

1.部署前端项目

2.部署后端项目

2.1 整体步骤

2.2.1 后端gitlab配置文件.gitlab-ci.yml

2.2.2 部署jar包

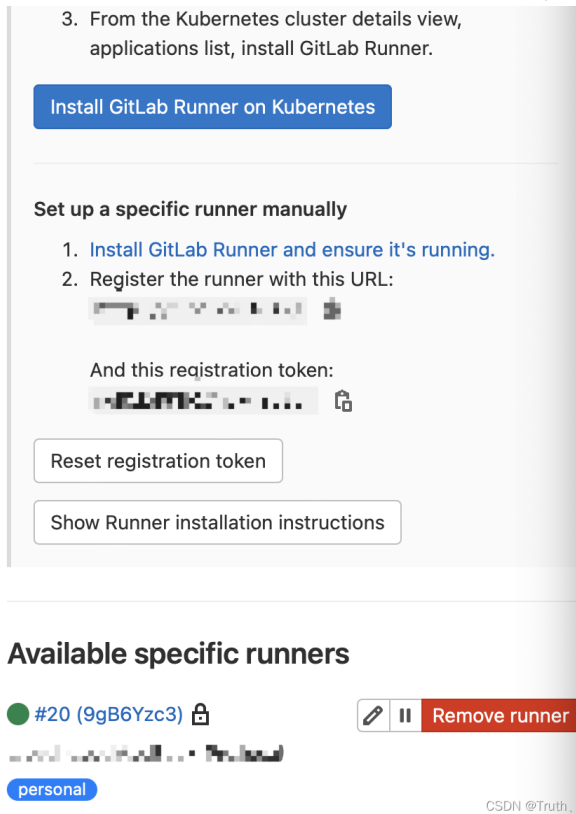
2.2.3 部署后端测试报告

3.部署成果

这里以部署主服务器的前后端为例

1.部署前端项目

1. 首先我们使用电脑本机下载安装gitlab runner，这里将本机称为服务器A。(电脑本机为mac环境，不同操作系统下面的命令可能会有差别。)
2. 使用 `gitlab-runner register` 命令在gitlab网站上注册gitlab runner，填入settings->CI/CD->Runners下的URL和token。结果如如下页面所示即为成功，这里填写的标签是**personal**，方便后续配置一一对应



3. 接下去需要一台远程服务器B，我们使用的是腾讯云的ubuntu服务器。腾讯云的ubuntu服务器默认为ubuntu账户，我们需要对其进行配置，以切换为root账户。
 - 使用用户名ubuntu登录后使用 `sudo passwd root` 命令，然后输入新的root密码并进行确认
 - 修改ssh配置 `sudo vi /etc/ssh/sshd_config`，将文件中的PermitRootLogin修改为yes
 - 重启ssh服务 `sudo service ssh restart`
4. 配置服务器A和B之间的免密通道，即本机和远程服务器之间的免密通道。
 - 使用 `ssh-keygen -t rsa -b 2048 -C "<comment>"` 命令，引号内可以填写邮箱地址。
 - 执行后推荐使用默认的地址和名称来存储公钥和私钥（默认存储在~/.ssh/id_rsa和

~/ssh/id_rsa.pub下)

- 不要在配置公钥的时候为公钥设置密码，一直默认回车即可

- 使用 `scp -r id_rsa.pub root@124.222.139.8:/root/.ssh/authorized_keys` 命令将公钥上传到服务器
- 使用 `ssh root@124.222.139.8` 来验证能否免密登录远程服务器B，若不需要输入密码即为成功。

5. 在gitlab的settings->CI/CD->Variables下新增一个 `SSH_PRIVATE_KEY` 变量，用来存储私钥。将本机中 `~/ssh/id_rsa` 下的内容复制为该变量的值。后续编写.gitlab-ci.yml需要用到。
6. 在前端项目目录下新增一个.gitlab-ci.yml文件，添加以下内容，用来下载依赖、编译和部署

```
image: node:latest

# 部署分为三个阶段install（下载依赖、构建项目、部署到服务器上）
stages:
  - install
  - build
  - deploy

cache:
  key:
    files:
      - package-lock.json
  paths:
    - node_modules

job_install:
  stage: install
  tags:
    # 该阶段使用的gitlab runner
    - personal
  before_script:
    # 切换依赖下载地址
    - npm config set registry https://registry.npm.taobao.org
  script:
    # 下载依赖
    - npm install

job_build:
  stage: build
  tags:
    # 该阶段使用的gitlab runner
    - personal
  script:
    - npm run build:prod
  artifacts:
    paths:
      - dist
```

```

job_deploy:
  stage: deploy
  tags:
    - personal
  before_script:
    - 'which ssh-agent || ( yum update -y && yum install openssh-client git -y
) '

    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan 124.222.139.8 >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
  script:
    # 将dist目录拷贝到服务器的/usr/local/www目录下(若服务器下不存在该目录, 可以在服务器上
    手动创建, 也可以使用2.2.2节中提到的使用test.sh的方法进行自动创建)
    - scp -r dist root@124.222.139.8:/usr/local/www







