**Nginx**

* 使用官方nginx镜像

# docker pull daocloud.io/nginx:1.12

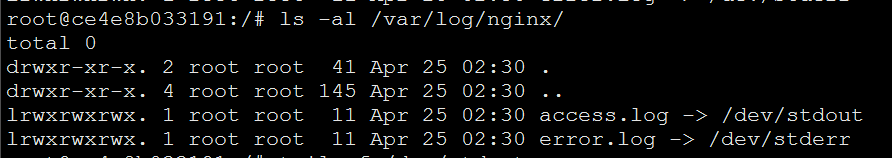
* 运行

# docker run -d -p 80:80 --name webserver daocloud.io/nginx:1.12

* 查看日志

# docker logs webserver

官方默认日志重定向到stderr和 stderr的



在容器里面通过tial是看不到日志的输出的，怎样才能看到？

把日志导向数据卷容器怎么做？

* 进入容器

# docker exec -it webserver /bin/bash

* 自定义web页面

# docker run -d -p 80:80 --name webserver -v /home/docker\_storage/nginx\_web/:/usr/share/nginx/html/:ro daocloud.io/nginx:1.12

如果报 Permission denied ，请加上 --privileged=true

* 重启nginx容器

# docker exec -it <container id> <path\_of\_nginx> -s reload

**PHP**

* 官方镜像

# docker pull daocloud.io/php:7.1.4-fpm

* 启动

# docker run -d --name php-fpm daocloud.io/php:7.1.4-fpm

* 修改php-fpm配置

（用户和用户组需要修改吗）

重新启动

# docker run -d --name php-fpm -v /home/docker\_storage/nginx\_web/:/var/www/html/:ro --privileged=true daocloud.io/php:7.1.4-fpm

修改nginx配置

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log /var/log/nginx/log/host.access.log main;

**root /var/www/html;**

index index.html index.htm index.php;

location / {

#root /usr/share/nginx/html;

#index index.html index.htm;

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

#

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

#root /usr/share/nginx/html;

}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80

#

#location ~ \.php$ {

# proxy\_pass http://127.0.0.1;

#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

#

location ~ \.php$ {

# root html;

fastcgi\_pass **php-fpm:9000;**

fastcgi\_index index.php;

#fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

include fastcgi\_params;

}

# deny access to .htaccess files, if Apache's document root

# concurs with nginx's one

#

#location ~ /\.ht {

# deny all;

#}

}

* 启动Nginx

# docker run -d --link php-fpm:php-fpm -p 80:80 --name webserver -v /home/docker\_storage/default.conf:/etc/nginx/conf.d/default.conf:ro -v /home/docker\_storage/nginx\_web/:/var/www/html/:ro --privileged=true daocloud.io/nginx:1.12

打开phpinfo的页面观察是否成功。

（注意启动php-fpm时要挂载本地web目录，启动nginx时也需要挂载，否则不能访问php资源。

可以在php-fpm启动时作为数据卷容器挂载web目录，然后nginx启动时使用php-fpm的数据卷容器。

# docker run -d --name php-fpm -v /home/docker\_storage/nginx\_web/:/var/www/html/:ro --privileged=true daocloud.io/php:7.1.4-fpm

# docker run -d --link php-fpm:php-fpm -p 80:80 --name webserver -v /home/docker\_storage/default.conf:/etc/nginx/conf.d/default.conf:ro --privileged=true --volumes-from php-fpm daocloud.io/nginx:1.12

或者把web目录直接作为一个数据卷容器，然后php-fpm和nginx容器都挂载该容器

）

**MySQL**

* 官方镜像

# docker pull daocloud.io/mysql:5.7.18

* 自定义 MySQL 配置文件
* 启动MySQL

# docker run -d --name mysql -p 3306:3306 -v /home/docker\_storage/mysql\_data/:/var/lib/mysql/:rw --privileged=true -e MYSQL\_ROOT\_PASSWORD=123456 daocloud.io/mysql:5.7.18

