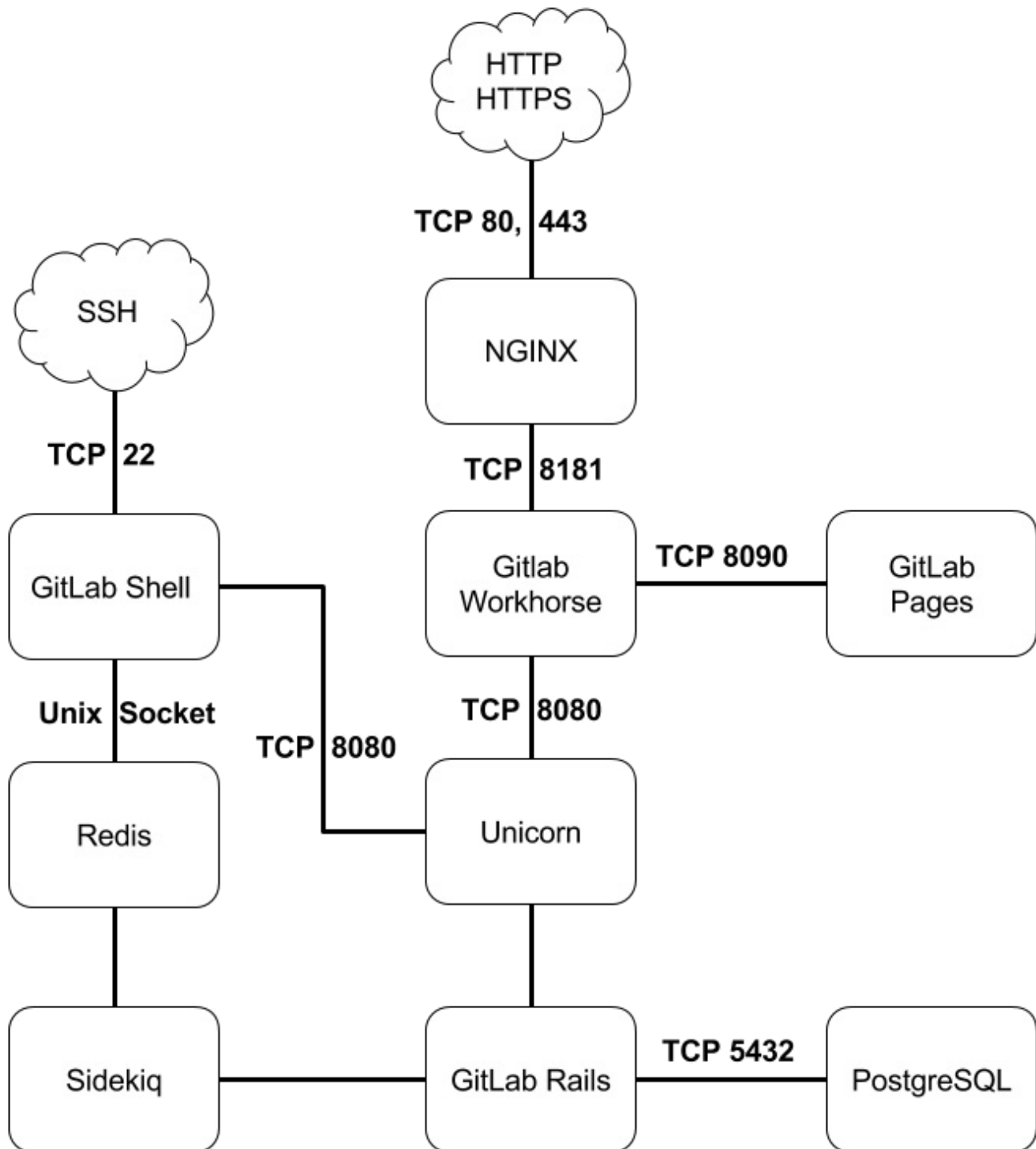


centos7安装配置gitlab

1. 简介

GitLab 是一个用于仓库管理系统的开源项目，相当于github的开源实现

gitlab架构图



2. 安装gitlab

本次使用的[安装文档](#)

1. 安装配置基础依赖

```
yum install -y curl policycoreutils openssh-server openssh-clients
systemctl enable sshd
systemctl start sshd
yum -y install postfix
systemctl enable postfix
systemctl start postfix
# 如果已经关闭防火墙如下命令不需要执行
firewall-cmd --permanent --add-service=http
systemctl reload firewalld
```