```

7. 代码提交修改到gitlab后就会自动触发部署, 部署成功后结果如下

pipelines. [Learn how to speed up your pipeline with needs.](#)

[Provide feedback](#)

Pipeline Needs Jobs 3 Tests 0

Install	Build	Deploy
 job_install 	 job_build 	 job_deploy 

CSDN @Truth,

8. 连接远程服务器, 以root身份登入, 使用 `apt install docker.io` 命令在远程服务器中安装docker

9. 安装完成后 `docker run -p 8080:80 -d -v /usr/local/www/dist:/usr/share/nginx/html nginx` 为项目配置运行的端口

10. 完成后访问 `124.222.139.8:8080` 就可以看到项目啦!

2.部署后端项目

2.1 整体步骤

1. 在使用 `gitlab-runner register` 命令在gitlab网站上注册gitlab runner, **gitlab runner**的tag写为 **backend**, 方便后续配置一一对应
2. 在项目根目录下添加.gitlab-ci.yml配置文件 (详细配置见2.2.1)
3. 在项目根目录下添加test.sh文件, 内容如下:

```
#!/bin/bash
```

```

# 判断传入的参数个数是不是一个
if [ ! $# -eq 1 ];then
    echo param error!
    exit 1
fi

# 判断目录是不是已经存在，如果不存在则创建，存在则输出“dir exist”
dirname=$1
echo "the dir name is $dirname"
if [ ! -d $dirname ];then
    mkdir $dirname
else
    echo dir exist
fi

```

4. 完成如上配置后每次向gitlab提交代码就会自动部署

2.2.1 后端gitlab配置文件.gitlab-ci.yml

在项目根目录下添加.gitlab-ci.yml文件

完整后端的gitlab配置如下：详见注解

```

# 定义一些变量，下面各阶段会使用
variables:
    server_ip: 124.222.139.8
    jar_name: backend-crowdsourcedtesting-0.0.1-SNAPSHOT.jar
    java_path: /usr/local/java1.8/bin
    upload_path: /usr/local/gitlab-project
    test_path: /usr/local/testHtml
    jacoco_path: /usr/local/jacocoHtml

# 定义执行的各个阶段及顺序
stages:
    - build
    - test
    - upload
    - deploy

# 使用 maven 镜像打包项目
maven-build:
    stage: build
    image: maven:3-jdk-8
    script:
        - mvn package -B -Dmaven.test.skip=true
    cache:

```

```

key: m2-repo
paths:
  - .m2/repository
artifacts:
  paths:
    # 将生成的jar包存到target目录下
    - target/$jar_name
tags:
  # 该阶段使用的gitlab runner
  - backend

# 测试并生成测试报告和测试覆盖率报告
test:
  stage: test
  image: maven:3-jdk-8
  tags:
    # 该阶段使用的gitlab runner
    - backend
  before_script:
    - 'which ssh-agent || ( yum update -y && yum install openssh-client git -y ) '
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan 124.222.139.8 >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
  script:
    # 生成测试报告, 会在target/site/目录下生成测试报告surefire-report.html
    - mvn surefire-report:report
    - ls -l target/site/
    # 将用来在远程服务器上自动创建目录的test.sh上传到服务器上的/usr/local目录下
    - scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no test.sh
root@$server_ip:/usr/local
    # 在服务器上执行命令, 运行test.sh脚本创建/usr/local/testHtml目录
    - ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@$server_ip
"sh /usr/local/test.sh /usr/local/testHtml"
    # 将生成的测试报告放到/usr/local/testHtml/index.html目录下
    - scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
target/site/surefire-report.html root@$server_ip:$test_path/index.html
    # 生成测试覆盖率报告, 会在target/site/jacoco目录下生成测试报告index.html
    - mvn jacoco:report
    - ls -l target/site/
    # 在服务器上执行命令, 运行test.sh脚本创建/usr/local/jacocoHtml目录
    - ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@$server_ip
"sh /usr/local/test.sh /usr/local/jacocoHtml"
    # 将生成的测试覆盖率报告放到/usr/local/jacocoHtml/index.html目录下
    - scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
target/site/jacoco/index.html root@$server_ip:$jacoco_path/index.html

