# DevOps搭建文档

引言：

DevOps（Development和Operations的组合词）是一种重视开发人员（dev）和运维人员（ops）之间沟通合作的一种开发模式。通过自动化“软件交付”和“构建变更”的流程，来使得构建、测试、发布软件能够更加快捷、频繁、可靠。可以将DevOps看成是软件开发生命周期（SDLC）由瀑布式到敏捷再到精益的发展之后的产物。

伴随着越来越多的公司开始选用DevOps作为一个项目的开发模式，DevOps也成为了时下最为流行、高效率的方式去交付软件。那么对于全球最为强大的开源ERP系统Odoo的模块开发来说，引入DevOps也是十分有必要的。

在引入DevOps开发之前，开发人员提交代码时，需要通知其他开发人员暂停手上的代码提交，然后恢复最新的纯净版数据库更新并配置，配置完成后才能让下一个开发人员提交。而且每次更新测试服务器，都要一个开发人员更新测试服务器，解决bug或新增功能无法第一时间让测试人员测试，导致测试人员效率低下，DevOps引入后，开发人员自测完成后直接提交代码，代码就自动同步到测试服务器，登录测试服务器升级完成后，就可以让测试人员测试，减少不需要人为操作，提高效率，可以让开发人员同步开发。

## 进行DevOps流程搭建时，使用的工具。

### 代码托管—gitlab



Gitlab 是一个基于 Git的代码托管和研发协作的开源平台，同时，在Gitlab上还集成了大量用于软件开发和部署以及项目管理的必要工具。其次，Gitlab的部署配置相对容易，简单，可以在Kubernetes上快速完成其部署和启动。

### CI/CD工具--gitlab-Runner

GitLab Runner 是一个与 GitLab CI/CD 配合使用以在管道中运行作业的应用程序。

### 容器化--docker



容器化，相较于传统的虚拟机部署有着太多的优势，效率、轻量、复用性、可移植性、隔离性等等，都是虚拟机部署相较于容器化部署的劣势。利用docker，我们可以很好的部署、管理我们的应用程序容器以及镜像，同时，docker也提供了容器运行时所需要的环境。

## Dev搭建流程

### docker部署gitlab

#### 1.前提条件

(1) 安装docker

sudo curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun

(2) 服务器可以联网

(3)服务器内存至少4G(内存不够会出现502错误)

#### 2.安装

(1) 查找gitlab镜像

docker search gitlab

(2) 拉取gitlab docker镜像

docker pull gitlab/gitlab-ce:latest

(3) 运行gitlab容器

docker run \

 -itd  \

 -p 80:80 \

 -p 27:22 \

 -v /usr/local/gitlab-test/etc:/etc/gitlab  \

 -v /usr/local/gitlab-test/log:/var/log/gitlab \

 -v /usr/local/gitlab-test/opt:/var/opt/gitlab \

 --restart always \

 --privileged=true \

 --name gitlab-test \

 gitlab/gitlab-ce

命令解释：

-i 以交互模式运行容器，通常与 -t 同时使用命令解释：

-t 为容器重新分配一个伪输入终端，通常与 -i 同时使用

-d 后台运行容器，并返回容器ID

-p 80:80 将容器内80端口映射至宿主机80端口，这是访问gitlab的端口

-p 27:22 将容器内22端口映射至宿主机27端口，这是访问ssh的端口

-v /usr/local/gitlab-test/etc:/etc/gitlab 将容器/etc/gitlab目录挂载到宿主机/usr/local/gitlab-test/etc目录下，若宿主机内此目录不存在将会自动创建，其他两个挂载同这个一样

--restart always 容器自启动

--privileged=true 让容器获取宿主机root权限

--name gitlab-test 设置容器名称为gitlab-test

gitlab/gitlab-ce 镜像的名称，这里也可以写镜像ID

(3).进入容器内

docker exec -it gitlab-test /bin/bash

(4).修改gitlab.rb （先查看下一个步骤再决定是否进行本步骤，本步骤是可以跳过的）

# 打开文件

vi /etc/gitlab/gitlab.rb

# gitlab访问地址，可以写域名。如果端口不写的话默认为80端口 （图片上拼写错误，正确的是external\_url）

external\_url 'http://192.168.17.195'

# ssh主机ip

gitlab\_rails['gitlab\_ssh\_host'] = '192.168.17.195'

# ssh连接端口

gitlab\_rails['gitlab\_shell\_ssh\_port'] = 27

(5). 修改gitlab.yml

# 打开文件

vi /opt/gitlab/embedded/service/gitlab-rails/config/gitlab.yml

## GitLab settings

gitlab:

## Web server settings (note: host is the FQDN, do not include http://)

host: 192.168.17.195

port: 80

https: false

# Uncommment this line below if your ssh host is different from HTTP/HTTPS one

# (you'd obviously need to replace ssh.host\_example.com with your own host).

# Otherwise, ssh host will be set to the `host:` value above

ssh\_host: 192.168.17.195

(6)、让修改后的配置生效

gitlab-ctl reconfigure

(7)、重启gitlab

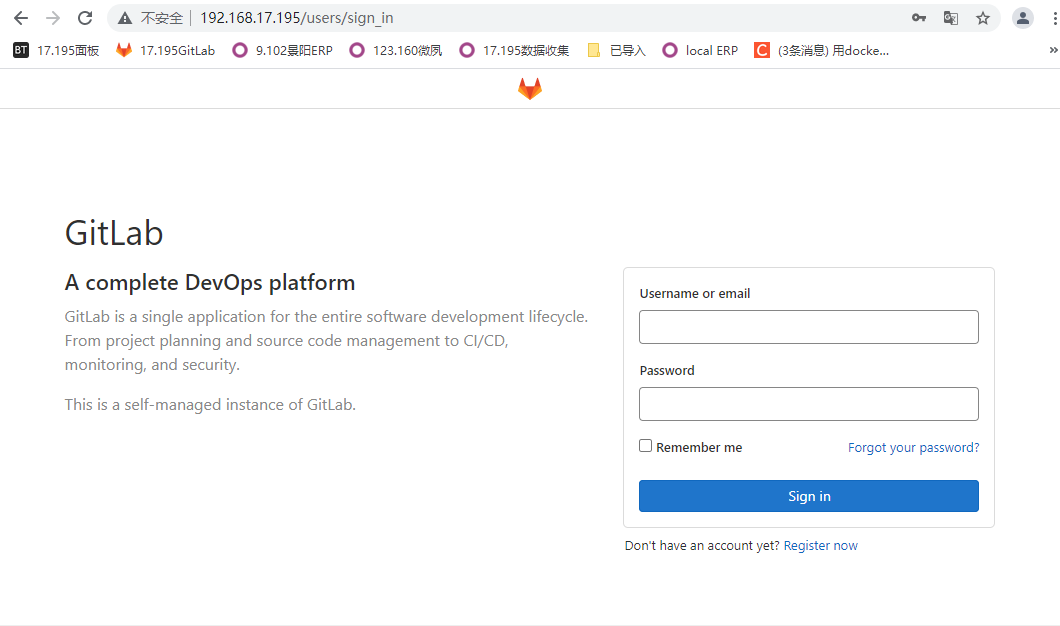
gitlab-ctl restart

(8)、退出容器

**exit**

(9)、在游览器输入如下地址，访问gitlab（eaxternal\_url配置的就是这个）

http://192.168.17.195



#如果访问不成功的话：

进入容器查看gitlab.rb和gitlab.yml文件是否配置成功

查看防火墙是否开放27端口

(10)、第一次访问默认是root账户，需要修改密码

启动Ruby on Rails控制台

gitlab-rails console -e production

搜索电子邮件或用户名

user = User.where(id: 1).first

更改密码

user.password = '12345678'

确认更改密码

user.password\_confirmation = '12345678'

保存

user.save!

