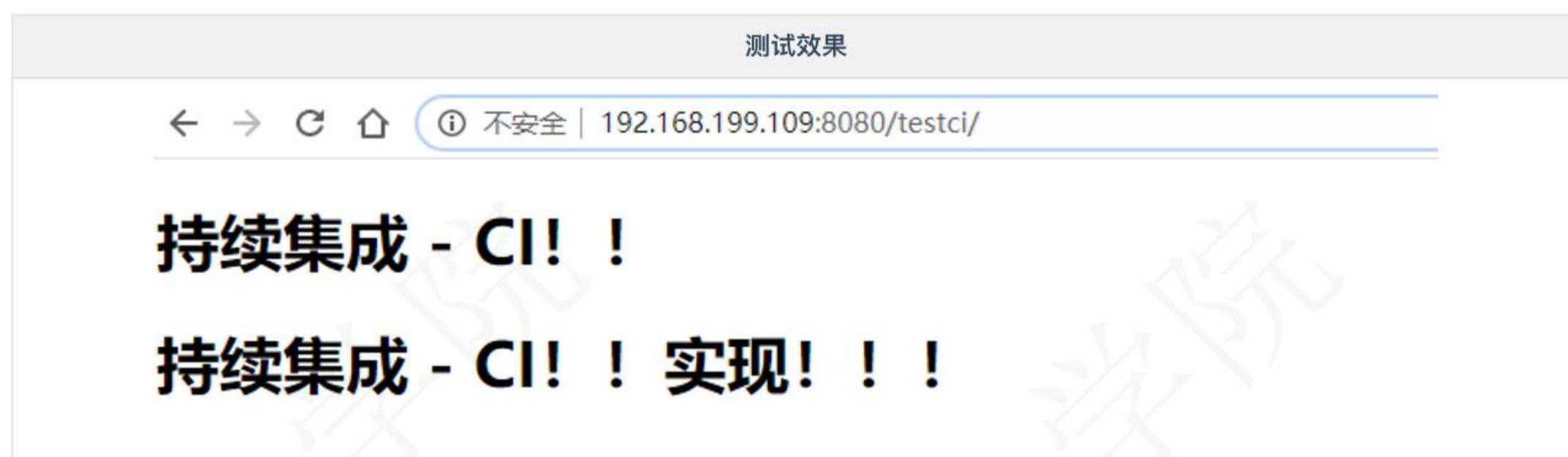
```
# docker-compose.yml
version: "3.1"
services:
  testci:
    build: docker
    restart: always
    container_name: testci
    ports:
      - 8080:8080
```

11.6.3 修改.gitlab-ci.yml

```
# ci.yml
stages:
- test

test:
  stage: test
  script:
    - echo first test ci
    - /usr/local/maven/apache-maven-3.6.3/bin/mvn package
    - cp target/testci-1.0-SNAPSHOT.war docker/testci.war
    - docker-compose down
    - docker-compose up -d --build
    - docker rmi $(docker images -qf dangling=true)
```

