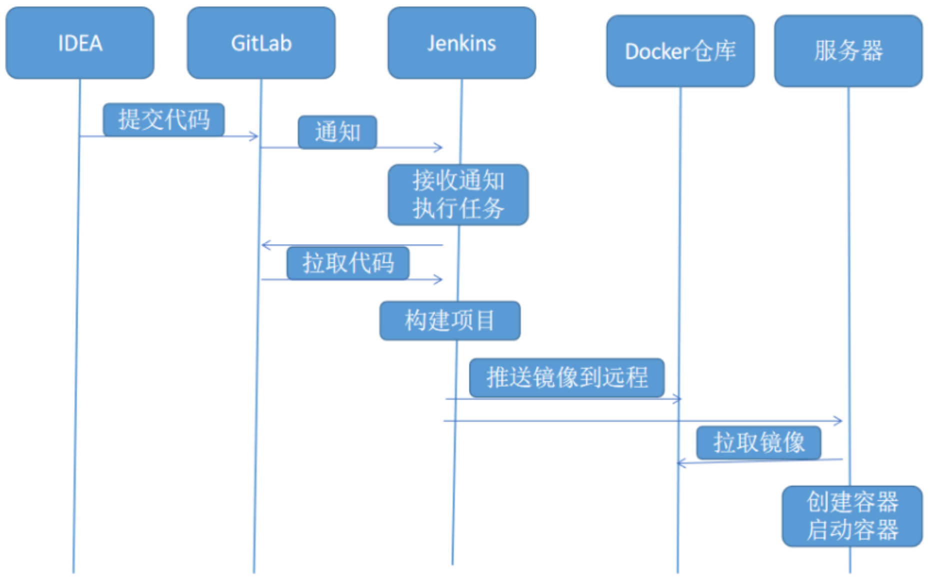
Jenkins+Docker+Gitlab三剑客实现自动化服务部署

概述

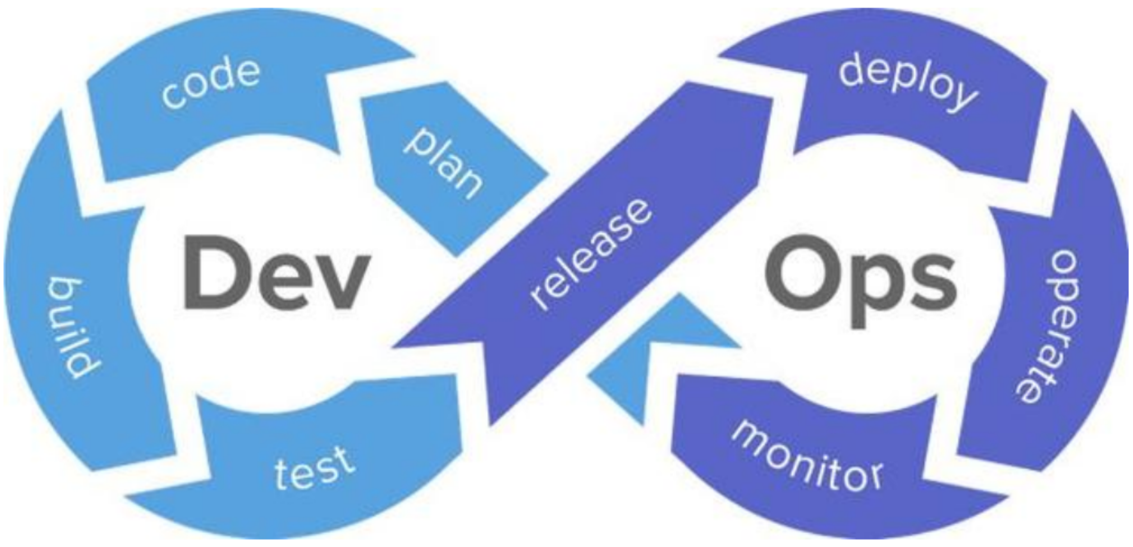
持续集成流程：



DevOps

> https://www.zhihu.com/question/58702398

DevOps是一种思想或方法论，它涵盖开发，测试，运维的整个流程，DevOps强调软件开发人员与软件测试，软件运维，质量保障（QA）部门之间有效的沟通与协作，强调通过自动化的方法管理软件变更、软件集成，使软件从构建到测试、发布更加快捷、可靠，最终按时交付软件。



1.安装启动Docker

1.1 安装docker

```
# 1.yum包更新
yum update
# 2.安装需要的软件包，yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的
yum install -y yum-utils device-mapper-persistent-data lvm2
# 3.设置yum源
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
# 4.安装docker，出现输入的界面都按 Y
yum install -y docker-ce
# 5.查看docker版本，验证是否成功
docker -v
```

1.2 配置镜像加速

https://liuurick.github.io/2019/12/17/Centos7%E4%B8%8A%E5%AE%89%E8%A3%85docker/

1.3 安装私有仓库

```
#搜索镜像
docker search registry
#拉取镜像
docker pull registry
#创建容器
docker run -d -p 5000:5000 registry
#配置私有仓库地址
vim /etc/docker/daemon.json
{
    "insecure-registries": [xxx.xxx.xxx.xxx:xxx]
}

sudo systemctl daemon-reload
sudo systemctl restart docker

#启动本地仓库容器
docker start 容器ID
```

1.4 访问私有仓库

http://192.168.60.129:5000/v2/_catalog

注意：如果访问不到，关闭防火墙，或者开放端口

https://liuurick.github.io/2019/01/11/%E9%98%B2%E7%81%AB%E5%A2%99%E7%AE%A1%E7%90%86/