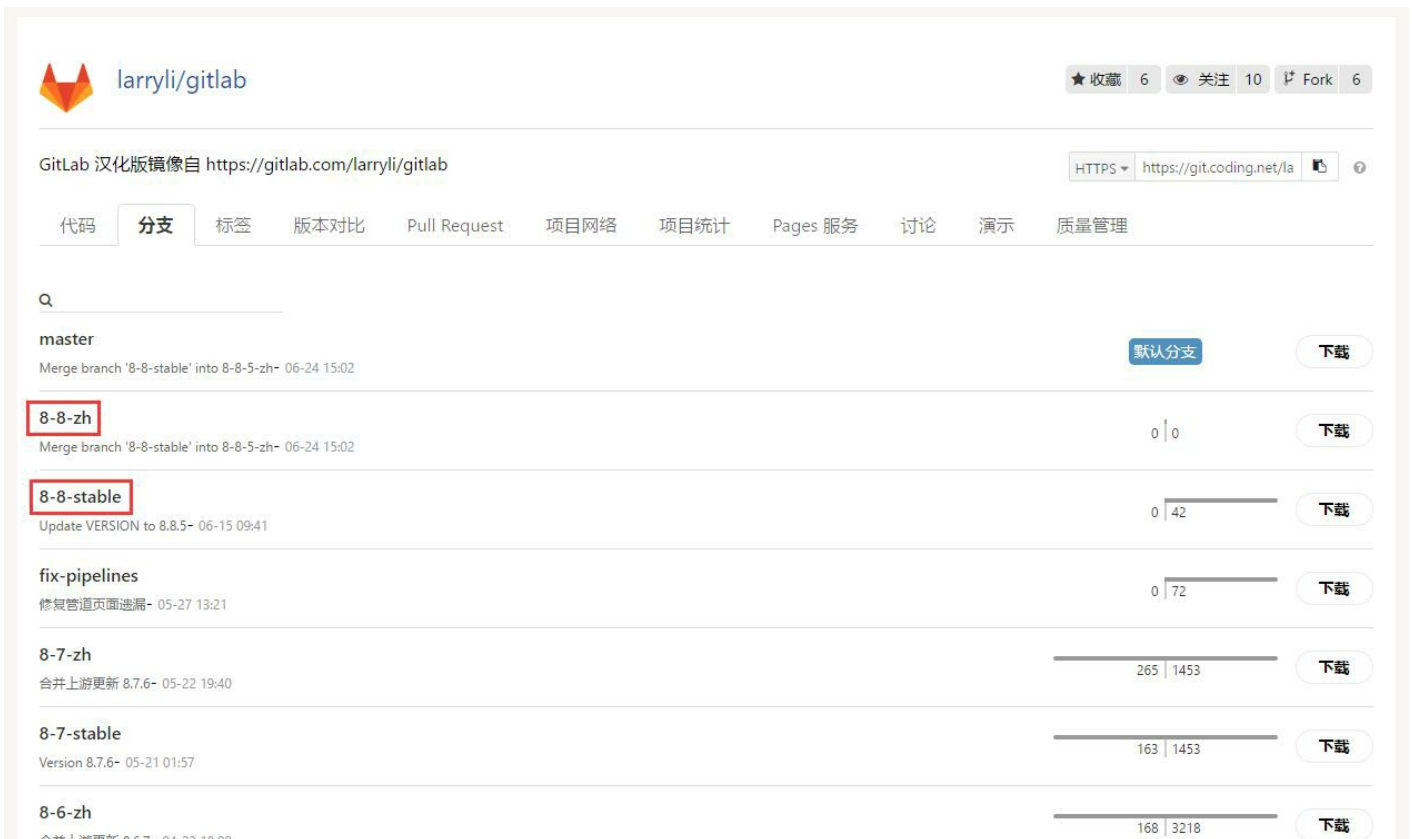
2. 安装gitlab包

```
# 使用官方源
cd /server/software
wget https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh
sh script.rpm.sh
yum install -y gitlab-ce

# 使用清华大学的源
cat >/etc/yum.repos.d/gitlab-ce.repo<<EOF
[gitlab-ce]
name=gitlab-ce
baseurl=http://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el7
repo_gpgcheck=0
gpgcheck=0
enabled=1
gpgkey=https://packages.gitlab.com/gpg.key
EOF
yum makecache
yum install gitlab-ce
```

