### **Gitlab-runner+gitlab+idea+ubuntu16.04实现持续集成**

——陈亮

目录

[1.环境准备 1](#_Toc17220)

[2.安装和注册gitlab-runner 3](#_Toc18342)

参考自:https://www.funtl.com , https://www.funtl.com/zh/apache-dubbo-ci

**1.环境准备**

首先得在ubuntu16.04上安装好docker,docker-compose,registry私有镜像仓库及客户端,nexus私有仓库,

**vi /etc/docker/daemon.json** 编辑设置好镜像加速地址和私有镜像仓库

1)创建工作目录  **/usr/local/docker/runner**

2)创建构建目录  **/usr/local/docker/runner/environment**

3)下载**jdk-8u152-linux-x64.tar.gz**并复制到 **/usr/local/docker/runner/environment**

4)下载**apache-maven-3.5.3-bin.tar.gz**并复制到**/usr/local/docker/runner/environment**

5)拷贝**Dockerfile**,**settings.xml,docker-compose** 到**/usr/local/docker/runner/environment**(Dockerfile是步骤2中创建的,不是idea项目下的)

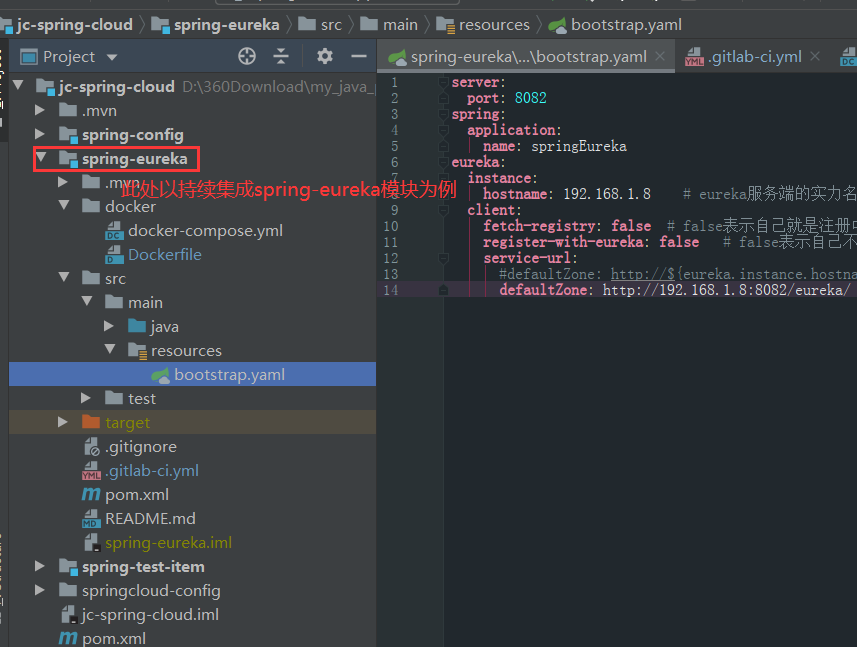
6)拷贝**docker-compose.yml** 到 **/usr/local/docker/runner** (docker-compose.yml是步骤2中创建的,不是idea项目下的)

7)Idea中创建好测试项目并上传到gitlab(用官方的**https://git.lug.ustc.edu.cn**也可以自己创建私有的)

此处以持续集成spring-eureka模块为例

1. 下载dockerize放tomcat\_home路径/download下,把tomcat设置成开机启动;,用迅雷下载最新版本(凌晨7点网速快),

下载地址:**https://github.com/jwilder/dockerize/releases/download/v0.6.1/dockerize-linux-amd64-v0.6.1.tar.gz**



**a.项目中创建gitlab-ci.yml**

在项目工程下编写 .gitlab-ci.yml 配置文件：

# 定义阶段  
**stages**:  
 - build  
 - push  
 - run  
 - clean  
# 下面是每个阶段的具体工作  
# 构建镜像  
**build**:  
 **stage**: build  
 **script**:  
 - /usr/local/maven/apache-maven-3.5.3/bin/mvn clean package  
 - cp target/spring-eureka-0.0.1-SNAPSHOT.jar docker  
 - cd docker  
 - docker build -t 192.168.1.8:5000/spring-eureka:v0.0.1 .  
# 上传到docker私有镜像仓库  
**push**:  
 **stage**: push  
 **script**:  
 - docker push 192.168.1.8:5000/spring-eureka:v0.0.1  
 # 运行docker容器(启动项目)  
**run**:  
 **stage**: run  
 **script**:  
 - cd docker  
 - docker-compose down  
 - docker-compose up -d  
# 清理  
**clean**:  
 **stage**: clean  
 **script**:  
# 清除虚悬镜像  
 - docker image prune -f  
  
  
#stages:  
# - test  
#test:  
# stage: test  
# script:  
# - echo "use gitlab-runner实现持续集成test succeed!陈亮hello world!!!"