* 启动PHP-FPM

# docker run -d --link mysql:mysql --name php-fpm -v /home/docker\_storage/nginx\_web/:/var/www/html/:ro --privileged=true daocloud.io/php:7.1.4-fpm

* 启动Nginx

# docker run -d --link php-fpm:php-fpm -p 80:80 --name webserver -v /home/docker\_storage/nginx\_conf/default.conf:/etc/nginx/conf.d/default.conf:ro --privileged=true --volumes-from php-fpm daocloud.io/nginx:1.12

* 测试 PHP & MySQL：

# vi /var/www/html/test.php

添加

<?php

//date

echo date("Y-m-d H:i:s")."<br />\\n";

//mysql

try {

$conn = new PDO('mysql:host=mysql;port=3306;dbname=mysql;charset=utf8', 'root', '123456');

} catch (PDOException $e) {

echo 'Connection failed: ' . $e->getMessage();

}

//$conn->exec('set names utf8');

$sql = "SELECT \* FROM `user` WHERE 1";

$result = $conn->query($sql);

while($rows = $result->fetch(PDO::FETCH\_ASSOC)) {

echo $rows['Host'] . ' ' . $rows['User']."<br />\\n";

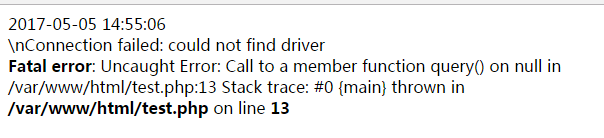
}

//phpinfo

phpinfo();

?>

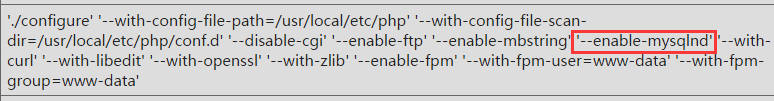
然后访问，报错



应该是php-mysql的驱动问题。

PHP连接mysql有三种方式：mysql、mysqli、pdo。其中mysql是不建议使用的了。

我们使用官方镜像只有mysqlnd



没有mysqli和pdo。所以用mysqli和pdo的连接都会失败。

PHP7 正式移除了 mysql 扩展。

我们可以在原官方镜像上进行扩展。

编写Dockerfile：

FROM daocloud.io/php:7.1.4-fpm

# Install modules

RUN docker-php-ext-configure pdo\_mysql --with-pdo-mysql=mysqlnd \

&& docker-php-ext-configure mysqli --with-mysqli=mysqlnd \

&& docker-php-ext-install pdo\_mysql \

&& docker-php-ext-install mysqli

["php-fpm"]

构建：

# docker build -t php7-fpm:7.1.4 .

重新启动php-fpm

# docker run -d --link mysql:mysql --name php-fpm -v /home/docker\_storage/nginx\_web/:/var/www/html/:ro --privileged=true php7-fpm:7.1.4

查看phpinfo可以看到有mysqli和pdo\_mysql的参数了。

测试mysqli连接可以使用如下代码：

<?php

$servername = "mysql";

$username = "root";

$password = "123456";

// 创建连接

$conn = new mysqli($servername, $username, $password);

// 检测连接

if ($conn->connect\_error) {

die("connect failed: " . $conn->connect\_error);

}

echo "connected success";