2.Docker下gitlab安装配置使用

2.1 安装

```
# 1.查找gitlab镜像
docker search gitlab
# 2.gitlab镜像拉取
docker pull gitlab/gitlab-ce
# 3.查看本地镜像
docker images

# 4.本机建立的3个目录
# 为了gitlab容器通过挂载本机目录启动后可以映射到本机，然后后续就可以直接在本机查看和编辑了，不用再进行
容器操作
# 5.配置文件
mkdir -p /home/gitlab/etc
# 6.数据文件
mkdir -p /home/gitlab/data
# 7.日志文件
mkdir -p /home/gitlab/logs

# 启动容器
docker run --name='gitlab' -d \
    --publish 4443:443 --publish 8888:80 \
    -v /home/gitlab/etc:/etc/gitlab \
    -v /home/gitlab/data:/var/opt/gitlab \
    -v /home/gitlab/logs:/var/log/gitlab \
    gitlab/gitlab-ce:latest
# 查看启动日志
docker logs -f gitlab
```

2.2 配置

> 按上面的方式，gitlab容器运行没问题，但在gitlab上创建项目的时候，生成项目的URL访问地址是按容器的hostname来生成的，也就是容器的id。作为gitlab服务器，我们需要一个固定的URL访问地址，于是需要配置gitlab.rb（宿主机路径：/home/github/config/gitlab.rb）

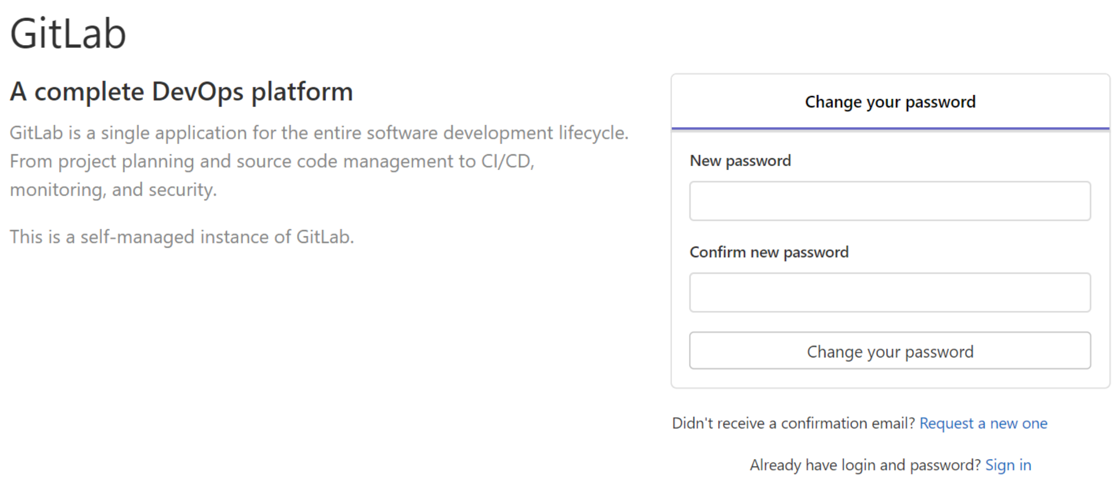
```
#配置域名或IP

#配置gitlab.rb
cd /home/gitlab/etc
vim gitlab.rb

#配置http协议所使用的访问地址，不加端口号默认为80
external_url 'http://192.168.60.129'

#配置gitlab.yml
cd /home/gitlab/data/gitlab-rails/etc
vim gitlab.yml

gitlab:
    host: 192.168.60.129
    port: 8888
```

```
      https: false
```

2.3 初始化密码

gitlab默认管理用户是root

登录： http://192.168.60.129:8888 登录修改root的密码



GitLab

A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

Change your password

New password

Confirm new password

Change your password

Didn't receive a confirmation email? Request a new one

Already have login and password? Sign in

2.4 登录

用户名密码

2.5 创建项目

http-demo

3.安装git

```
# 安装
yum install -y git
# 查看版本
git version
```

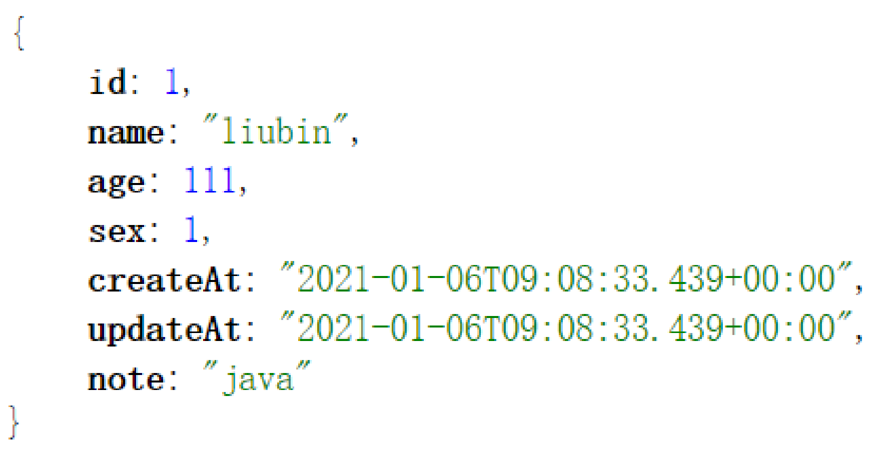
4.使用git管理项目

4.1 使用IDEA从Gitlab检出空项目

项目地址：http://192.168.60.129:8888/root/http-demo.git

4.2 复制项目并允许

IDEA中运行项目并访问：http://127.0.0.1:10000/user/1



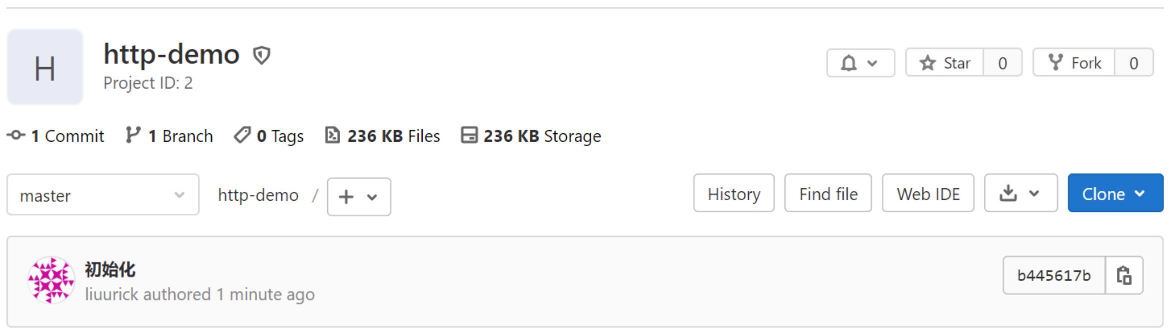
```
{
    id: 1,
    name: "liubin",
    age: 111,
    sex: 1,
    createAt: "2021-01-06T09:08:33.439+00:00",
    updateAt: "2021-01-06T09:08:33.439+00:00",
    note: "java"
}
```