如果需要汉化请查看最近汉化到的稳定版本，例如本次安装时使用8.8.9版本
`yum install -y gitlab-ce-8.8.9`

通过如下方法查看最新的稳定汉化版 打开如下URL <https://coding.net/u/larryli/p/gitlab/git/branches>



GitLab 汉化版镜像自 <https://gitlab.com/larryli/gitlab>

代码 分支 标签 版本对比 Pull Request 项目网络 项目统计 Pages 服务 讨论 演示 质量管理

Q

master Merge branch '8-8-stable' into 8-8-5-zh- 06-24 15:02 默认分支 下载

8-8-zh Merge branch '8-8-stable' into 8-8-5-zh- 06-24 15:02 0 | 0 下载

8-8-stable Update VERSION to 8.8.5- 06-15 09:41 0 | 42 下载

fix-pipelines 修复管道页面漏洞- 05-27 13:21 0 | 72 下载

8-7-zh 合并上游更新 8.7.6- 05-22 19:40 265 | 1453 下载

8-7-stable Version 8.7.6- 05-21 01:57 163 | 1453 下载

8-6-zh 合并上游更新 8.6.7- 05-23 18:00 168 | 3218 下载

从上面的方法获版本后再去下面的网站里获取这个大版本的小版本号
<http://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el7>

```
# 如果网络环境不好可能会出现无法下载完成的情况
# 可以手动下载包来安装
curl -LJO https://packages.gitlab.com/gitlab/gitlab-ce/packages/el/7/gitlab-ce-8.13.1-ce.0.el7.x86_64.rpm/download
yum localinstall -y gitlab-ce-8.13.1-ce.0.el7.x86_64.rpm
```

3. 配置启动gitlab

```
gitlab-ctl reconfigure
```

我在配置的时候由于已经创建了git用户并且正在登录，导致无法成功安装

4. web访问登录测试

如果第一次登录会让你设置管理员密码，设置完成后即可正常使用了 管理员用户名为 root 密码为你刚刚设置的密码 <http://192.168.12.211/>

修改gitlab默认的web访问端口号

```
1. 修改/etc/gitlab/gitlab.rb文件的如下部分
external_url 'http://192.168.12.211:8090'
....
....
nginx['listen_port'] = 8090
```

```
2. 使用配置生效
gitlab-ctl reconfigure
```

配置gitlab邮件

1. 修改/etc/gitlab/gitlab.rb文件的如下部分

```
#####
# GitLab email server settings #
#####
# see https://gitlab.com/gitlab-org/omnibus-gitlab/blob/master/doc/settings/smtp.md#smtp-settings
# Use smtp instead of sendmail/postfix.

gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.163.com"
gitlab_rails['smtp_port'] = 25
gitlab_rails['smtp_user_name'] = "will835559313@163.com"
gitlab_rails['smtp_password'] = "xxxxxxxx"
gitlab_rails['smtp_domain'] = "163.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
...
gitlab_rails['gitlab_email_from'] = 'will835559313@163.com'
...
user['git_user_email'] = "will835559313@163.com"
```

2. 使用配置生效

```
gitlab-ctl reconfigure
```

设置时区

```
# 修改配置文件 /etc/gitlab/gitlab.rb
gitlab_rails['time_zone'] = 'Asia/Shanghai'

# 使用配置生效
gitlab-ctl reconfigure
```

升级gitlab 本次升级从 8.8.9 升级到 8.13.3

官方文档

```
# 直接安装 8.13.3 版本 gitlab会自动升级处理
yum install -y gitlab-ce-8.13.3

# 汉化
gitlab-ctl stop
git clone https://gitlab.com/xhang/gitlab.git
cd gitlab
git checkout 8-13-3-zh
\cp -r * /opt/gitlab/embedded/service/gitlab-rails
gitlab-ctl start

# 8.12.8可以使用如下的汉化方式
# git diff origin/8-12-stable origin/8-12-8-zh > /tmp/8.12.diff
# cd /opt/gitlab/embedded/service/gitlab-rails
# git apply /tmp/8.12.diff
```