?>

**DVWA**

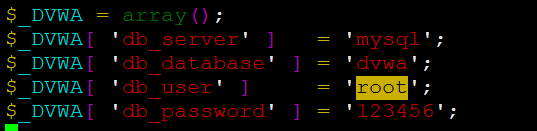
<http://www.dvwa.co.uk/>

下载源码包，上传到web目录下。

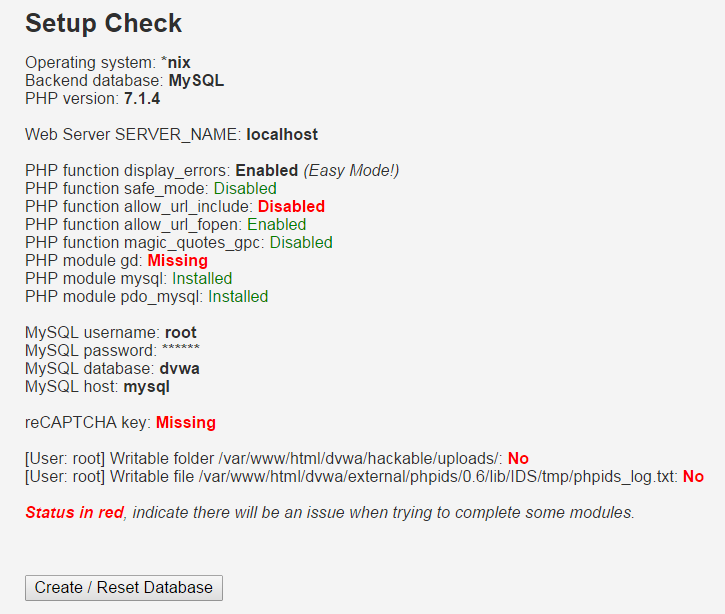
复制配置文件：

# cp config/config.inc.php.dist config/config.inc.php

编辑配置文件：



打开dvwa页面，正常显示如下：



PHP5.2开始allow\_url\_include就默认为Off了，因此出于安全最好不要开启。这里因为测试的话可以暂时打开。

还要安装gd扩展（php处理图形的扩展库）

修改Dockerfile：

FROM daocloud.io/php:7.1.4-fpm

# Install modules

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

libjpeg-dev libpng-dev libfreetype6-dev \

&& docker-php-ext-configure pdo\_mysql --with-pdo-mysql=mysqlnd \

&& docker-php-ext-configure mysqli --with-mysqli=mysqlnd \

&& docker-php-ext-configure gd -with-gd --enable-gd-native-ttf \

--with-jpeg-dir=/usr/include/ --with-png-dir=/usr/include/ --with-freetype-dir=/usr/include/ \

&& docker-php-ext-install pdo\_mysql \

&& docker-php-ext-install mysqli \

&& docker-php-ext-install gd

CMD ["php-fpm"]

构建：

# docker build -t php7-fpm:7.1.4 .

因为allow\_url\_include开启需要设置php.ini，容器里好像没有这个文件，因此我就把php7源码里的php.ini放进去。

启动php（注意dvwa要有写权限）

# docker run -d --link mysql:mysql --name php-fpm -v /home/docker\_storage/nginx\_web/:/var/www/html/:rw -v **/home/docker\_storage/php\_conf/php.ini:/usr/local/etc/php/php.ini:ro** --privileged=true php7-fpm:7.1.4