注意：记得开启防火墙端口

4.3 提交代码到gitlab

在工程根目录创建 `.gitignore`，此文件中记录了在提交代码时哪些文件或目录被忽略

```
.idea/
target/
*.iml
```



H http-demo
Project ID: 2

Star 0 Fork 0

1 Commit 1 Branch 0 Tags 236 KB Files 236 KB Storage

master http-demo / + History Find file Web IDE Clone

初始化
liuurick authored 1 minute ago
b4456l7b

5.SpringBoot工程制作镜像

5.1 SpringBoot允许jar包

- 打包配置pom

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

- 使用maven打包并运行访问微服务工程

```
maven install
```

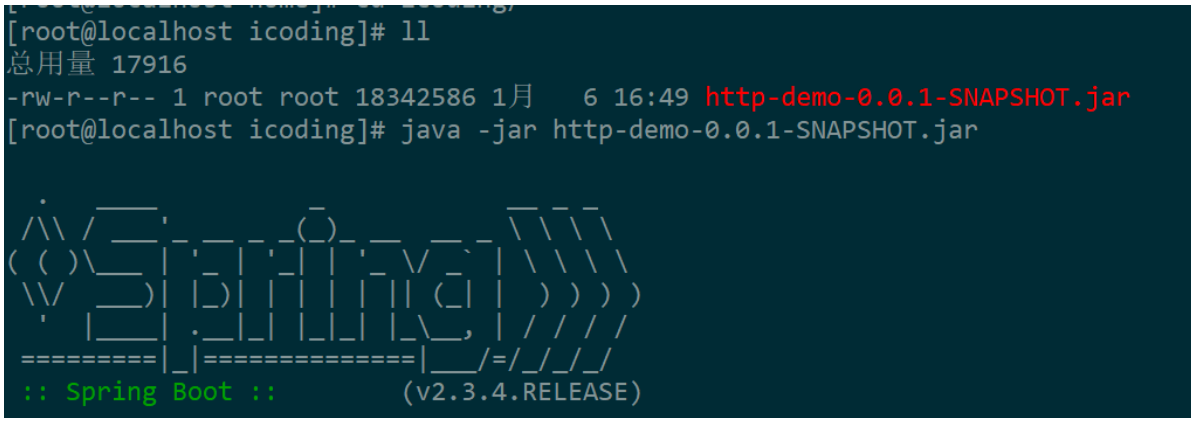
5.2 创建Docker镜像

在linux上新建一个目录，将上一步的jar包拷贝到Linux服务器，准备创建镜像

```
cd /home
mkdir icoding
```

测试jar包是否可以运行，执行:java -jar

```
java -jar http-demo-1.0-SNAPSHOT.jar
```



```
[root@localhost icoding]# ll
总用量 17916
-rw-r--r-- 1 root root 18342586 1月   6 16:49 http-demo-0.0.1-SNAPSHOT.jar
[root@localhost icoding]# java -jar http-demo-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.4.RELEASE)
```

访问：http://192.168.60.129:10000/get/user/1

在http-demo-1.0-SNAPSHOT.jar所在文件夹位置编写Dockerfile文件

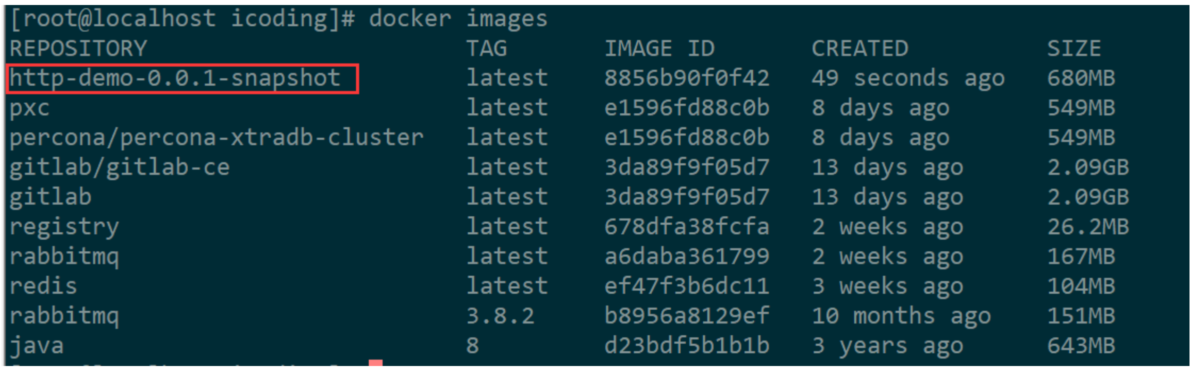
```
vim Dockerfile

FROM java:8
#VOLUME 指定了临时文件目录/tmp
#其效果是在主机 /var/lib/docker 目录下创建了一个临时文件，并链接到容器的/tmp
VOLUME /tmp
# 将jar包添加到容器中并更名为app.jar
ADD http-demo-0.0.1-SNAPSHOT.jar app.jar
#运行jar包
RUN bash -c 'touch /app.jar'
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

在Dockerfile文件所在目录创建镜像