5. 权限管理

参考文档1

参考文档2

参考文档3

- group : 用户可以创建组, 创建组后自动变为组的owner 组有Guest、Reporter、Developer、Master、Owner 角色的用户, 群组和Github的Orgnization类似, 但又不完全一样 Owner可以对属于组的项目做所有操作, Master用户可以创建属于组的项目
- project : 用户可以创建项目, 项目可以属于用户也可以属于组
- user : 用户可以加入组中去, 可以创建项目 用户有Admin, NonAdmin两种类型

Gitlab定义了以下几个角色: [参考文档](#)

- Guest - 访客 只能查看项目主页，无法查看文件
- Reporter - 报告者; 可以理解为测试员、产品经理等，一般负责提交issue等
- Developer - 开发者; 负责开发
- Master - 主人; 一般是组长，负责对Master分支进行维护
- Owner - 拥有者; 一般是项目经理

如下实例使用的架构以一般小公司为例(仅有一个项目)

人员:

- 项目经理
 - 产品经理
 - PHP开发
 - IOS开发
 - ANDROID开发
 - 测试
 - 运维
1. 项目经理创建相就的group 如 PHP,IOS,ANDROID 项目经理为所有group的Owner
 2. 项目经理把相关人员加入到各分组并组于相应的权限
 3. 导入相关的代码到相应的组后即可使用

如下实例使用的架构以一般中型公司为例(有多个项目)

1. 项目经理为自己所在的项目组创建一个group 每个项目组的代码全部属于这个组，不属于个人
2. 项目经理为每个项目的每个成员配置相应的权限

注意： gitlab 9 之后版本添加了可以把项目分享给其他 group 并设置权限的功能

6. gitlab汉化

一般汉化的版本会比官方英文版本晚几个版本，如果需要汉化就不能安装最新版
这个汉化版本会比下面使用的那个快一点 <https://gitlab.com/xhang/gitlab/> 可以直接checkout出相应的版本 直接替换原来安装的gitlab-ce web相关的文件

参考文档

```
1.查看当前使用的gitlab版本
cat /opt/gitlab/embedded/service/gitlab-rails/VERSION
# 当前安装版本是8.8.9，因此中文补丁需要打8.8版本。

2.克隆gitlab仓库
cd /server/software/
git clone https://git.coding.net/larryli/gitlab.git

3.使用补上进行汉化
# # 8.8 版本的汉化补丁（8-8-stable是英文稳定版，8-8-zh是中文版，两个 diff 结果便是汉化补丁）
cd gitlab
git diff origin/8-8-stable origin/8-8-zh > /tmp/8.8.diff

# 停止 gitlab
gitlab-ctl stop

# 应用汉化补丁
cd /opt/gitlab/embedded/service/gitlab-rails
git apply /tmp/8.8.diff

# 启动gitlab
gitlab-ctl start
```

```
# 访问测试
http://192.168.12.211/
```

7. 使用docker运行

```
1. 拉取镜像
docker pull beginor/gitlab-ce

2. 运行
docker run \
  --detach \
  --publish 8443:443 \
  --publish 8080:80 \
  --name gitlab \
  --restart unless-stopped \
  --volume /mnt/sda1/gitlab/etc:/etc/gitlab \
  --volume /mnt/sda1/gitlab/log:/var/log/gitlab \
  --volume /mnt/sda1/gitlab/data:/var/opt/gitlab \
  beginor/gitlab-ce
3. 访问运行
http://192.168.12.211:8080/
```

8. 备份与恢复

参考文档

1. 备份配置文件

```
cd /etc && tar cvzf /data/backup/gitlab/gitlab-etc-$CI_BUILD_REF.tgz gitlab
```

2. 备份数据文件

默认数据备份目录是/var/opt/gitlab/backups，手动创建备份文件：

```
# Omnibus 方式安装使用以下命令备份
gitlab-rake gitlab:backup:create

# 日常备份，添加 crontab
# 每天2点执行备份
0 2 * * * /opt/gitlab/bin/gitlab-rake gitlab:backup:create CRON=1

# 如要修改备份周期和目录，在/etc/gitlab/gitlab.rb中修改以下两个选项

# 设置备份周期为7天 - 604800秒
gitlab_rails['backup_keep_time'] = 604800
# 备份目录
gitlab_rails['backup_path'] = '/data/backup/gitlab'
```