```

上传生成的 jar 包到你的应用服务器

upload-jar:

stage: upload

tags:

该阶段使用的gitlab runner

- backend

before_script:

- 'which ssh-agent || (yum update -y && yum install openssh-client git -y) '

- eval \$(ssh-agent -s)

- echo "\$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -

- mkdir -p ~/.ssh

- chmod 700 ~/.ssh

- ssh-keyscan 124.222.139.8 >> ~/.ssh/known_hosts

- chmod 644 ~/.ssh/known_hosts

script:

- ls -l target/

在服务器上运行test.sh脚本自动创建/usr/local/gitlab-project目录

- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@\$server_ip

"sh /usr/local/test.sh /usr/local/gitlab-project"

将生成的jar包放到服务器的/usr/local/gitlab-project目录下

- scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no target/\$jar_name

root@\$server_ip:\$upload_path/\$jar_name

启动 SpringBoot jar包

deploy-test:

stage: deploy

tags:

该阶段使用的gitlab runner

- backend

before_script:

- 'which ssh-agent || (yum update -y && yum install openssh-client git -y) '

- eval \$(ssh-agent -s)

- echo "\$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -

- mkdir -p ~/.ssh

- chmod 700 ~/.ssh

- ssh-keyscan 124.222.139.8 >> ~/.ssh/known_hosts

- chmod 644 ~/.ssh/known_hosts

script:

自动重启springboot的jar包

自动杀死8082端口对应的进程

- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@\$server_ip

"netstat -tunlp|grep 8082|awk '{print \\$7}'|cut -d '/' -f 1|xargs test -z || netstat -tunlp|grep 8081|awk '{print \\$7}'|cut -d '/' -f 1|xargs kill"

after_script:

在服务端重启jar包

- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@\$server_ip

"nohup java -jar \$upload_path/\$jar_name --server.port=8082 >/dev/null 2>&1 &"

2.2.2 部署jar包

对后端项目进行部署，将后端打包生成的jar包部署到服务器的8082端口。

部署jar包时的一些说明：

- `SSH_PRIVATE_KEY` 与前端项目部署时创建变量一样
- `"nohup java -jar $upload_path/$jar_name --server.port=8082 >/dev/null 2>&1 &"` 是上传jar包成功后在服务器端执行的命令，该命令可以启动jar包对应的springboot项目且保持不关闭
- 需要提前在远程服务器的/usr/local目录下创建gitlab-project目录
 - 若要使用代码在远程服务器上自动创建gitlab-project目录，首先需要有一个shell脚本test.sh，并放在项目根目录下

```
#!/bin/bash
# 判断传入的参数个数是不是一个
if [ ! $# -eq 1 ];then
    echo param error!
    exit 1
fi

# 判断目录是不是已经存在，如果不存在则创建，存在则输出“dir exist”
dirname=$1
echo "the dir name is $dirname"
if [ ! -d $dirname ];then
    mkdir $dirname
else
    echo dir exist
fi
```

- gitlab-ci.yml代码如下：

```
// 将test.sh脚本发到服务器上
- scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no test.sh
root@$server_ip:/usr/local
// 执行test.sh脚本创建目标目录
- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
root@$server_ip "sh /usr/local/test.sh /usr/local/gitlab-project"
- scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
target/$jar_name root@$server_ip:$upload_path/$jar_name
```

- 变量中的 `jar_name` 的值需要对应的改成自己的项目打成jar包后的名字，否则会找不到jar包的位置。一般命名为pom.xml下的artifactId标签的xxx加上version标签的内容。如这儿就是 `backend-crowdsourcetesting-0.0.1-SNAPSHOT.jar`

```
<artifactId>backend-crowdsourcetesting</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

- 后端需要自动杀死进程并重启jar包，请在.gitlab-ci.yml使用以下命令：