```
docker build -t http-demo-0.0.1-snapshot .
```

查看镜像

```
docker images
```



```
[root@localhost icoding]# docker images
REPOSITORY                        TAG       IMAGE ID       CREATED          SIZE
http-demo-0.0.1-snapshot          latest    8856b90f0f42   49 seconds ago   680MB
pxc                               latest    e1596fd88c0b   8 days ago       549MB
percona/percona-xtradb-cluster    latest    e1596fd88c0b   8 days ago       549MB
gitlab/gitlab-ce                  latest    3da89f9f05d7   13 days ago      2.09GB
gitlab                            latest    3da89f9f05d7   13 days ago      2.09GB
registry                          latest    678dfa38fcfa   2 weeks ago      26.2MB
rabbitmq                          latest    a6daba361799   2 weeks ago      167MB
redis                             latest    ef47f3b6dc11   3 weeks ago      104MB
rabbitmq                          3.8.2     b8956a8129ef   10 months ago    151MB
java                              8         d23bdf5b1b1b   3 years ago      643MB
```

再次运行项目：

```
docker run -d -p 10000:10000 http-demo-0.0.1-snapshot
```

5.3 创建启动容器

5.4访问界面

5.5 停止与删除

5.6 使用maven构建镜像

第二种方法通过maven的 `docker-maven-plugin` 插件可完成从打包到构建镜像，构建容器等过程

5.6.1 编写pom_docker.xml

```
<plugin>
                <groupId>com.spotify</groupId>
                <artifactId>docker-maven-plugin</artifactId>
                <version>1.0.0</version>
                <!--docker镜像相关的配置信息-->
                <configuration>
                    <!--镜像名，这里使用工程名-->
                    <imageName>${project.artifactId}</imageName>
                    <!--Dockerfile文件所在目录-->

 <dockerDirectory>${project.basedir}/src/main/resources</dockerDirectory> <!-- 指定
Dockerfile 路径-->
                    <!--TAG，这里用工程版本号-->
                    <imageTags>
```

```
                        <imageTag>${project.version}</imageTag>
                    </imageTags>
                    <!--构建镜像的配置信息-->
                    <resources>
                        <resource>
                            <targetPath>/</targetPath>
                            <directory>${project.build.directory}</directory>

 <include>${project.artifactId}-${project.version}.jar</include>
                        </resource>
                    </resources>
                </configuration>
            </plugin>
```

5.6.2 拷贝Dockerfile文件

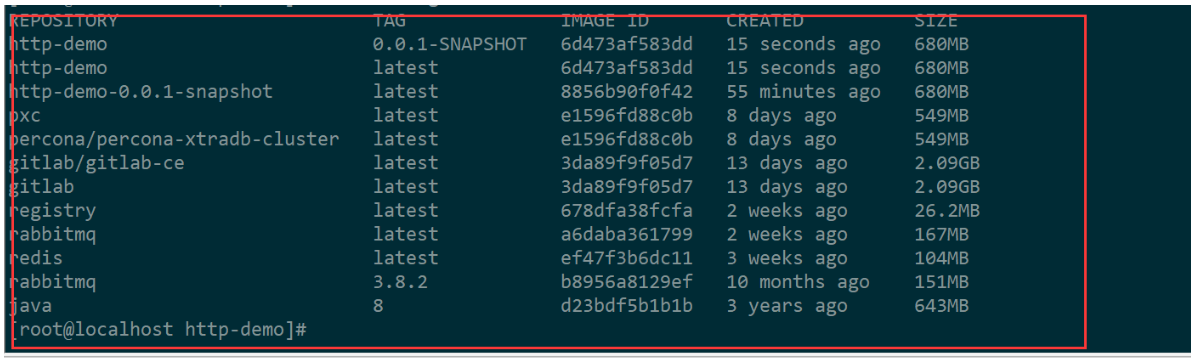
5.6.3 在IDEA中提交修改的文件

5.6.4 clone最新项目

```
git clone http://192.168.60.129:8888/root/http-demo.git
```

5.6.5 打包构建镜像

```
#进入工程目录
cd http-demo
# 打包构建镜像
mvn -f pom.xml clean package -DeskipTests docker:build
```

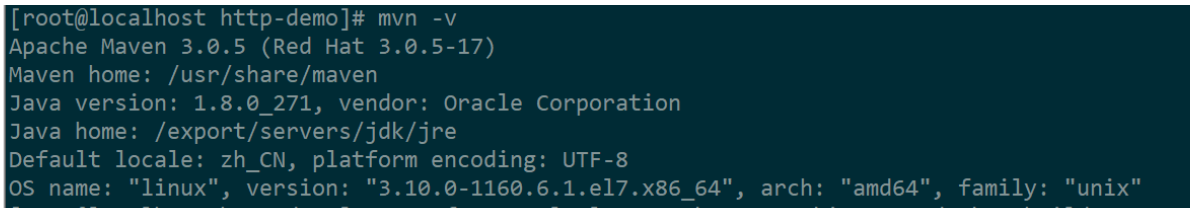


```
REPOSITORY                          TAG              IMAGE ID        CREATED          SIZE
http-demo                           0.0.1-SNAPSHOT   6d473af583dd    15 seconds ago   680MB
http-demo                           latest           6d473af583dd    15 seconds ago   680MB
http-demo-0.0.1-snapshot            latest           8856b90f0f42    55 minutes ago   680MB
pxc                                 latest           e1596fd88c0b    8 days ago       549MB
percona/percona-xtradb-cluster      latest           e1596fd88c0b    8 days ago       549MB
gitlab/gitlab-ce                    latest           3da89f9f05d7    13 days ago      2.09GB
gitlab                              latest           3da89f9f05d7    13 days ago      2.09GB
registry                            latest           678dfa38fcfa    2 weeks ago      26.2MB
rabbitmq                            latest           a6daba361799    2 weeks ago      167MB
redis                               latest           ef47f3b6dc11    3 weeks ago      104MB
rabbitmq                            3.8.2            b8956a8129ef    10 months ago    151MB
java                                8                d23bdf5b1b1b    3 years ago      643MB
[root@localhost http-demo]#
```

注意：

- 建议还是一个pom文件吧
- 这里需要安装一下maven