修改php.ini，开启 allow\_url\_include

重启容器

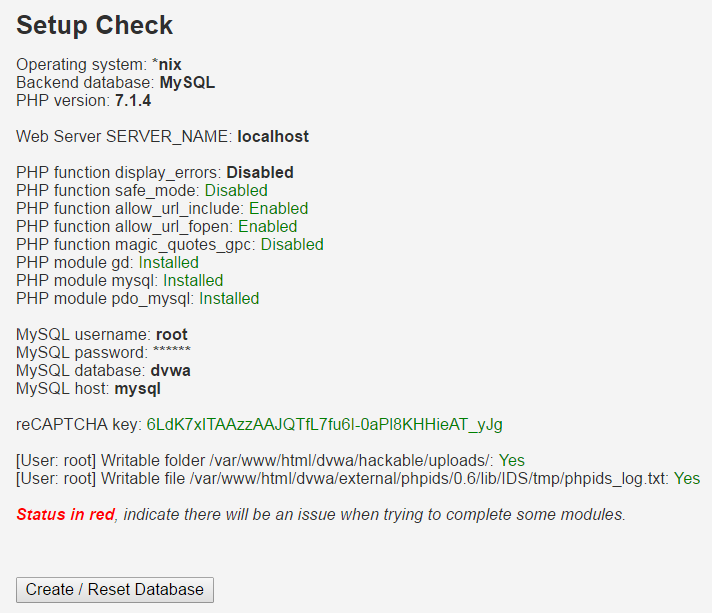
# docker restart php-fpm

修改dvwa的配置

$\_DVWA[ 'recaptcha\_public\_key' ] = '6LdK7xITAAzzAAJQTfL7fu6I-0aPl8KHHieAT\_yJg';

$\_DVWA[ 'recaptcha\_private\_key' ] = '6LdK7xITAzzAAL\_uw9YXVUOPoIHPZLfw2K1n5NVQ' ;

查看dvwa



然后点击创建数据库，创建好后就可以登录了。

默认账号密码是 admin/password

**HAproxy**

# docker pull haproxy:1.4.27

要修改一下默认的haproxy配置，不要启用daemon

# docker run -d --name app\_1 daocloud.io/nginx:1.12

# docker run -d --name app\_2 daocloud.io/nginx:1.12

# docker run -d -p 80:80 --link webserver\_1:webserver\_1 --link webserver\_2:webserver\_2 --name myhaproxy -v /home/docker\_storage/haproxy/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro --privileged=true haproxy:1.4.27

其中haproxy.cfg如下

global

log 127.0.0.1 local0

log 127.0.0.1 local1 notice

#log loghost local0 info

maxconn 4096

#chroot /usr/share/haproxy # 这个干嘛的

uid 99

gid 99

#daemon # docker的话要注释

#debug

#quiet

defaults

log global

mode http

option httplog

option dontlognull

retries 3

#redispatch

maxconn 2000

contimeout 5000

clitimeout 50000

srvtimeout 50000

listen web\_cluster 0.0.0.0:80

option httpchk GET /index.html

balance roundrobin

server app\_1 app\_1:80 check inter 2000 fall 3

server app\_2 app\_2:80 check inter 2000 fall 3

**Gitlab & Jenkins**

这里使用 Ubuntu 16.04 测试。

安装docker： apt-get install docker.io

（安装的版本是1.12.6）

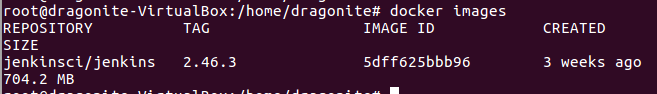
然后我们使用daocloud的加速器，记得要看daocloud的文档，docker 1.12以上的版本有点变化。

**Jenkins安装**

<https://jenkins.io/download/>

这里使用2.46.3 lts版本

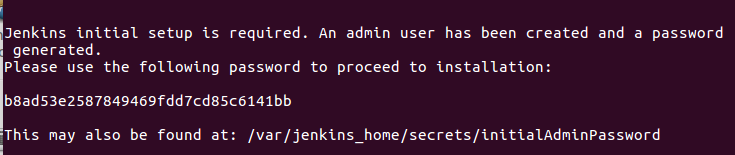
# docker pull jenkinsci/jenkins:2.46.3



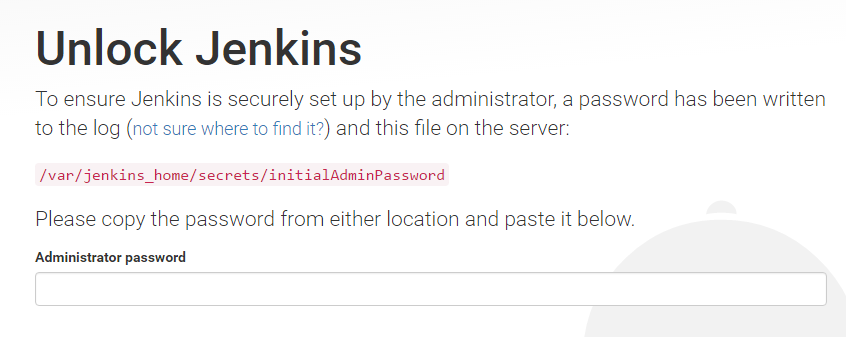
启动 （记住使用数据卷，因为jenkins会把插件和数据存放到/var/jenkins\_home下，所以我们需要把数据持久化）