3. 恢复

恢复之前，确保备份文件所安装 GitLab 和当前要恢复的 GitLab 版本一致。首先，恢复配置文件：

```
1. 恢复配置文件
mv /etc/gitlab /etc/gitlab.$CI_BUILD_REF
# 将下面配置备份文件的时间戳改为你所备份的文件的时间戳
tar xf /data/backup/gitlab/gitlab-etc-2016-10-27.tgz -C /etc

2. 恢复数据文件
# 将数据备份文件拷贝至备份目录
cp 1477559029_gitlab_backup.tar /var/opt/gitlab/backups/

# 停止连接数据库的进程
gitlab-ctl stop unicorn
gitlab-ctl stop sidekiq

# 恢复备份文件，将覆盖GitLab数据库！
gitlab-rake gitlab:backup:restore BACKUP=1477559029

# 启动 GitLab
```

```
gitlab-ctl start

# 检查 GitLab
gitlab-rake gitlab:check SANITIZE=true
```

9. 持续集成(GitLab-CI)

Gitlab-CI是GitLab Continuous Integration（Gitlab持续集成）的简称。从Gitlab的8.0版本开始，gitlab就全面集成了Gitlab-CI,并且对所有项目默认开启。只要在项目仓库的根目录添加.gitlab-ci.yml文件，并且配置了Runner（运行器），那么每一次合并请求（MR）或者push都会触发CI pipeline。

Gitlab-runner是.gitlab-ci.yml脚本的运行器，Gitlab-runner是基于Gitlab-CI的API进行构建的相互隔离的机器（或虚拟机）。GitLab Runner 不需要和Gitlab安装在同一台机器上，但是考虑到GitLab Runner的资源消耗问题和安全问题，也不建议这两者安装在同一台机器上。Gitlab Runner分为两种，Shared runners和Specific runners。Specific runners只能被指定的项目使用，Shared runners则可以运行所有开启 Allow shared runners选项的项目。

Pipelines是定义于.gitlab-ci.yml中的不同阶段的不同任务。我把Pipelines理解为流水线，流水线包含有多个阶段（stages），每个阶段包含有一个或多个工序（jobs），比如先购料、组装、测试、包装再上线销售，每一次push或者MR都要经过流水线之后才可以合格出厂。而.gitlab-ci.yml正是定义了这条流水线有哪些阶段，每个阶段要做什么事。

Badges 徽章，当Pipelines执行完成，会生成徽章，你可以将这些徽章加入到你的README.md文件或者你的网站。

1. 安装gitlab-ci-multi-runner

```
# 使用清华大学的源
cat >/etc/yum.repos.d/gitlab-ci-multi-runner.repo<<EOF
[gitlab-ci-multi-runner]
name=gitlab-ci-multi-runner
baseurl=https://mirrors.tuna.tsinghua.edu.cn/gitlab-ci-multi-runner/yum/el7
repo_gpgcheck=0
gpgcheck=0
enabled=1
gpgkey=https://packages.gitlab.com/gpg.key
EOF
yum makecache
yum install -y gitlab-ci-multi-runner
```

2. 注册Runner

以管理员身份登录查看token <http://192.168.12.211/admin/runners>

要注册新的 runner 你需要输入后面的注册授权码。Ru
权码是 **DRw6RkQzHyXrQZg4D2bz**

你可以点击下面的按钮重置 runner 的注册授权码。

[重置 runner 注册授权码](#)

一个 'runner' 就是一个运行构建的进程。你可以按照自己的需要设置很多个 runner。Runners 可以被部署在不同的用户、服务器上，甚至在你本地使用的机器上。

每个 runner 可以是下列状态之一：

- 共享的** - 会运行全部未特殊指定项目的构建
- 特定的** - 只运行被特殊指定到该 runner 的项目的构建
- 暂停的** - 该 runner 不会接受任何新构建

类型	Runner 授权码	描述
----	------------	----

每台机器可以注册多个runner

注册共享的runner

```
# Runner需要注册到Gitlab才可以被项目所使用，一个gitlab-ci-multi-runner服务可以注册多个Runner。
gitlab-ci-multi-runner register

Running in system-mode.

Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/ci):
http://192.168.12.211/ci
Please enter the gitlab-ci token for this runner:
DRw6RkQzHyXrQZg4D2bz          # 输入Token
Please enter the gitlab-ci description for this runner:
[xxy-web-test-02]: test-runner # 输入runner的名称
Please enter the gitlab-ci tags for this runner (comma separated):
test,php                      # 输入runner的标签，以区分不同的runner，标签间逗号分隔
Registering runner... succeeded runner=YDPz2or3
Please enter the executor: ssh, shell, parallels, docker, docker-ssh, virtualbox:
shell
Runner registered successfully. Feel free to start it, but if it s running already the config should be automatically reloaded!
```