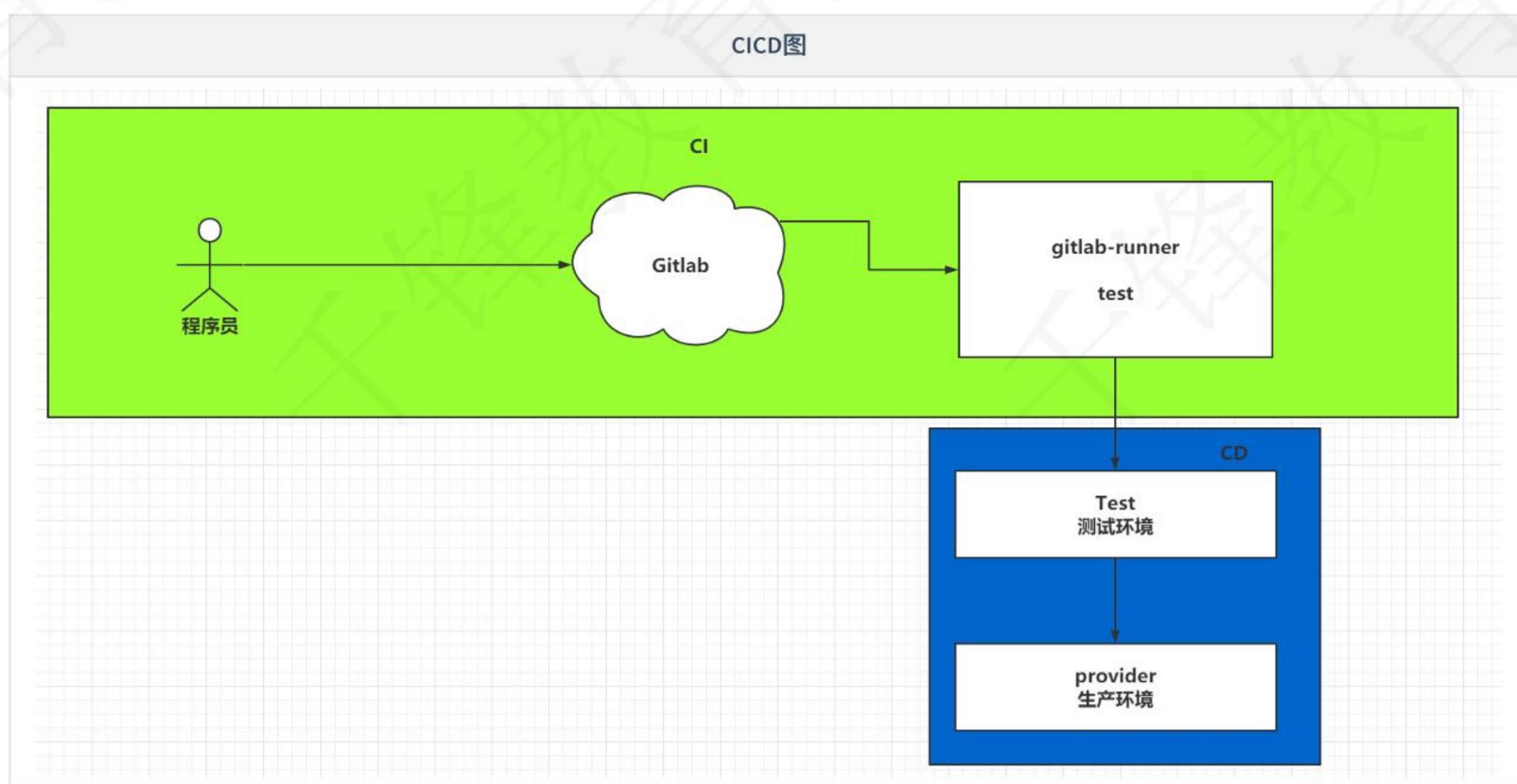
11.6.4 测试



11.7 CD介绍

CD (持续交付, 持续部署)

持续交付：将代码交付给专业的测试团队去测试
持续部署：可以直接将指定好tag的代码直接部署到生产环境中



11.8 安装Jenkins

11.8.1 编写docker-compose.yml

官网：<https://www.jenkins.io/>

```
version: "3.1"
services:
  jenkins:
    image: jenkins/jenkins
    restart: always
    container_name: jenkins
    ports:
      - 8888:8080
      - 50000:50000
    volumes:
      - ./data:/var/jenkins_home
```