```
yum install maven -y
```



```
[root@localhost http-demo]# mvn -v
Apache Maven 3.0.5 (Red Hat 3.0.5-17)
Maven home: /usr/share/maven
Java version: 1.8.0_271, vendor: Oracle Corporation
Java home: /export/servers/jdk/jre
Default locale: zh_CN, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-1160.6.1.el7.x86_64", arch: "amd64", family: "unix"
```

5.6.6 创建启动容器

基于http-demo:1.0-SNAPSHOT镜像创建容器，容器名称为http-demo

```
docker run -d -p 10000:10000 http-demo:1.0-SNAPSHOT
```

容器创建成功，可以通过`docker ps -a`命令查看

5.6.7 访问界面

6.持续集成

> http://www.ruanyifeng.com/blog/2015/09/continuous-integration.html

传统的软件开发流程如下：

1. 项目经理分配模块给开发人员
2. 每个模块的开发人员并行开发，并进行单元测试
3. 开发完毕，将代码集成部署到测试服务器，测试人员进行测试
4. 测试人员发现bug，提交bug，开发人员修改bug
5. bug修改完毕再次集成、测试

问题描述：

1. 模块直接依赖关系复杂，在集成时发现大量bug
2. 测试人员等待测试时间过长
3. 软件交付无法保障

解决上述问题思考：

1. 能否集成测试时间提前？
2. 能否使用自动化工具代替人工集成部署的过程？

持续集成：

> 持续集成（Continuous integration，简称CI），持续集成的思想是每天要多次将代码合并到主干，并进行集成，测试，这样就可以提早发现错误，进行修正。持久集成也属于DevOps

持续集成的好处：

1. 自动化集成部署，提高了集成效率
2. 更快的修复问题
3. 更快的进行交付
4. 提高了产品质量

7.Jenkins安装配置使用

Jenkins一个领先自动化服务器的安装与配置

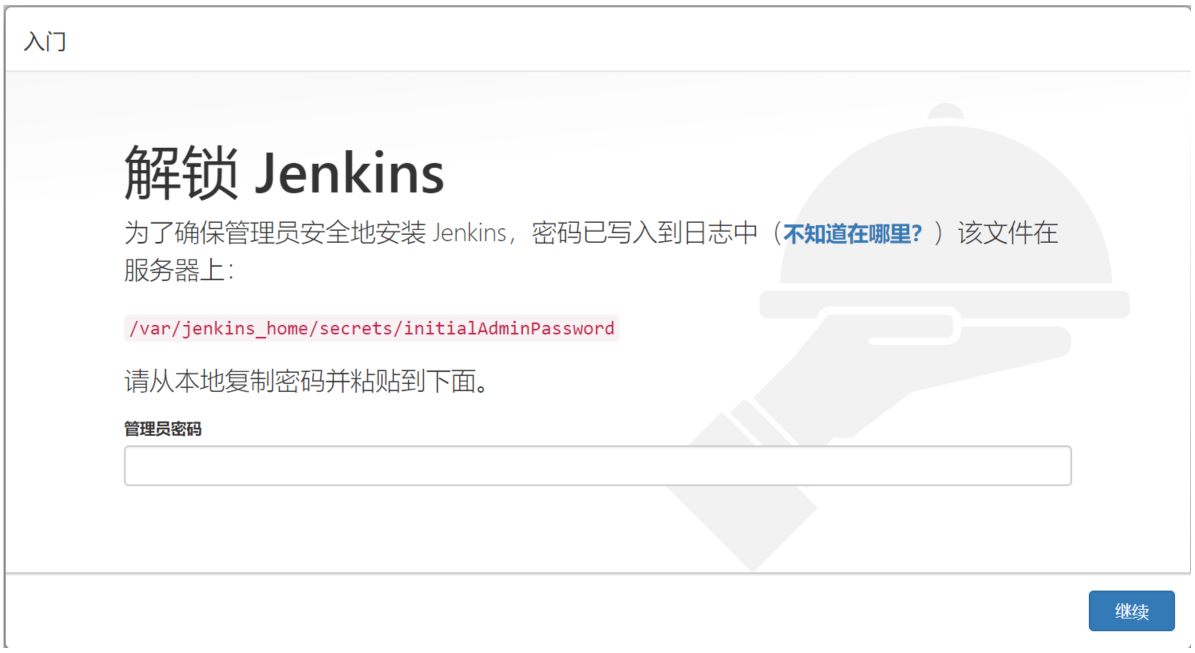
> https://www.jenkins.io/

7.1 安装