# docker run -p 8080:8080 -p 50000:50000 -v /var/jenkins\_home:/var/jenkins\_home jenkins

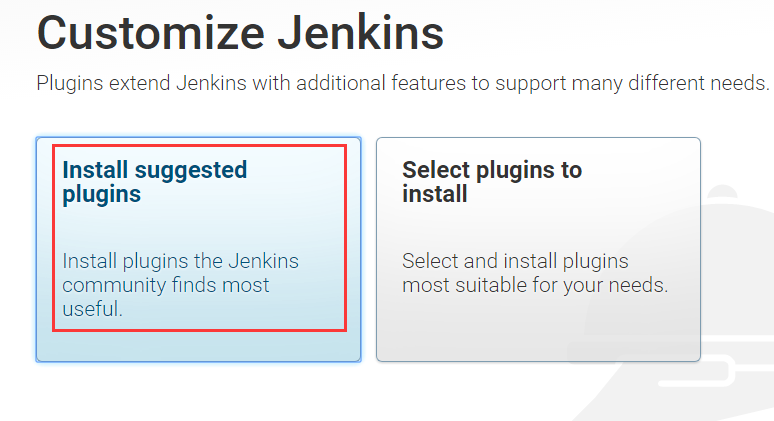
启动后它会在控制台生成一个口令的，



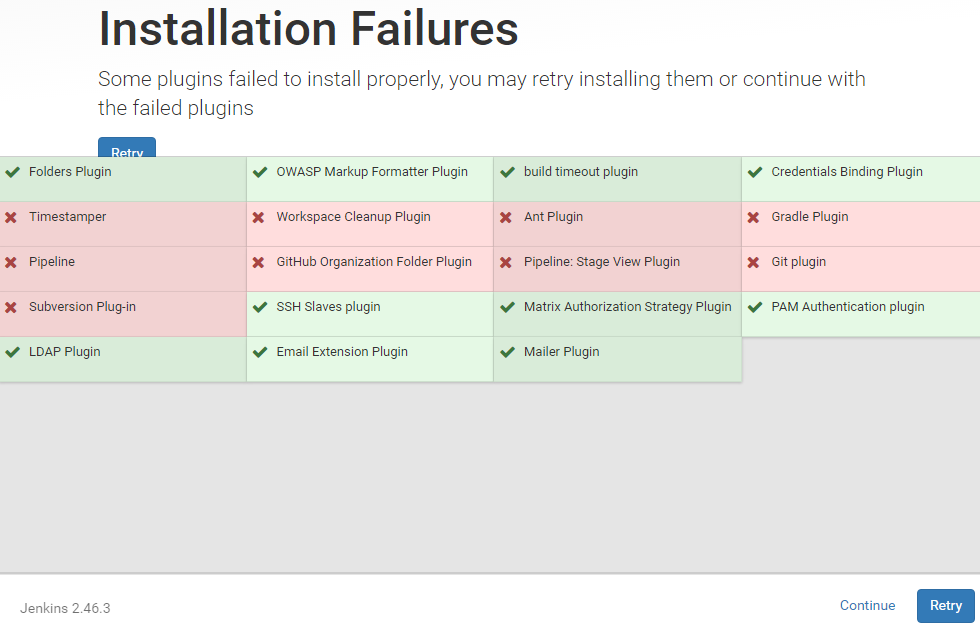
然后打开页面 http://ip:8080



输入密码

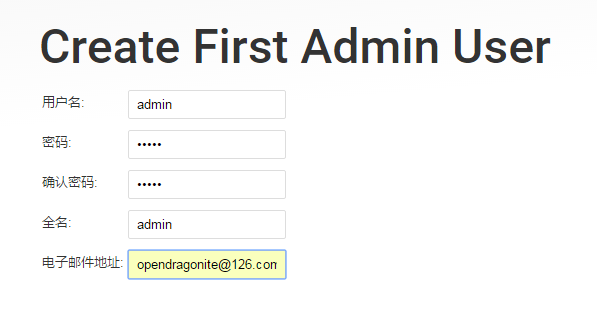


安装默认插件



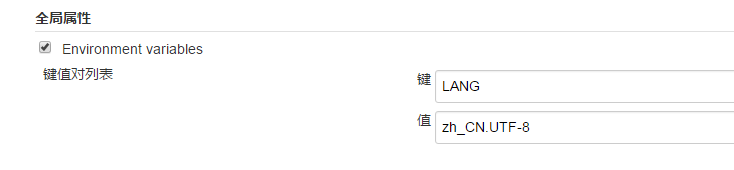
安装过程中可能出现插件装不了的情况，多retry几次就可以全部安装了。

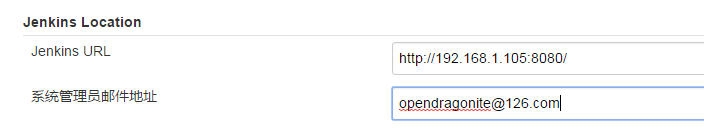
插件装完后，创建管理员



进去还需要配置。（可以先把/var/jenkins\_home 备份一下，搞坏了可以换个docker继续使用这个初始目录，这样就不用重新安装插件了）

配置

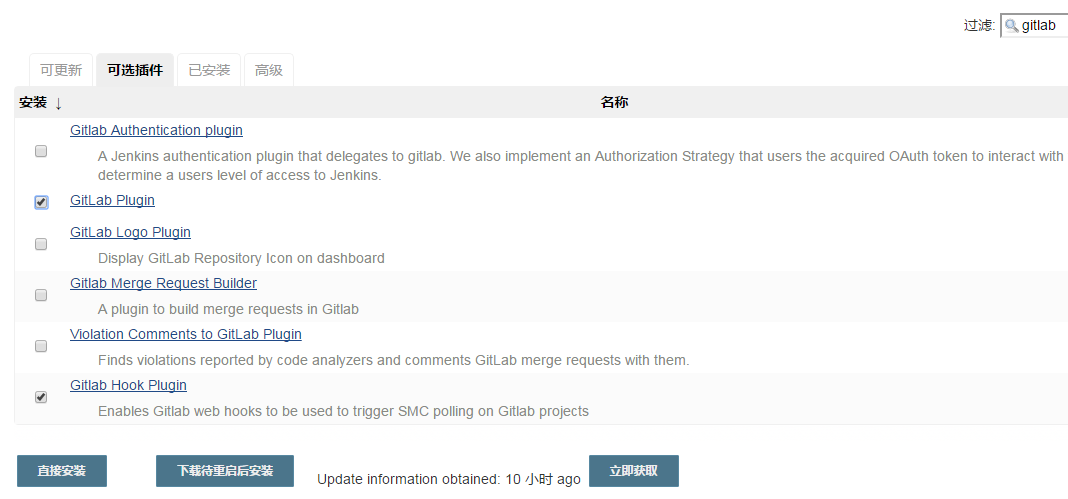






安装gitlab插件：

选择 系统管理 -> 管理插件 -> 可选插件 那里选择gitlab



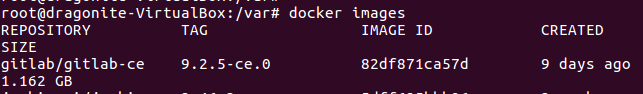
然后直接安装。

**Gitlab安装**

<https://about.gitlab.com/>

Gitlab的系统要求还是蛮高的（2U4GB）。

# docker pull gitlab/gitlab-ce:9.2.5-ce.0



启动