```
// 自动杀死8082端口对应的进程
- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@$server_ip
  "netstat -tunlp|grep 8082|awk '{print \$7}'|cut -d '/' -f 1|xargs test -z || netstat -
  tunlp|grep 8082|awk '{print \$7}'|cut -d '/' -f 1|xargs kill"
// 重启服务
- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@$server_ip
  "nohup java -jar $upload_path/$jar_name --server.port=8082 >/dev/null 2>&1 &"
```

2.2.3 部署后端测试报告

这部分我们对后端项目进行测试，并产生 html 形式的测试报告，部署到服务器的8081端口。

1. 在项目的pom.xml文件中加入如下配置代码

测试报告plugin：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-report-plugin</artifactId>
  <configuration>
    <showSuccess>>false</showSuccess>
  </configuration>
</plugin>
```

测试覆盖率报告plugin：

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.7.7.201606060606</version>
  <configuration>
    <destFile>target/coverage-reports/jacoco-unit.exec</destFile>
    <dataFile>target/coverage-reports/jacoco-unit.exec</dataFile>
  </configuration>
  <executions>
    <execution>
      <id>jacoco-initialize</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>jacoco-site</id>
      <phase>package</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```



```
    </execution>
  </executions>
</plugin>
```

2. 这时候我们在项目目录下运行 `mvn surefire-report:report` 命令就会在项目的target/site目录下生成surefire-report.html文件，即是我们需要的测试报告文件。
3. 在项目目录下运行 `mvn clean test jacoco:report` 命令就会在项目的target/site/jacoco目录下生成index.html文件，即我们需要的测试覆盖率报告文件。
4. 接下去就是将surefire-report.html和index.html部署到服务器上
5. 新增一个测试阶段：

```
stages:
  - build
  - test
  - upload
  - deploy
```

6. 测试阶段的配置代码如下。首先运行 `mvn surefire-report:report` 命令在target/site/目录下生成测试报告surefire-report.html，将它发送到服务器上的 `$test_path` 路径下（注意需要将名字改为index.html，否则使用nginx部署后会出现403 Forbidden）

```
test:
  stage: test
  image: maven:3-jdk-8
  tags:
    - backend
  before_script:
    - 'which ssh-agent || ( yum update -y && yum install openssh-client git -y ) '
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan 124.222.139.8 >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
  script:
    # 生成测试报告，会在target/site/目录下生成测试报告surefire-report.html
    - mvn surefire-report:report
    - ls -l target/site/
    # 将用来在远程服务器上自动创建目录的test.sh上传到服务器上的/usr/local目录下
    - scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no test.sh
    root@$server_ip:/usr/local
    # 在服务器上执行命令，运行test.sh脚本创建/usr/local/testHtml目录
    - ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
    root@$server_ip "sh /usr/local/test.sh /usr/local/testHtml"
    # 将生成的测试报告放到/usr/local/testHtml/index.html目录下
```

```
- scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
target/site/surefire-report.html root@$server_ip:$test_path/index.html
# 生成测试覆盖率报告，会在target/site/jacoco目录下生成测试报告index.html
- mvn jacoco:report
- ls -l target/site/
# 在服务器上执行命令，运行test.sh脚本创建/usr/local/jacocoHtml目录
- ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
root@$server_ip "sh /usr/local/test.sh /usr/local/jacocoHtml"
# 将生成的测试覆盖率报告放到/usr/local/jacocoHtml/index.html目录下
- scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
target/site/jacoco/index.html root@$server_ip:$jacoco_path/index.html
```

7. 在服务器上运行 `docker run -p 8081:80 -d -v $test_path:/usr/share/nginx/html nginx` 命令即可将测试报告对应的页面挂载到服务器的8081端口。（注意test_path对应的是一个目录，而不是目录下的index.html文件）
8. 在服务器上运行 `docker run -p 8086:80 -d -v $jacoco_path:/usr/share/nginx/html nginx` 命令即可将测试覆盖率报告对应的页面挂载到服务器的8086端口。（注意jacoco_path对应的是一个目录，而不是目录下的index.html文件）

3.部署成果

主分支

- 主分支前端部署地址：124.222.139.8:8080
- 主分支测试报告生成地址：124.222.139.8:8081
- 主分支后端部署地址：124.222.139.8:8082
- 主分支测试覆盖率报告部署地址：124.222.139.8:8086

测试分支

- 测试分支后端部署地址；124.222.139.8:8083
- 测试分支前端部署地址；124.222.139.8:8084
- 测试支测试报告生成地址：124.222.139.8:8085
- 测试分支测试覆盖率报告部署地址：124.222.139.8:8087