11.8.2 运行并访问Jenkins

第一次运行时，会因为data目录没有权限，导致启动失败

```
chmod 777 data
```

访问<http://192.168.199.109:8888>

访问速度奇慢无比。。。。

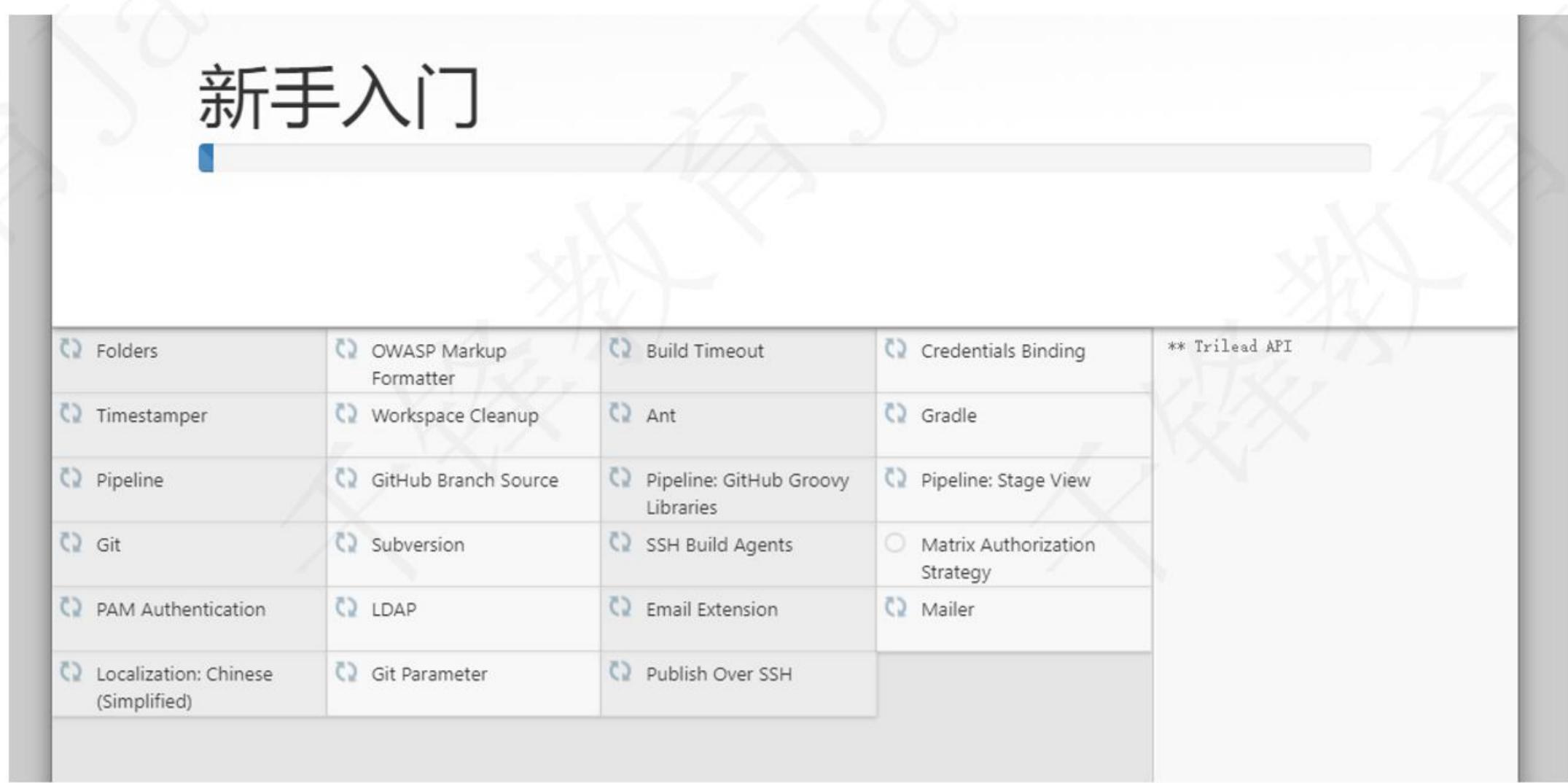
访问成功后，需要输入密码，可在日志中查看

```
jenkins
jenkins Jenkins initial setup is required. An admin user has been created and a password generated.
jenkins Please use the following password to proceed to installation:
jenkins cdafe9d7cf154ed2885e30da67571729
jenkins This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
jenkins
```