**b.项目中的Dockerfile**

**模块项目下创建docker文件夹,把Dockerfile放入其中**

**FROM** openjdk:8-jre  
  
**MAINTAINER** Lusifer **<**topsale@vip.qq.com**>  
  
ENV** APP\_VERSION 0.0.1**-**SNAPSHOT  
**ENV** DOCKERIZE\_VERSION v0.6.1  
#如果不依赖其它服务不用dockerize  
#RUN wget https://github.com/jwilder/dockerize/releases/download/$DOCKERIZE\_VERSION/dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz \  
 #https://github.com/jwilder/dockerize/releases/download/v0.6.1/dockerize-linux-amd64-v0.6.1.tar.gz   
 #&& tar -C /usr/local/bin -xzvf dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz \  
 #&& rm dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz  
  
**RUN** mkdir **/**app  
  
**COPY** spring-eureka-$APP\_VERSION.jar **/**app**/**app.jar  
#ENTRYPOINT ["dockerize", "-timeout", "5m", "-wait", "tcp://192.168.10.131:3306", "java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app/app.jar"]  
**ENTRYPOINT** ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app/app.jar"]  
**EXPOSE** 8082

**c.项目中的docker-compose.yml**

把docker-compose.yml放入docker文件夹中

**version**: '3.1'  
**services**:  
 **spring-eureka**:  
 **image**: 192.168.1.8:5000/spring-eureka:v0.0.1  
 **container\_name**: spring-eureka  
 **ports**:  
 - 8082:8082  
**networks**:  
 **default**:  
 **external**:  
 **name**: eureka

**d.项目中的pom.xml文件中加入maven的私有仓库nexus:最后注释包含1和2的地方就是私服配置;同时maven的setting.xml文件中也要nexus私服的相关配置,请看资料中的setting.xml文件(注释有数字的地方就是)。**

*<?*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**>2.2.2.RELEASE</**version**>  
 <**relativePath**/> <!-- lookup parent from repository -->  
 </**parent**>  
 <**groupId**>com.example.test</**groupId**>  
 <**artifactId**>spring-test-item</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>spring-test-item</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**properties**>  
 <**java.version**>1.8</**java.version**>  
 <**spring-cloud.version**>Hoxton.RELEASE</**spring-cloud.version**>  
 </**properties**>  
  
 <**dependencies**>  
 <!--thymeleaf前端模板-->  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  
 </**dependency**>  
 <!--web 这个依赖要加不然回报Completed shut down of DiscoveryClient,eureka也发现不了该服务-->  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <!--config server center-->  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-config</**artifactId**>  
 </**dependency**>  
 <!--eureka client-->  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
 </**dependency**>  
 <!--test-->  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.junit.vintage</**groupId**>  
 <**artifactId**>junit-vintage-engine</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
 </**dependencies**>  
  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>${spring-cloud.version}</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
 <!--1.maven nexus私有仓库-->  
 <**repositories**>  
 <**repository**>  
 <**id**>maven-central</**id**>  
 <**name**>maven-central</**name**>  
 <**url**>http://192.168.1.8:8081/repository/maven-central/</**url**>  
 <**snapshots**>  
 <**enabled**>true</**enabled**>  
 </**snapshots**>  
 <**releases**>  
 <**enabled**>true</**enabled**>  
 </**releases**>  
 </**repository**>  
 <**repository**>  
 <**id**>nexus</**id**>  
 <!--<name>Nexus Releases</name>-->  
 <**name**>maven-releases</**name**>  
 <**url**>http://192.168.1.8:8081/repository/maven-releases/</**url**>  
 </**repository**>  
 </**repositories**>  
 <!--2.mavne nexus私有仓库中的插架管理-->  
 <**distributionManagement**>  
 <**snapshotRepository**>  
 <**id**>nexus</**id**>  
 <!--<name>Nexus Snapshot</name>-->  
 <**name**>maven-snapshots</**name**>  
 <**url**>http://192.168.1.8:8081/repository/maven-snapshots/</**url**>  
 </**snapshotRepository**>  
 <**site**>  
 <**id**>nexus</**id**>  
 <**name**>Nexus Sites</**name**>  
 <**url**>dav:http://192.168.1.8:8081/repository/maven-snapshots/</**url**>  
 </**site**>  
 </**distributionManagement**>  
</**project**>