```
#搜索镜像
docker search jenkins
#拉取镜像
docker pull jenkins/jenkins:lts
#创建容器
docker run --name=jenkins\
  -u root \
  --rm \
  -d \
  -p 8080:8080 \
  -p 50000:50000 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /usr/bin/docker:/usr/bin/docker \
  -v /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.242.b08-
0.el7_7.x86_64:/usr/java/jdk1.8.0_181 \
  -v /usr/local/maven3:/usr/local/maven \
  -v /usr/local/maven_repository:/usr/local/maven_repository \
  -v /home/jenkins-data:/var/jenkins_home \
  jenkins/jenkins:lts
```

7.2 解锁Jenkins

http://192.168.60.129:8080/ 首次登录需要解锁Jenkins



入门

解锁 Jenkins

为了确保管理员安全地安装 Jenkins，密码已写入到日志中（**不知道在哪里?**）该文件在服务器上：

`/var/jenkins_home/secrets/initialAdminPassword`

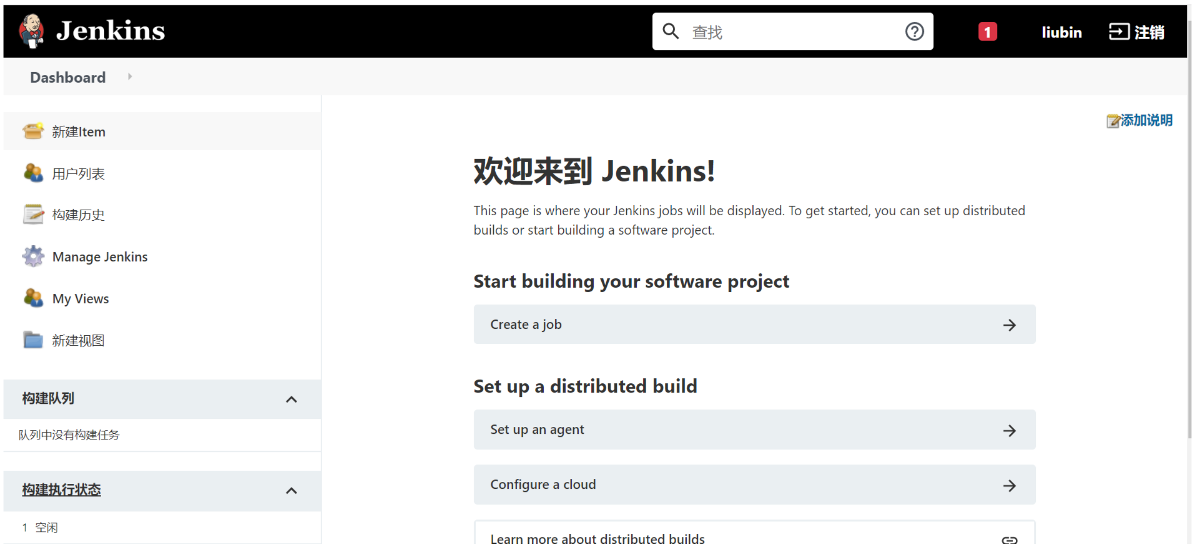
请从本地复制密码并粘贴到下面。

管理员密码

进入容器内部docker exec -it jenkins bash

/home/jenkins-data:/var/jenkins_home/secrets/initialAdminPassword，得到密码并粘贴过去

然后是自动配置，配置了好久。。。。。。。。



欢迎来到 Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

Set up a distributed build

Set up an agent

Configure a cloud

Learn more about distributed builds

7.3登录

8.持续集成

8.1 编写pom.xml文件

使用Jenkins进行构建，在http-demo工程根目录编写pom_docker_registry.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.liuurick</groupId>
    <artifactId>http-demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>http-demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
```

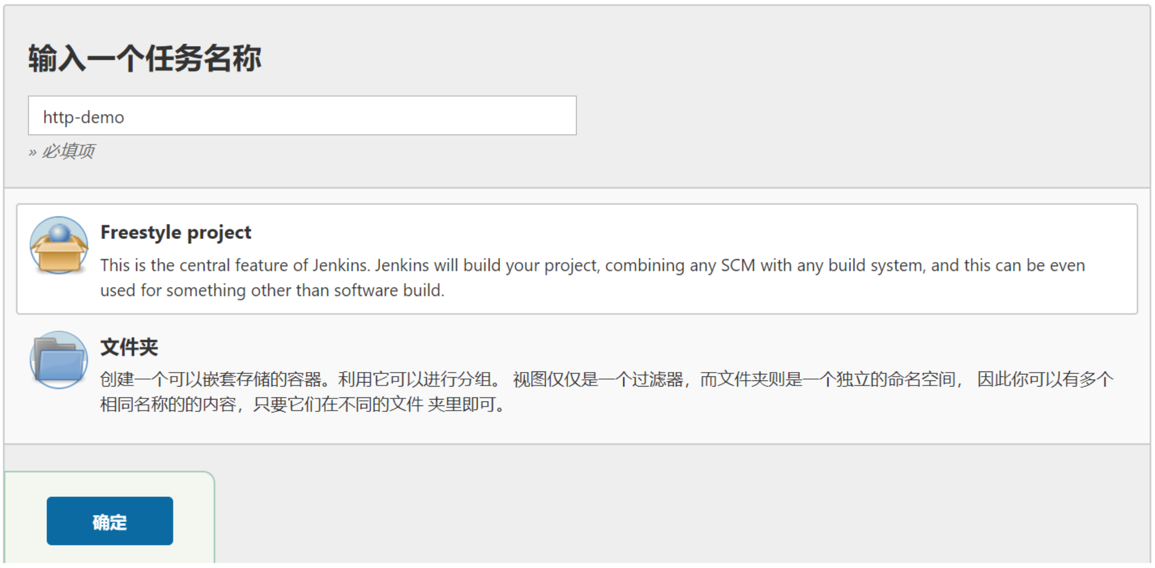
```xml
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>

            <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>docker-maven-plugin</artifactId>
                <version>1.0.0</version>
                <configuration>

 <dockerDirectory>${project.basedir}/src/main/resources</dockerDirectory>
                     <imageTags>
                        <imageTag>${project.version}</imageTag>
                    </imageTags>
                    <registryUrl>192.168.60.129:5000</registryUrl>
                    <pushImage>true</pushImage>
                    <imageName>192.168.60.129:5000/${project.artifactId}</imageName>
                    <resources>
                        <resource>
                            <targetPath>/</targetPath>
                            <directory>${project.build.directory}</directory>

 <include>${project.artifactId}-${project.version}.jar</include>
                        </resource>
                    </resources>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

提交 gitlab仓库

8.2.创建持续集成任务

8.2.1 创建构建任务