手动指定插件安装：指定下面两个插件即可

```
publish ssh.
```

```
git param.
```



安装成功后，需要指定用户名和密码，登陆成功

The screenshot shows the Jenkins dashboard. At the top, it says 'Jenkins'. On the right, there are links for 'ROOT' and '注销' (Logout). The main area displays the 'Welcome to Jenkins!' message with a button to '开始创建一个新任务' (Start creating a new job). On the left, there's a sidebar with various Jenkins management options like '新建Item', '用户列表', '构建历史', etc.

11.9 配置Jenkins的目标服务器

执行过程为代码提交到Gitlab，Jenkins会从Gitlab中拉取代码，并在Jenkins中打包并发布到目标服务器中。

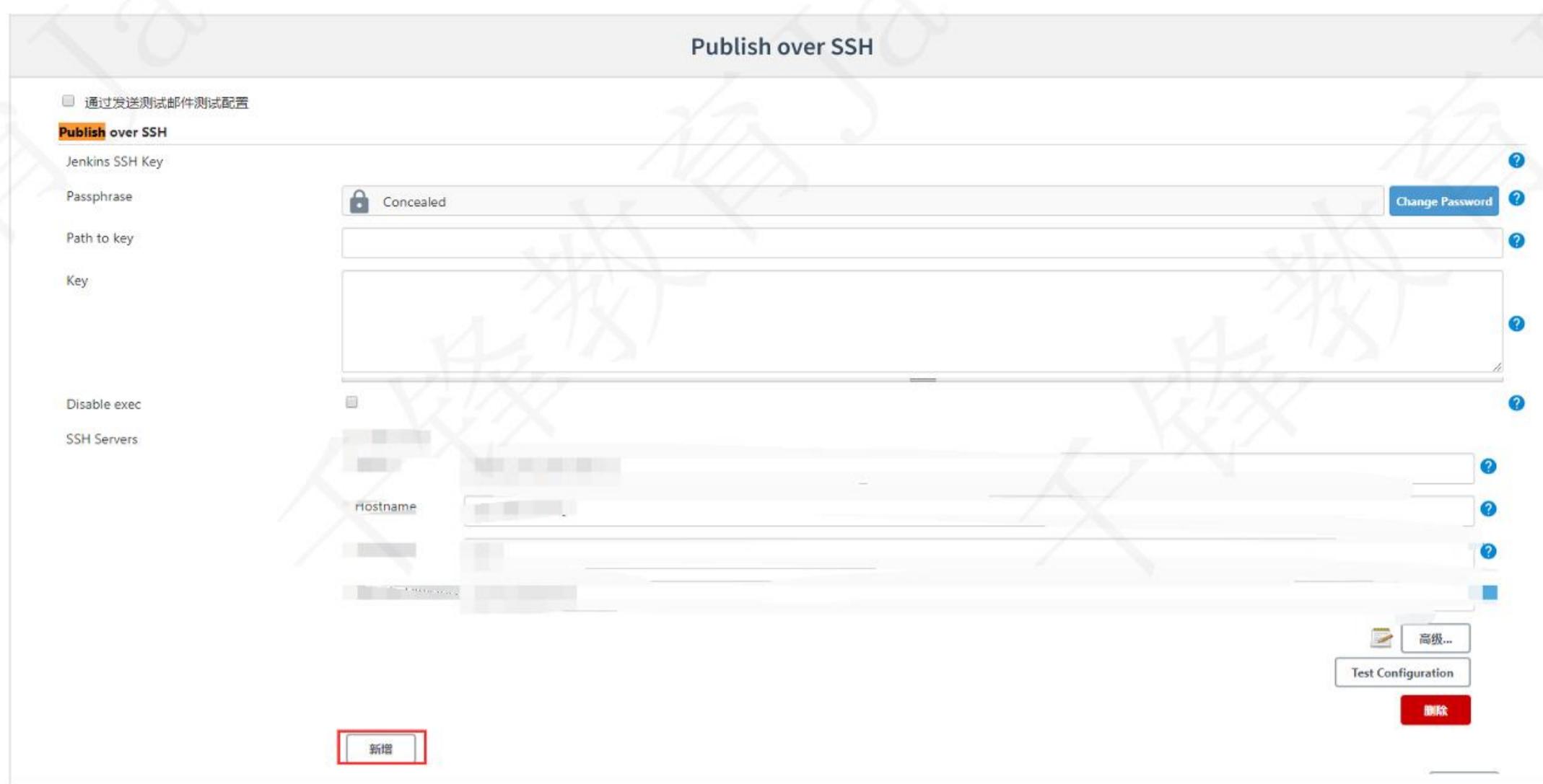
11.9.1 点击左侧的系统设置

The screenshot shows the Jenkins left sidebar. It has a '左侧导航' (Left Navigation) header. Underneath, there's a 'Manage Jenkins' link with a gear icon.

11.9.2 选中中间区域的系统设置

The screenshot shows the 'Configure System' page. It has a '系统设置' (System Settings) header. In the center, there's a 'Configure System' button with a gear icon and the text 'Configure global settings and paths.'

11.9.3 搜索Publish over SSH



11.9.4 点击上图新增

新增SSH连接

SSH Server

Name	指定目标服务器的别名
Hostname	指定目标服务器的ip
Username	目标服务器的用户名
Remote Directory	目标服务器哪个目录 (必须是存在的)

Use password authentication, or use a different key

Passphrase / Password: Linux的登录密码

Path to key

Key

Jump host

Port: 22

Timeout (ms): 300000

Disable exec

Proxy type

Proxy host

Proxy port

Proxy user

Proxy password

中 · · , 音量高 ?