**2.安装和注册gitlab-runner**

**1)创建daemon.json**

在 /usr/local/docker/runner/environment 目录下创建 daemon.json，用于配置加速器和仓库地址

{

"注释":"docker镜像加速地址",

"registry-mirrors": ["https://vt6z4y79.mirror.aliyuncs.com","https://docker.mirrors.ustc.edu.cn"

,"http://hub-mirror.c.163.com","http://harbor.test.com"],

"注释":"自己的私有镜像仓库",

"insecure-registries":["http://192.168.1.8:5000"],

"max-concurrent-downloads": 10

}

在 /usr/local/docker/runner/environment 目录下创建 Dockerfile

**2)创建Dockerfile**

#不要出现交互的命令:比如yes/no,选yes安装;所以要跟新哪个配置文件要先删除再拷贝(构建)

#gitlab-runner版本要指定,最新版运行不起来,因为使用方式不一样

FROM gitlab/gitlab-runner:v11.0.2

MAINTAINER chenliang <@1484157619.qq.com>

# 修改软件源

RUN echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted universe multiverse' >> /etc/apt/sources.list && \

echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted universe multiverse' >> /etc/apt/sources.list && \

echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted universe multiverse' >> /etc/apt/sources.list && \

echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe multiverse' >> /etc/apt/sources.list && \

apt-get update -y && \

apt-get clean

# 安装 Docker

#RUN curl -fsSL get.docker.com -o get-docker.sh

# 可能会出现 404 错误，请移步下面的特别说明

#RUN sh get-docker.sh --mirror Aliyun

#RUN ARG DEBIAN\_FRONTEND=noninteractive && apt-get update -y && apt-get install -y gnupg2 --assume-yes apt-utils && \

#RUN apt-get update -y && apt-get upgrade -y && apt-get -y install apt-utils

#RUN RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys

#RUN apt-get -y update && apt-get install -y \

#linux-image-extra-$(uname -r) && apt-get install -y linux-image-extra-virtual

RUN apt-get install -y apt-transport-https ca-certificates curl software-properties-common && \

curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | apt-key add - && \

#curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add - && \

add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb\_release -cs) stable" && \

#add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs)stable"

apt-get update -y && \

apt-get install -y docker-ce

#WORKDIR /root/install-package

#WORKDIR ~/install-package

COPY daemon.json /etc/docker/daemon.json

# 安装 Docker Compose

COPY docker-compose /usr/local/bin/docker-compose

#RUN wget https://raw.githubusercontent.com/topsale/resources/master/docker/docker-compose

RUN chmod +x /usr/local/bin/docker-compose

# copy配置文件、包和安装Java,拷贝包或文件到指定目录下,要指明拷贝成具体包或文件

COPY jdk-8u152-linux-x64.tar.gz /usr/local/java/jdk-8u152-linux-x64.tar.gz

COPY apache-maven-3.5.3-bin.tar.gz /usr/local/maven/apache-maven-3.5.3-bin.tar.gz

#COPY settings.xml /usr/local/maven/apache-maven-3.5.3/conf/settings.xml #apache-maven-3.5.3-bin.tar.gz解压后才能执行