```
http-demo
```



输入一个任务名称

http-demo

» 必填项

Freestyle project

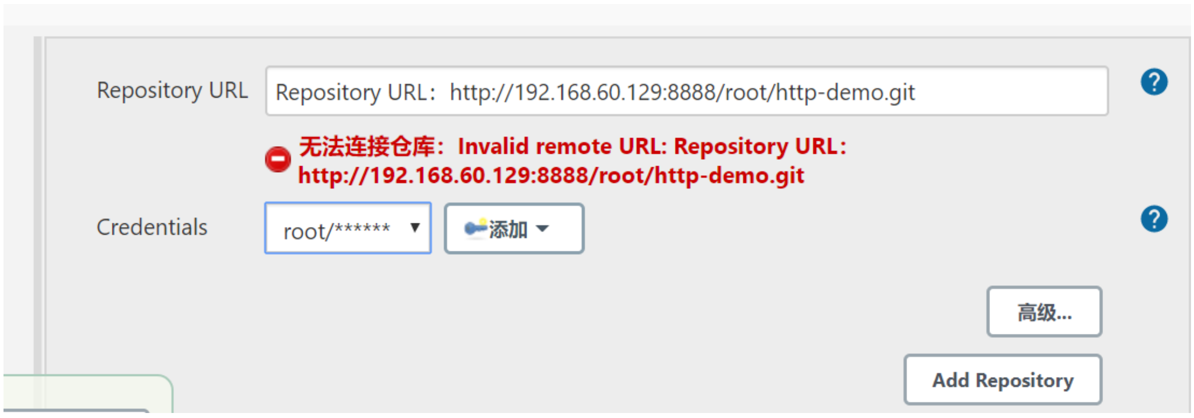
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

文件夹

创建一个可以嵌套存储的容器。利用它可以进行分组。 视图仅仅是一个过滤器，而文件夹则是一个独立的命名空间， 因此你可以有多个相同名称的的内容，只要它们在不同的文件 夹里即可。

8.2.2 配置git仓库

```
Repository URL：http://192.168.60.129:8888/root/http-demo.git
```



Repository URL: Repository URL：http://192.168.60.129:8888/root/http-demo.git

无法连接仓库：Invalid remote URL: Repository URL：http://192.168.60.129:8888/root/http-demo.git

Credentials: root/******

8.2.3 maven构建配置

使用shell脚本停止容器、删除容器、删除镜像,shell脚本如下：

```
#!/bin/bash
result=$(docker ps | grep "192.168.60.129:5000/http-demo")
if [[ "$result" != "" ]]
then
echo "stop http-demo"
docker stop http-demo
fi
result1=$(docker ps -a | grep "192.168.60.129:5000/http-demo")
if [[ "$result1" != "" ]]
then
```

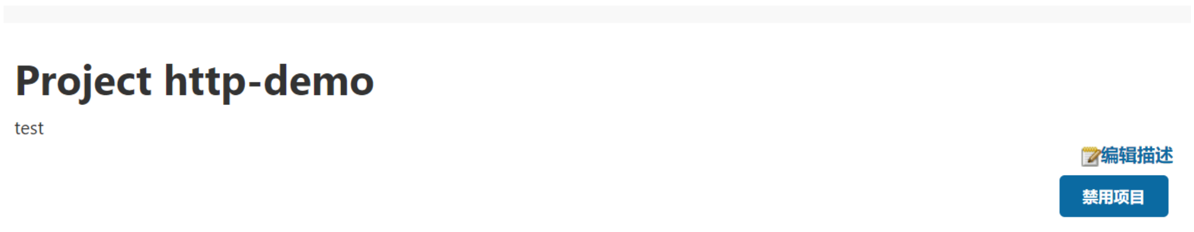
```
echo "rm http-demo"
docker rm http-demo
fi
result2=$(docker images | grep "192.168.60.129:5000/http-demo")
if [[ "$result2" != "" ]]
then
echo "192.168.60.129:5000/http-demohttp-demo:0.0.2-SNAPSHOT"
docker rmi 192.168.60.129:5000/http-demo-0.0.1-SNAPSHOT
fi
```

执行maven构建

```
clean package -f pom_docker_registry.xml  -DskipTests docker:build
```

拉取镜像，创建容器，启动容器