### docker安装gitlab-runner

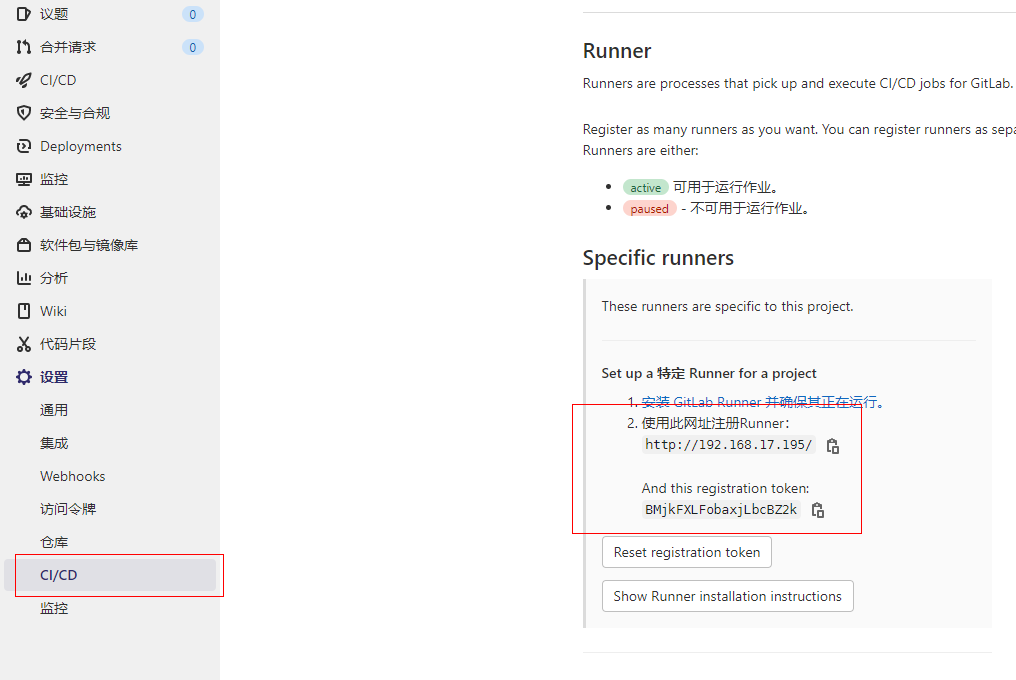
#### 1. 拉取gitlab-runner docker镜像

docker pull gitlab/gitlab-runner:latest

#### 2. 运行镜像

docker run -d --name gitlab-runner --restart always -v /var/run/docker.sock:/var/run/docker.sock -v /srv/gitlab-runner/config:/etc/gitlab-runner gitlab/gitlab-runner:latest

#### 3. 注册gitlab-runner



docker exec -it gitlab-runner gitlab-runner register

输入以下命令

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

http://192.168.17.195/

token Please enter the gitlab-ci token for this runner:

BMjkFXLFobaxjLbcBZ2k

Please enter the gitlab-ci description for this runner:

test

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

test

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

shell

#### 4. git-runner免密登录测试服务器

(1) 进入容器

docker exec -it [容器id] bash

(2)使用ssh-keygen命令创建密钥对

ssh-keygen  -t  rsa

ssh-keygen -R 192.168.9.102

(3)执行命令后根据提示输入密钥的密码和存放路径

/root/.ssh/id\_rsa.pub

/root/.ssh/id\_rsa

(4)ssh-copy-id命令推送公钥文件到以后需要分发文件的服务器

ssh-copy-id -i /root/.ssh/id\_rsa.pub [sunell@192.168.9.102](mailto:sunell@192.168.9.102)

(5)命令执行后在分发服务器的对应用户家目录中会生成.ssh目录，并且该目录下还有一个authorized\_keys

#### 5. 编写.gitlab-ci.yml文件

(1) 在ERP项目的根目录编写.gitlab-ci.yml配置文件

stages:          *# 定义流水线阶段*

  - build

  - test

  - deploy

build-job:       *#构建工作*

  tags:

    - erp

  only:

    - Develop\_V1.0

  stage: build

  script:

    - echo "$ERP\_Service\_Pwd"

    - echo "运行打包脚本-------------------------------------------"

    - chmod u+x ./script/odoo\_code\_package.sh && ./script/odoo\_code\_package.sh

    - echo "运行打包脚本结束--------------------------------------"

    - echo "运行下载脚本-------------------------------------------"

    - chmod u+x ./script/odoo\_download\_code.sh && ./script/odoo\_download\_code.sh $ERP\_Service\_IP $ERP\_Service\_Pwd $ERP\_Service\_Path

    - echo "运行下载脚本结束--------------------------------------"

unit-test-job:   *# 单元测试工作*

  tags:

    - erp

  only:

    - Develop\_V1.0

  stage: test    *# It only starts when the job in the build stage completes successfully.*

  script:

    - echo "运行测试用例------------------------------------------"

    - echo "运行测试用例结束-------------------------------------"

deploy-job:      *# 部署工作*

  tags:

    - erp

  only:

    - Develop\_V1.0

  stage: deploy  *# It only runs when \*both\* jobs in the test stage complete successfully.*

  script:

    - echo "运行执行脚本------------------------------------------"

*#    - chmod u+x ./script/odoo\_install\_env.sh && ./script/odoo\_install\_env.sh $ERP\_Service\_IP $ERP\_Service\_Pwd $ERP\_Service\_Path*

    - chmod u+x ./script/odoo\_run\_code.sh && ./script/odoo\_run\_code.sh $ERP\_Service\_IP $ERP\_Service\_Pwd

    - echo "运行执行脚本脚本--------------------------------------"

上面的配置把一次 Pipeline 分成三个阶段：

• 构建工作（build）

• 单元测试工作(test)

• 部署工作(deploy)

设置 Job.only 后，只有当 Develop\_V1.0 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 Develop\_V1.0指在提交到 Develop\_V1.0 时执行