11.10 配置GitLab免密码登录

链接Gitlab需要使用密码，我们可以通过SSH的方式，免密码登陆Gitlab拉取代码，避免每次都输入密码。

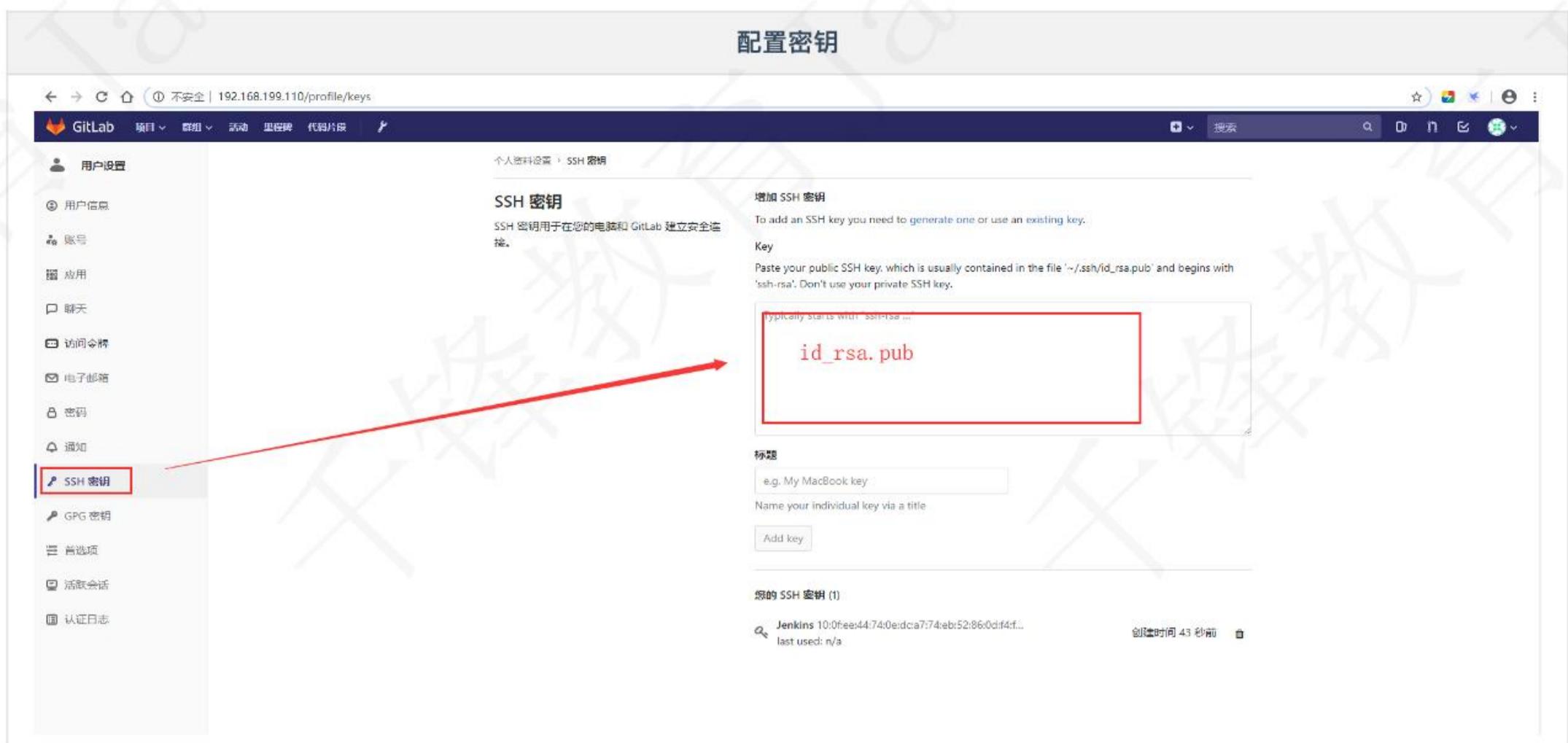
11.10.1 登录Jenkins容器内部

```
docker exec -it jenkins bash
```

11.10.2 输入生成SSH秘钥命令

```
ssh-keygen -t rsa -C "邮箱(随便写)"
```

11.10.3 将秘钥复制到GitLab的SSH中



11.11 配置JDK和Maven

我们需要在Jenkins中将代码打包，需要依赖JDK和Maven的环境

11.11.1 复制软件到data目录下

效果

```
[root@localhost data]# pwd  
/opt/docker_jenkins/data  
[root@localhost data]# ls  
apache-maven-3.6.3  
apache-maven-3.6.3-bin.tar.gz  
com.cloudbees.hudson.plugins.folder.config.AbstractFolderConfiguratio  
config.xml  
copy_reference_file.log  
github-plugin-configuration.xml  
hudson.model.UpdateCenter.xml  
hudson.plugins.build_timeout.operations.BuildStepOperation.xml  
hudson.plugins.emaiext.ExtendedEmailPublisher.xml  
hudson.plugins.git.GitSCM.xml  
hudson.plugins.git.GitTool.xml  
hudson.plugins.gradle.Gradle.xml  
hudson.plugins.timestamper.TimestamperConfig.xml  
hudson.scm.SubversionSCM.xml  
hudson.tasks.Ant.xml  
hudson.tasks.Mailer.xml  
hudson.tasks.Maven.xml  
hudson.tasks.Shell.xml  
hudson.triggers.SCMTrigger.xml  
identity.key.enc  
jdk1.8.0_231  
jdk-8u231-linux-x64.tar.gz
```