#COPY docker-compose.yml /usr/local/docker/runner/docker-compose.yml

#COPY Dockerfile /usr/local/docker/runner/environment/Dockerfile

#RUN mkdir -p /usr/local/java #创建文件不往Dockerfile里面写,如果文件存在不能交互,会报错

WORKDIR /usr/local/java

#COPY jdk-8u152-linux-x64.tar.gz /usr/local/java

RUN tar -zxvf jdk-8u152-linux-x64.tar.gz && \

rm -fr jdk-8u152-linux-x64.tar.gz

# 安装 Maven

#RUN mkdir -p /usr/local/maven

WORKDIR /usr/local/maven

# RUN wget https://raw.githubusercontent.com/topsale/resources/master/maven/apache-maven-3.5.3-bin.tar.gz

RUN tar -zxvf apache-maven-3.5.3-bin.tar.gz && \

rm -fr apache-maven-3.5.3-bin.tar.gz

#拷贝文件到指定目录下,要指明拷贝成具体文件

COPY settings.xml /usr/local/maven/apache-maven-3.5.3/conf/settings.xml

# 配置环境变量

#jdk-8u152-linux-x64.tar.gz解压后包名变为jdk1.8.0\_171，要特别注意，可以先解压看一下

ENV JAVA\_HOME /usr/local/java/jdk1.8.0\_171

ENV MAVEN\_HOME /usr/local/maven/apache-maven-3.5.3

ENV PATH $PATH:$JAVA\_HOME/bin:$MAVEN\_HOME/bin

WORKDIR /

**3)创建docker-compose.yml**

在 /usr/local/docker/runner 目录下创建 docker-compose.yml

version: '3.1'

services:

gitlab-runner:

build: environment

restart: always

container\_name: gitlab-runner

privileged: true

volumes:

- ./config:/etc/gitlab-runner

- /var/run/docker.sock:/var/run/docker.sock

**4)启动和注册 Runner**

**a.启动:** cd /usr/local/docker/runner

docker-compose build

docker-compose up

提示:Dockerfile构建gitlab-runner镜像时:构建失败在所难免,失败的镜像不要删,继续修改运行直到成功(因为在构建失败镜像的基础上构建会少执行很多步骤,比如有些镜像已拉取,就节约了时间;如果构建的项目镜像损坏或删除了,必须打开注释掉的构建步骤,全部执行,一般是进入容器手工调试会出现这种情况),构建成功后才把失败的镜像、虚悬镜像和不要的容器删除,如果太多可以只留下gitlab-runner镜像和有用的镜像,其它全部删除，再构建一遍(docker-compose down , docker-compose up)。

**b.注册:**

docker exec -it gitlab-runner gitlab-runner register

# 输入 GitLab 地址

Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):

https://git.lug.ustc.edu.cn/

# 输入 GitLab Token

Please enter the gitlab-ci token for this runner:

ymYH4vstY1tA-2KhgZ-V

# 输入 Runner 的说明

Please enter the gitlab-ci description for this runner:

可以为空

# 设置 Tag，可以用于指定在构建规定的 tag 时触发 ci

Please enter the gitlab-ci tags for this runner (comma separated):

deploy

# 这里选择 true ，可以用于代码上传后直接执行

Whether to run untagged builds [true/false]:

true

# 这里选择 false，可以直接回车，默认为 false

Whether to lock Runner to current project [true/false]:

false

# 选择 runner 执行器，这里我们选择的是 shell

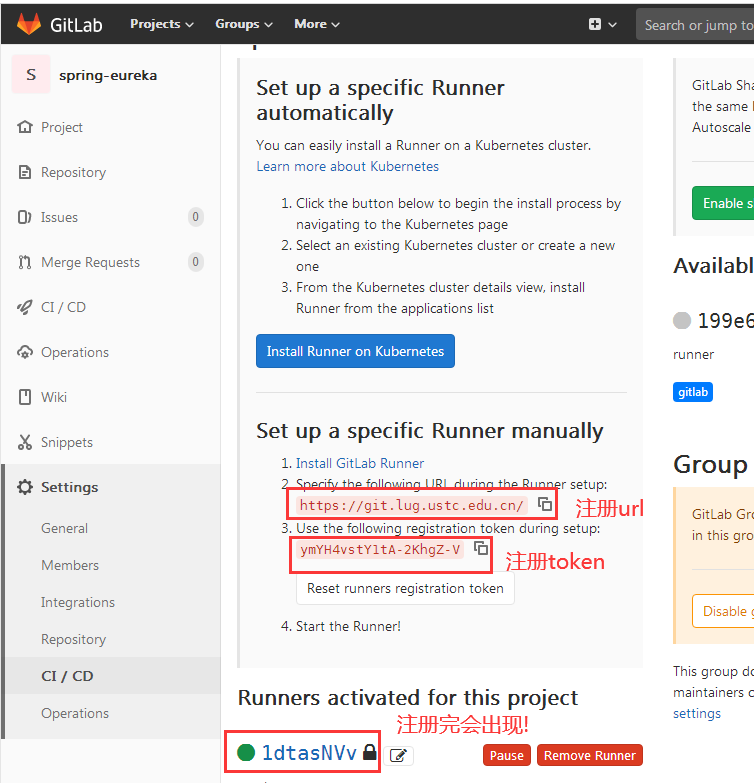
Please enter the executor: virtualbox, docker+machine, parallels, shell, ssh, docker-ssh+machine, kubernetes, docker, docker-ssh:

shell

**5)使用 Runner**

GitLab CI 地址与令牌参数

项目 –> 设置 –> CI/CD –> Runner 设置



安装依赖(install\_deps)

运行测试(test)

编译(build)

部署测试服务器(deploy\_test)

部署生产服务器(deploy\_production)

注意： 设置 Job.only 后，只有当 develop 分支和 master 分支有提交的时候才会触发相关的 Jobs。

节点说明：

stages：定义构建阶段，这里只有一个阶段 deploy

deploy：构建阶段 deploy 的详细配置也就是任务配置

script：需要执行的 shell 脚本

only：这里的 master 指在提交到 master 时执行

tags：与注册 runner 时的 tag 匹配

[#](https://www.funtl.com/zh/apache-dubbo-ci/%E5%9F%BA%E4%BA%8E-Docker-%E5%AE%89%E8%A3%85-GitLab-Runner.html" \l "测试集成效果)测试集成效果

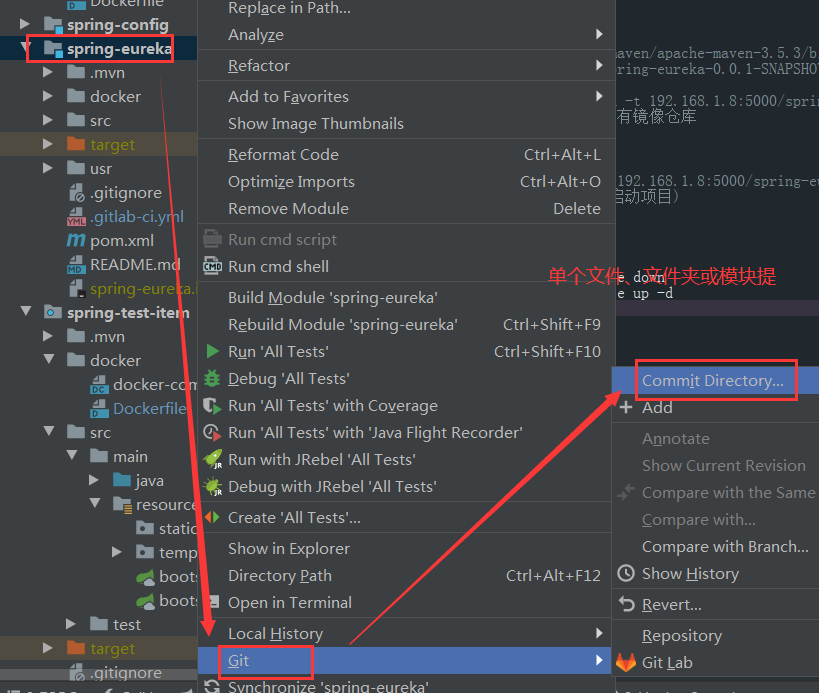
所有操作完成后 push 代码到服务器，查看是否成功：

1. **idea中测试**

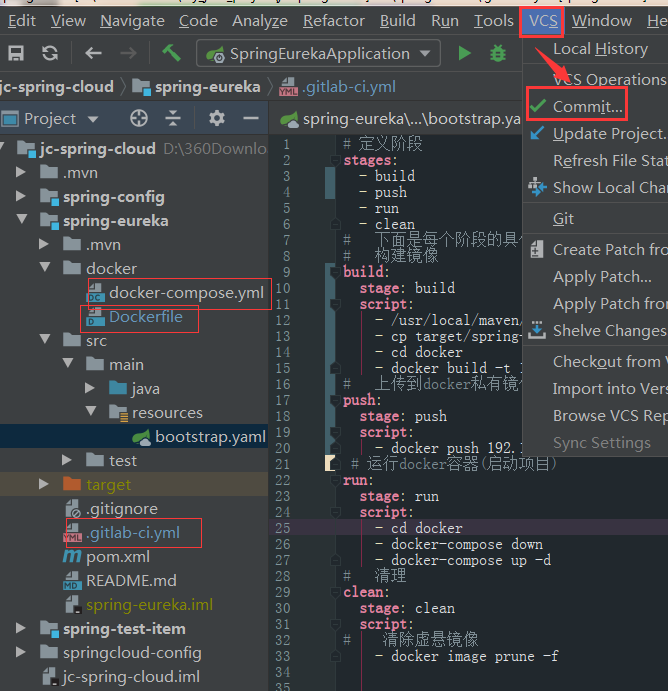
测试技巧:持续集成分很多阶段(stage),每个阶段又有一个或多个任务,阶段和任务多出现错误在所难免.查看gitlab中运行情况,把项目中的.gitlab-ci.yml对应成功的阶段注释掉，重新commit和push测试,直到所有阶段都成功;这样测试快,项目和很多依赖已经下载到gitlab-runner部署的机器了，每次测试都基于上次测试下载的。