为某个project注册特定的runner

- 1.找出project的runner注册授权码 在项目配置下 例如 `http://192.168.12.211/PHP/famulei_v2/runners`
- 2.注册时使用这个注册授权码即可注册为这个project专用的runner

使用非交互注册

```
gitlab-ci-multi-runner register \
--non-interactive \
--url "http://192.168.12.211/ci/" \
```



```
--registration-token "DRw6RkQzHyXrQZg4D2bz" \
--description "runner-lab2-docker" \
--tag-list "lab2,docker" \
--executor "docker" \
--docker-image python:2.7
```

3. 查看runner <http://192.168.12.211/admin/runners>

4. 配置构建任务

参考文档

任何提交或者 Merge Request 的合并都可以触发 Pipeline

gitlab-ci.yml语法参考文档

1. 在项目根目录添加.gitlab-ci.yml文件

2. 测试使用实例

把以下内容放入项目根目录的.gitlab-ci.yml
提交之后 之后 更新代码就会自动构建

定义 stages

```
stages:
  - build
  - test
```

定义 job

```
job1:
  stage: test
  script:
    - echo "I am job1"
    - echo "I am in test stage"
```

定义 job

```
job2:
  stage: build
  script:
    - echo "I am job2"
    - echo "I am in build stage"
```

5. 使用docker来构建测试python项目测试

提示使用docker时一定要把gitlab-runner加入docker组

```
usermod -aG docker gitlab-runner
```

1. 创建python项目的gitlab仓库 python-docker-test

2. 添加如下文件到 python-docker-test 仓库

README.md

python-docker-test 自动创建docker测试

hello.py

```
# coding=utf-8
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
def index():
    return 'Hello, World!'
```

```
def index():
    return '<h1>Hello World !</h1>'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

requirements.txt

flask

pip.conf

```
[global]
index-url = http://pypi.douban.com/simple
trusted-host = pypi.douban.com
timeout = 120
```

.dockerignore

```
.git
.gitlab-ci.yml
README.md
```

.gitignore

*.pyc

Dockerfile

```
FROM python:2.7
COPY . /myapp/
WORKDIR /myapp/
RUN mkdir ~/.pip && cp pip.conf ~/.pip/
RUN pip install -r requirements.txt
CMD ["python", "hello.py"]
EXPOSE 5000
```

3. 添加 .gitlab-ci.yml 文件到仓库

此方法使用executor为shell的runner才能构建 当有代码更新时把代码打包到新构建的镜像中去 启动镜像就可以直接运行 script的命令都是在shell里运行

.gitlab-ci.yml

```
before_script:
  - docker info

build_image:
  only:
    - master
  # 指定标签为tag的runners
  tags:
```

```

- lab2
- shell
script:
- docker ps -a | grep flask_hello_${CI_BUILD_REF} && rm -f flask_hello_${CI_BUILD_REF}
- docker build -t python:hello .
- docker run -d --name flask_hello_${CI_BUILD_REF} -p 127.0.0.1:5000:5000 python:hello && sleep 3
- curl -s http://127.0.0.1:5000/ | grep -i 'hello' && docker rm -f flask_hello_${CI_BUILD_REF} || exit -1

```

下面的方法使用executor为docker的runner构建方式 script的命令都是在容器里运行

构建过程中由于无法解析主机需要修改配置文件/etc/gitlab/gitlab.rb
external_url 'http://192.168.12.211'

```

# 使用配置生效
gitlab-ctl reconfigure

```

.gitlab-ci.yml

```

image: python:2.7

services:
- mysql:5.6

variables:
# Configure mysql environment variables (https://hub.docker.com/r/_/mysql/)
MYSQL_DATABASE: el_duderino
MYSQL_ROOT_PASSWORD: mysql_strong_password

before_script:
- mkdir ~/.pip && cp pip.conf ~/.pip/
- pip install -r requirements.txt

test:
only:
- master
# 指定标签为tag的runners
tags:
- lab2
- docker
script:
- python hello.py &
- sleep 3 ; curl -s http://127.0.0.1:5000/ | grep -i 'hello' && pkill python || exit -1

```