11.11.2 在监控界面中配置JDK和Maven

配置环境变量

The screenshot shows the Jenkins global configuration page under 'Manage Jenkins' > 'Configure System'. It lists several installed tools:

- JDK**: A section for managing Java Development Kits. One entry is shown with Name: jdk, JAVA_HOME: /var/jenkins_home/jdk1.8.0_231, and a '删除 JDK' (Delete JDK) button.
- Git**: A section for managing Git installations. One entry is shown with Name: Default, Path to Git executable: git, and a 'Delete Git' button.
- Gradle**: A section for managing Gradle installations. One entry is shown with a 'Delete Gradle' button.
- Ant**: A section for managing Ant installations. One entry is shown with a 'Delete Ant' button.
- Maven**: A section for managing Maven installations. One entry is shown with Name: maven, MAVEN_HOME: /var/jenkins_home/apache-maven-3.6.3, and a 'Delete Maven' button.

11.11.3 手动拉取gitlab项目

使用SSH无密码连接时，第一次连接需要手动确定

手动拉取一次

```
jenkins@b19b812f639e:~$ git clone git@192.168.199.110:root/testci.git
Cloning into 'testci'...
The authenticity of host '192.168.199.110 (192.168.199.110)' can't be established.
ECDSA key fingerprint is SHA256:AvPsDPAtDOHQ1GEB0RfSpc1p9Nb6BRz2aneg491Ilwg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.199.110' (ECDSA) to the list of Known hosts.
remote: Counting objects: 28, done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 28 (delta 7), reused 0 (delta 0)
Receiving objects: 100% (28/28), done.
Resolving deltas: 100% (7/7), done.
```

11.12 创建maven任务

实现通过Jenkins的Maven任务，自动去Gitlab拉取代码，并在本地打包，发布到目标服务器上

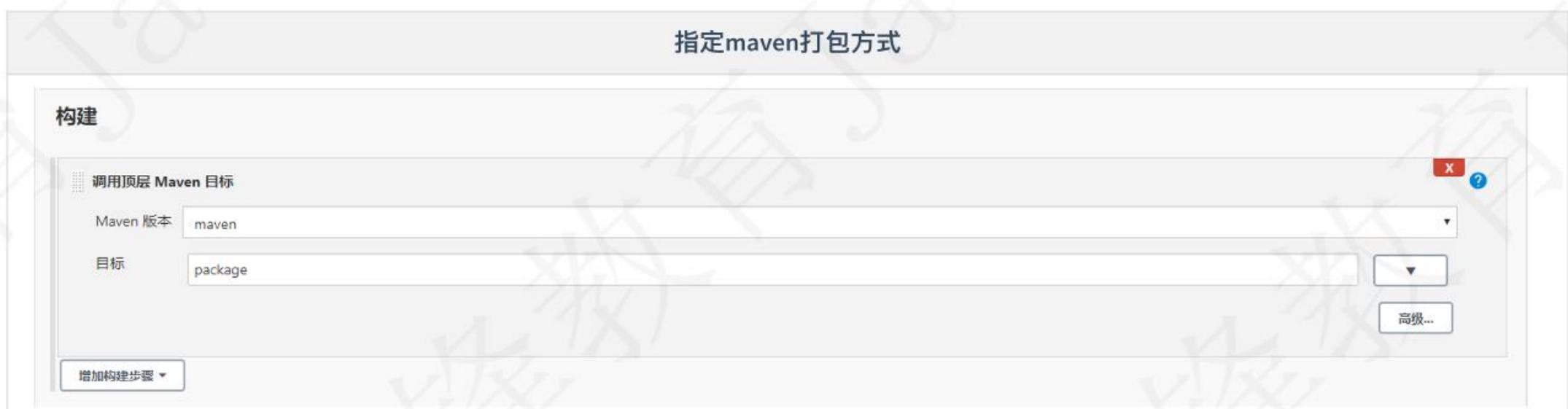
11.12.1 创建maven工程，推送到GitLab中

随便创建一个即可.....