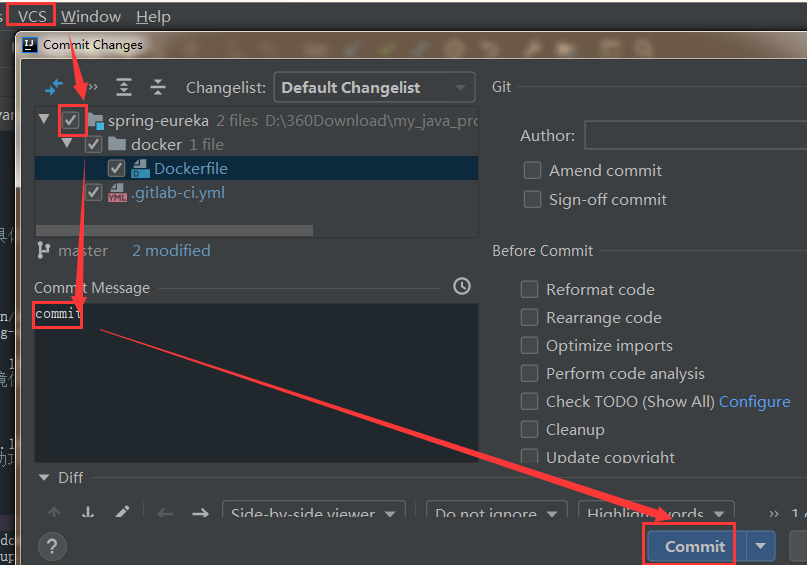
**commit提交**

**指定文件、文件夹或模块提交**

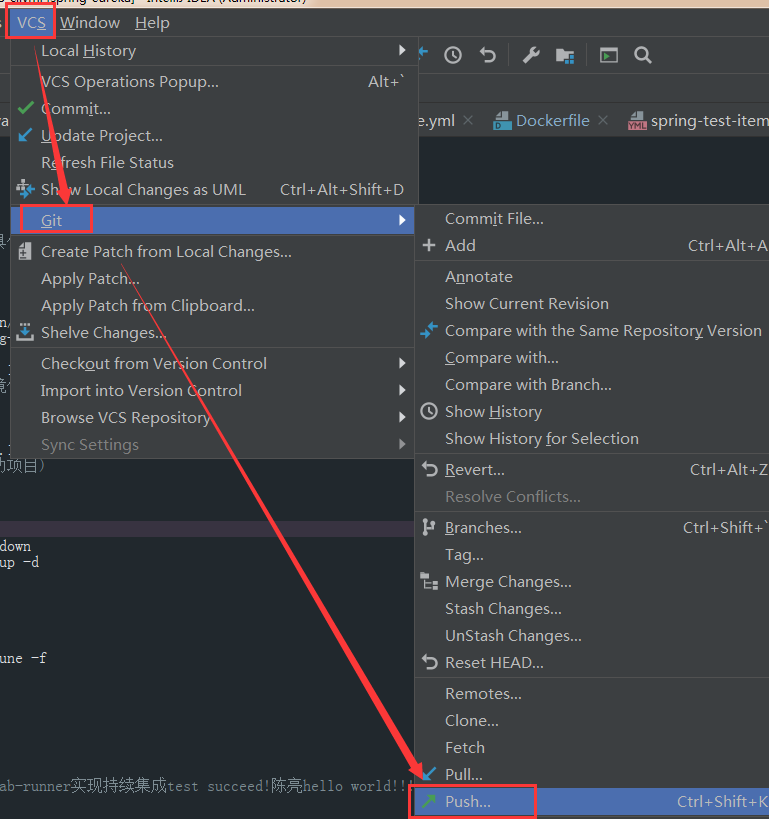
****

**提交整个项目**

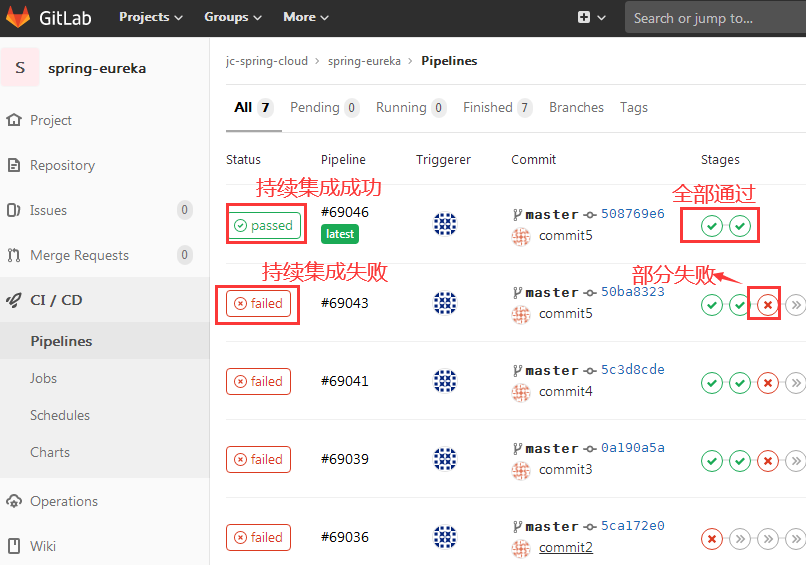
****

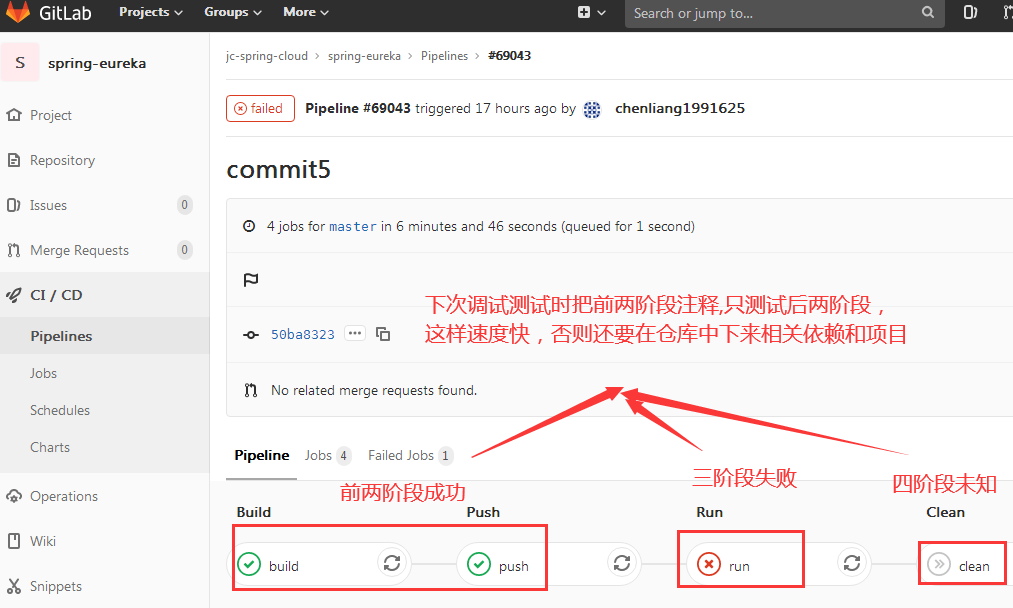
****

**push推送到gitlab**

****

**去gitlab看看情况:**

passed 表示执行成功，点开第二个红框圈起的failed查看：



**其它命令**

首先得交互式进入gitlab-runner容器,手动调试用

docker exec -it gitlab-runner容器id bash

删除注册信息:

gitlab-ci-multi-runner unregister --name "名称"