```
docker run --name http-demo -p 10000:10000 -idt 192.168.60.129:5000/http-demo-0.0.1-SNAPSHOT
```


Project http-demo

test

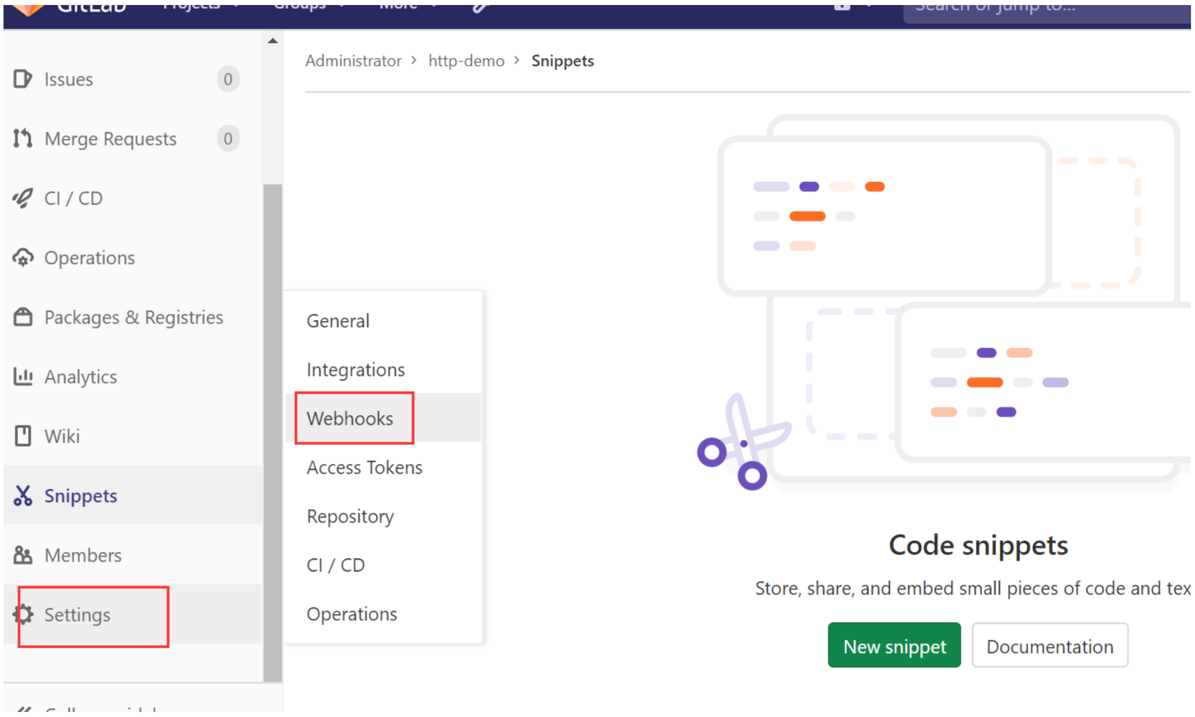
8.4 执行任务

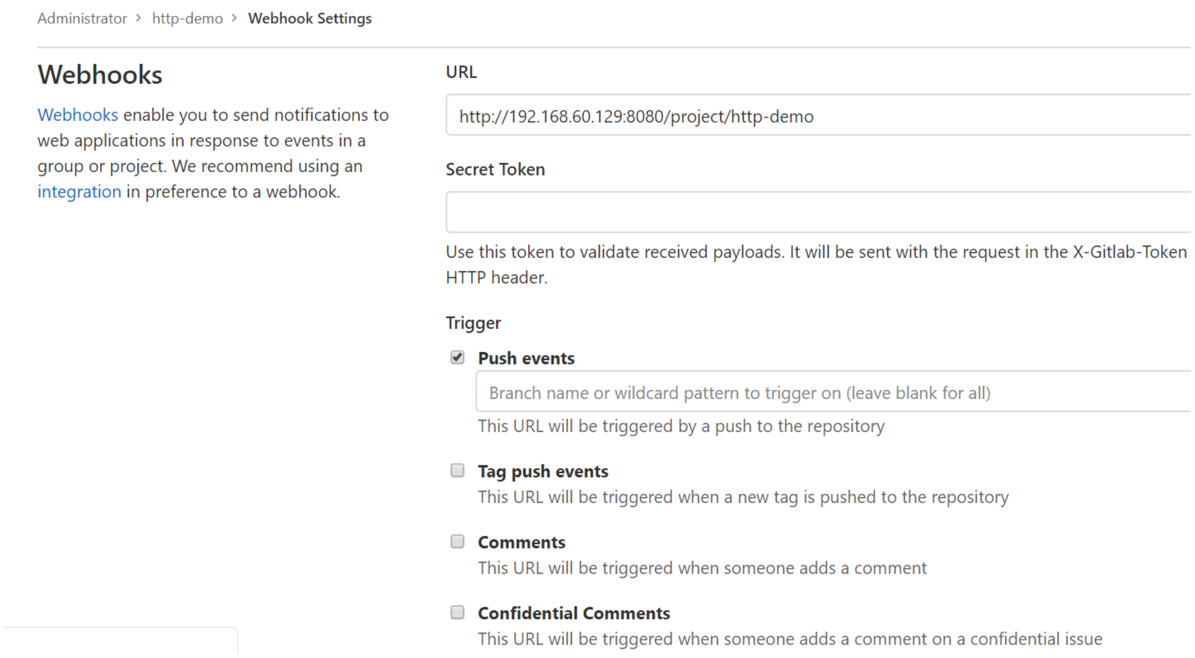
修改代码内容自动构建

注意： 可能会遇到容器端口号冲突的问题，docker stop即可

8.5 自动构建

```
jenkins中拿到钩子地址：**Build Triggers**

gitlab中配置钩子地址：http://192.168.60.129:8080/project/http-demo
```




Administrator › http-demo › Webhook Settings

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

http://192.168.60.129:8080/project/http-demo

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

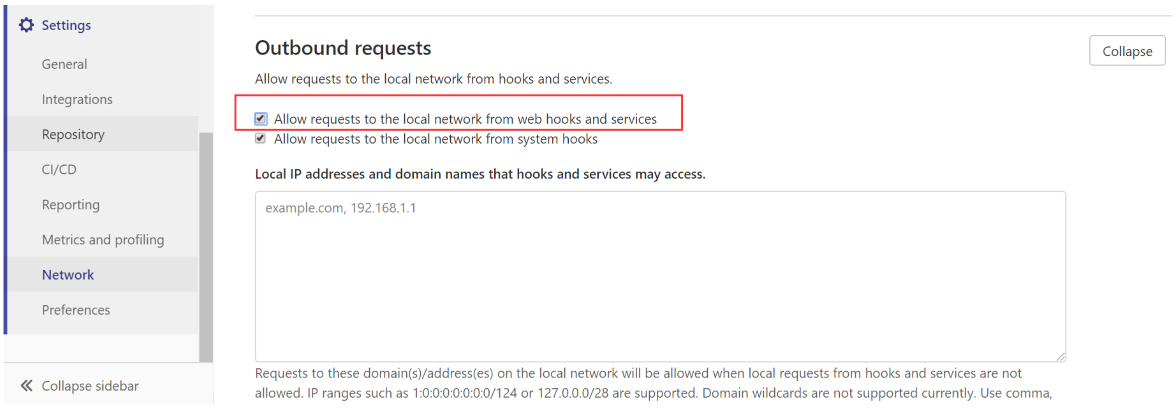
- [x] **Push events**
  Branch name or wildcard pattern to trigger on (leave blank for all)
  This URL will be triggered by a push to the repository
- [ ] **Tag push events**
  This URL will be triggered when a new tag is pushed to the repository
- [ ] **Comments**
  This URL will be triggered when someone adds a comment
- [ ] **Confidential Comments**
  This URL will be triggered when someone adds a comment on a confidential issue

添加成功后：Test--》Push events测试一下

注意：

- URL is blocked Requests to the local network are not allowed

  Admin area-->Settings-->Network-->勾选



Outbound requests

Allow requests to the local network from hooks and services.

- [✓] Allow requests to the local network from web hooks and services
- [✓] Allow requests to the local network from system hooks

Local IP addresses and domain names that hooks and services may access.

example.com, 192.168.1.1

Requests to these domain(s)/address(es) on the local network will be allowed when local requests from hooks and services are not allowed. IP ranges such as 1:0:0:0:0:0:0:0/124 or 127.0.0.0/28 are supported. Domain wildcards are not supported currently. Use comma,

- 通过用户名密码构建的不需要配置gitlab