11.12.2 Jenkins的监控页面中创建maven任务

指定GitLab地址

The screenshot shows the Jenkins 'Source Management' configuration for a new job. It is set to use a **Git** repository. The repository URL is specified as `git@192.168.199.110:root/testcd.git`. The 'Credentials' dropdown is set to '无' (None). Under 'Branches to build', the '指定分支 (为空时代表any)' (Specify branch (empty means any)) field contains `*/master`. There are buttons for '高级...' (Advanced), 'Add Repository' (Add Repository), and '增加分支' (Add Branch). Other sections like 'Additional Behaviours' are visible at the bottom.



11.12.3 执行maven任务

立即构建，并查看日志

Project testcd

相关链接

控制台查看日志信息

控制台输出

打包成功

11.12.4 最终效果

```
打包成功
[root@localhost testcd]# cd target
[root@localhost target]# ls
maven-archiver  testcd-1.0-SNAPSHOT  testcd-1.0-SNAPSHOT.war
[root@localhost target]# pwd
/opt/docker_jenkins/data/workspace/testcd/target
[root@localhost target]#
```

11.13 实现持续交付持续部署

实现根据tag标签，实现持续交付和持续部署

11.13.1 安装Persistent Parameter的插件

The screenshot shows the Jenkins 'Install Plugin' interface. A search bar at the top contains the text 'Persistent'. Below it, tabs include '可更新' (Updatable), '可选插件' (Optional Plugins) (which is selected), '已安装' (Installed), and '高级' (Advanced). A table lists available plugins under the heading '安装 1 名称'. The first item, 'Persistent Parameter', has a checked checkbox and is highlighted with a red border. Its description states: 'String, text, boolean and choice parameters with default values set from the previous build (if any)'. Below the table are three buttons: '直接安装' (Install Now), '下载待重启后安装' (Download for Reboot), and '立即获取' (Get Now).

11.13.2 重新指定构建项目的方式

The screenshot shows the Jenkins project configuration interface. The top section is titled '根据标签构建项目' (Build with Label) and contains a 'Git 参数' (Git Parameters) configuration for a 'Tag' parameter. It includes fields for '名称' (Name), '描述' (Description), '参数类型' (Parameter Type) set to '标签' (Label), and '默认值' (Default Value) set to 'master'. The bottom section is titled '自定义构建' (Custom Build) and contains a '构建' (Build) configuration with a '执行 shell' (Execute Shell) step. The command field contains the following Jenkinsfile code:

```
echo $Tag
cd /var/jenkins_home/workspace/testcd
git checkout $Tag
git pull origin $Tag
/var/jenkins_home/apache-maven-3.6.3/bin/mvn clean package
```

11.13.3 构建项目成功后，需要将内容发布到目标服务器

The screenshot shows the Jenkins 'Send build artifacts over SSH' configuration screen. It is a 'SSH Publishers' configuration. Under the 'Transfers' section, there is a 'Transfer Set' configuration with the following details:

- Source files: `**/*.war,docker-compose.yml,*.docker/**`
- Remove prefix: `target/`
- Remote directory: `testcd`
- Exec command:

```
cd /usr/local/jenkins/testcd
cp target/testcd-1.0-SNAPSHOT.war docker
docker-compose down
docker-compose up -d --build
docker image prune -f
```

A note at the bottom states: 'All of the transfer fields (except for Exec timeout) support substitution of Jenkins environment variables.'

11.13.4 添加程序代码

```
# Dockerfile 文件
FROM daocloud.io/library/tomcat:8.5.15-jre8
COPY testcd-1.0-SNAPSHOT.war /usr/local/tomcat/webapps

# docker-compose.yml文件
version: "3.1"
services:
  testcd:
    build: docker
    restart: always
    container_name: testcd
    ports:
      - 8081:8080
```

11.13.5 测试