# docker run --name gitlab -d \

-p 9022:22 -p 9080:80 -p 9443:443 \

--restart always \

-v /var/gitlab/config:/etc/gitlab \

-v /var/gitlab/logs:/var/log/gitlab \

-v /var/gitlab/data:/var/opt/gitlab \

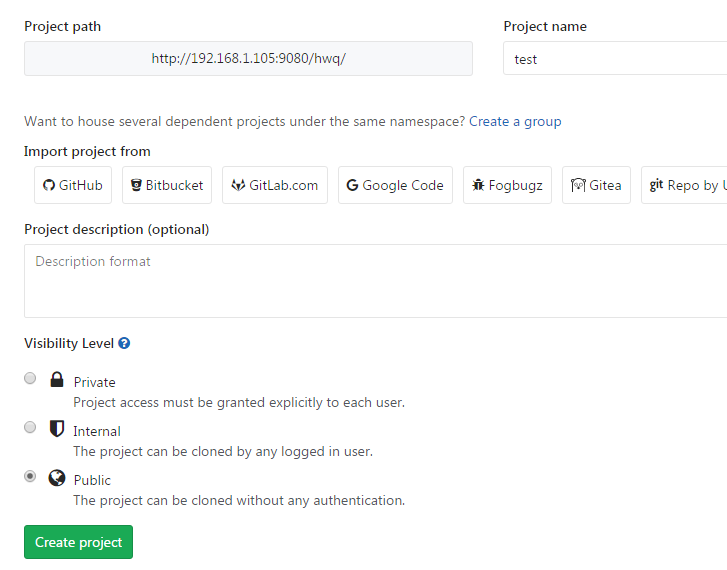
gitlab/gitlab-ce:9.2.5-ce.0

配置

访问 <http://192.168.1.105:9080>

第一次登陆使用的是root账号，页面会让你设置密码（我设了admin123456），然后就可以用这个账号密码登陆了，也可以注册新账号。

注册新账号并不会发送注册邮件？



创建一个测试项目，创建好后只能需要url来clone，不能使用ssh，除非你向项目添加了ssh认证。

记住clone的时候，那个url地址要加上gitlab对应端口号（如果不是80的话）

然后修改后提交就可以在页面上看到提交了。

**RabbitMQ**

# docker pull daocloud.io/rabbitmq:3

# docker pull daocloud.io/rabbitmq:3-management

RabbitMQ 已经有一些自带管理插件的镜像。用这些镜像创建的容器实例可以直接使用默认的 15672 端口访问，默认账号密码是guest/guest

# docker run -d --hostname my-rabbit --name some-rabbit daocloud.io/rabbitmq:3-management

然后打开浏览器访问 http://容器 IP:15672 ，就可以管理你的 RabbitMQ 实例了，或者你可以暴露主机端口来访问：

# docker run -d --hostname my-rabbit --name some-rabbit -p 8080:15672 daocloud.io/rabbitmq:3-management

这时，你可访问 http://localhost:8080 或者 http://宿主 IP:8080 管理 RabbitMQ 服务了。

出于安全因素的考虑，guest用户只能通过localhost登陆使用，并建议修改guest用户的密码以及新建其他账号管理使用rabbitmq。

（用户、角色、权限）

首先启动rabbitmq容器，然后利用dockerfile创建python容器并连接到rabbitmq容器进行运行app。

创建完容器后运行

生产者

# docker run -it --rm --name my-rabbit-producer --link some-rabbit:rabbit -v "$PWD":/usr/src/myapp -w /usr/src/myapp py-rabbit python send.py

消费者

# docker run -it --rm --name my-rabbit-consumer --link some-rabbit:rabbit -v "$PWD":/usr/src/myapp -w /usr/src/myapp py-rabbit python receive.py

集群：