查看注册列表:

gitlab-ci-multi-runner list

**故障排查:**

Idea中提交和push后,gitlab中CI/CD没反应,之前都可以运行的

原因:项目继续集成部署到测试服务器,还运行着呢,把它停止(有虚悬镜像也要删除)

查看运行(项目)容器: docker ps

停止容器: docker stop 容器1 id 容器2 id

启动容器: docker start 容器1 id 容器2 id

删除指定容器: docker rm -f 容器id

查看镜像: docker images

删除虚悬镜像: docker rmi $(docker images -q -f dangling=true) 或用 docker rmi 镜像id -f 强制删除

1. **dockerize依赖等待机制的使用**

已经在1.环境准备阶段把dockerize下载放tomcat中了,地址:tomcat home路径/download/dockerize;比如(如果在本机):

localhost:8080/download/dockerize-linux-amd64-v0.6.1.tar.gz

源下载地址: <https://github.com/jwilder/dockerize/releases> 使用参考: https://www.codercto.com/a/74474.html

概念:比如消费者服务模块要等待提供者模块启动,如果先启动消费者也不会报错;比如所有模块等待配置中心先启动；

相互依赖的模块设置到同一局域网(在项目的docker-compose.yml中设置,如下:)。

**version**: '3.1'  
**services**:  
 **spring-eureka**:  
 **image**: 192.168.1.8:5000/spring-eureka:v0.0.1  
 **container\_name**: spring-eureka  
 **ports**:  
 - 8082:8082  
**networks**:  
 **default**:  
 **external**:  
 **name**: config-net

dockerize在项目下的Dockerfile中构建镜像时下载使用,如下:

**FROM** openjdk:8-jre  
**MAINTAINER** Lusifer **<**topsale@vip.qq.com**>  
ENV** APP\_VERSION 0.0.1**-**SNAPSHOT

#v0.6.1不好下载,用迅雷下载(凌晨7点左右网速快)  
**ENV** DOCKERIZE\_VERSION v0.6.1  
#如果不依赖其它服务不用dockerize,很有可能下载不下载不下来,那就手动  
#RUN wget https://github.com/jwilder/dockerize/releases/download/$DOCKERIZE\_VERSION/dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz \ #https://github.com/jwilder/dockerize/releases/download/v0.6.1/dockerize-linux-amd64-v0.6.1.tar.gz  
#下载下来放到tomcat中  
**RUN** wget http:**//**192.168.1.8:8089**/**download**/**dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz \  
 **&&** tar **-**C **/**usr**/**local**/**bin **-**xzvf dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz \  
 **&&** rm dockerize-linux-amd64-$DOCKERIZE\_VERSION.tar.gz  
**RUN** mkdir **/**app  
**COPY** spring-test-item-$APP\_VERSION.jar **/**app**/**app.jar  
#等待mysql数据库和配置中心spring-config启动加载了配置文件(http://192.168.1.8:8083/spring-test-item.yml该模块配置文件在配置中心的位置),再启动自己(spring-test-item)—这就是dockerize的作用  
**ENTRYPOINT**["dockerize","-timeout","5m","-wait","tcp://192.168.10.131:3306","-wait","http://192.168.1.8:8083/spring-test-item.yml","java","-Djava.security.egd=file:/dev/./urandom","-jar","/app/app.jar"]

#下两行v0.02版本的使用方法  
#CMD dockerize -wait <http://192.168.1.8:8083/spring-test-item.yml> -timeout 5m  
#ENTRYPOINT["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app/app.jar","--spring.profiles.active=dev"]  
#暴露端口(从配置文件bootstrap.yaml中找:所有依赖服务的端口也要暴露):本模块8084,8083是本模块依赖spring-config的端口(d在ocker-compose.yml文件中也要暴露)  
**EXPOSE** 8084 8083