• tags：与注册 runner 时的 tag 匹配

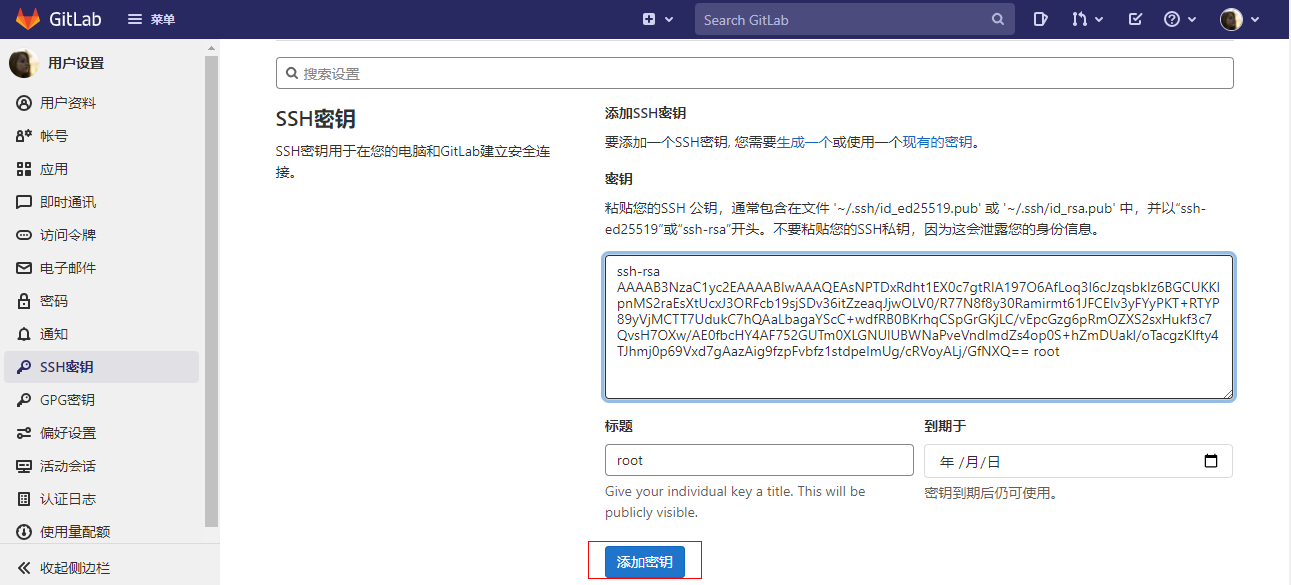
### gitlab配置

#### 1. 在wondow开发环境生成gitlab服务器公钥

ssh-keygen -t rsa -C  'root'

到 C:\Users\你的用户名\.ssh 目录下，会看到有两个文件：id\_rsa, id\_rsa.pub。有 .pub 后缀的文件就是公钥，另一个文件则是密钥

复制id\_rsa.pub,里面的内容到gitlab服务器的ssh密钥



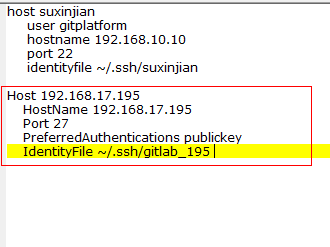
#### 2. window环境的.ssh目录下添加gitlab公钥



路径：C:\Users\当前用户\.ssh

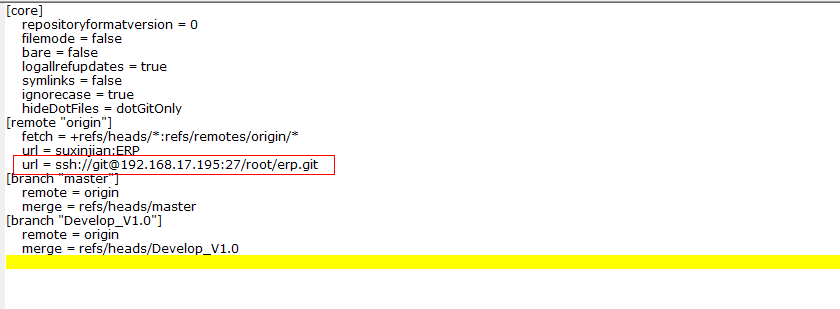
#### **3. config文件添加一下配置**

如下图：

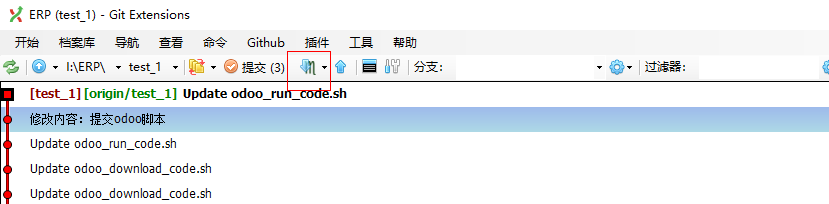


#### 4. ERP代码目录下的.git目录的config文件添加远程路径

如下图：



#### 5. 使用 Git Extensions，打开ERP项目，先拉取代码分支



### docker部署odoo