

Gitlab 安装与使用

Docker 安装

```
yum update -y
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-
ce.repo
yum install -y docker-ce
docker -v
```

```
[root@iZ2zeewcelgkyyw1ttadd5Z gitlab]# docker -v
Docker version 20.10.4, build d3cb89e
```

```
# 启动docker服务
systemctl start docker
# 开机启动docker服务
systemctl enable docker
```

```
# 更换docker阿里源
# https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors 获取源地址
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<- 'EOF'
{
  "registry-mirrors": ["https://yb0133ts.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Docker Compose 安装

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.4/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

```
[root@iZ2zeewcelgkyyw1ttadd5Z gitlab]# docker-compose --version
docker-compose version 1.28.4, build cabd5cfb
```

Gitlab 安装

```
mkdir /root/gitlab
cd /root/gitlab
vim /root/gitlab/docker-compose.yml
```

```
version: '3'
services:
```

```
web:
  image: 'gitlab/gitlab-ce:latest'
  restart: always
  hostname: '39.105.47.81'
  container_name: 'gitlab'
  environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'http://39.105.47.81:8880'
      gitlab_rails['gitlab_shell_ssh_port'] = 8822
      gitlab_rails['time_zone'] = 'Asia/Shanghai'
  ports:
    - '8880:8880'
    - '8443:443'
    - '8822:22'
  volumes:
    - '/root/gitlab/config:/etc/gitlab'
    - '/root/gitlab/logs:/var/log/gitlab'
    - '/root/gitlab/data:/var/opt/gitlab'
```

```
# 安装并启动gitlab
docker-compose up -d
# 查看安装成功
docker ps
```

```
[root@iZ2zeewce1gkyywlttadd5Z gitlab]# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
8654574e2382   gitlab/gitlab-ce:latest             "/assets/wrapper"       14 minutes ago Up 14 minutes (healthy)  80/tcp, 0.0.0.0:8880->8880/tcp, 0.0.0.0:8822->22/tcp, 0.0.0.0:8443->443/tcp
```

访问<http://39.105.47.81:8880/> 即可访问gitlab主页

第一次设置管理员用户名及登录密码

设置中文显示



Gitlab 权限管理

访问权限

- Private - 私有，只有属于该项目成员才有原先查看
- Internal - 内部，用个Gitlab账号的人都可以clone
- Public - 公开，任何人可以clone

行为权限

在满足行为权限之前，必须具备访问权限（如果没有访问权限，那就无所谓行为权限了），行为权限是指对该项目进行某些操作，比如提交、创建问题、创建新分支、删除分支、创建标签、删除标签等。

角色

Gitlab定义了以下几个角色：

- Guest - 访客
- 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

权限

不同角色，拥有不同权限，下面列出Gitlab各角色权限

1. 工程权限

行为	Guest	Reporter	Developer	Master	Owner
创建issue	✓	✓	✓	✓	✓
留言评论	✓	✓	✓	✓	✓
更新代码		✓	✓	✓	✓
下载工程		✓	✓	✓	✓
创建代码片段		✓	✓	✓	✓
创建合并请求			✓	✓	✓
创建新分支			✓	✓	✓
提交代码到非保护分支			✓	✓	✓
强制提交到非保护分支			✓	✓	✓
移除非保护分支			✓	✓	✓
添加tag			✓	✓	✓
创建wiki			✓	✓	✓
管理issue处理者			✓	✓	✓
管理labels			✓	✓	✓
创建里程碑				✓	✓
添加项目成员				✓	✓
提交保护分支				✓	✓
使能分支保护				✓	✓
修改/移除tag				✓	✓
编辑工程				✓	✓
添加deploy keys				✓	✓
配置hooks				✓	✓
切换visibility level					✓
切换工程namespace					✓
移除工程					✓
强制提交保护分支					✓
移除保护分支					✓

PS: 关于保护分支的设置，可以进入Settings->Protected branches进行管理

2. 组权限

行为	Guest	Reporter	Developer	Master	Owner
浏览组	✓	✓	✓	✓	✓
编辑组					✓
创建项目				✓	✓
管理组成员					✓
移除组					✓

Gitlab Runner 安装

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



指定Runner

自动设置一个specific的Runner

可以轻松地在Kubernetes集群上安装Runner。 [进一步了解关于Kubernetes的信息](#)

1. 点击下面的按钮转到Kubernetes页面开始安装过程
2. 选择一个既有的Kubernetes集群或者创建一个新的
3. 在Kubernetes集群详细信息视图中，从应用程序列表中安装Runner

[在Kubernetes上安装Runner](#)

手动设置specific Runner

1. 安装GitLab Runner
2. 在 Runner 设置时指定以下 URL:
`http://39.105.47.81:8880/`
3. 在安装过程中使用以下注册令牌:
`5JhWpSHkDXzKyWSCgt`

[重置 Runner 注册令牌](#)

4. 启动 Runner!

Shared runners

The same shared runner executes code from multiple projects, unless you configure autoscaling with [MaxBuilds](#) set to 1 (which it is on GitLab.com).

[Disable shared runners](#) for this project

此GitLab实例尚未提供任何共享Runner。管理员可以在管理区域中注册共享Runner。

群组Runner

Gitlab群组Runner可以用来运行群组内所有项目的代码。可以使用 [Runners API](#) 进行托管。

[禁用群组Runner](#) 对于这个项目

该群组未提供任何群组Runner。群组维护者可以在通过 [群组CI/CD 设置](#) 注册群组级 Runner

```
# 注册gitlab-runner到gitlab
docker run --rm -it -v /srv/gitlab-runner/config:/etc/gitlab-runner
gitlab/gitlab-runner register
# Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/): 上方的
url
# Please enter the gitlab-ci token for this runner: 上方的令牌
# Please enter the gitlab-ci description for this runner: runner的描述
# Please enter the gitlab-ci tags for this runner (comma separated): gitlab-
ci.yml 根据tag执行
# Please enter the executor: virtualbox, docker+machine, docker, shell, ssh,
kubernetes, docker-ssh, parallels, docker-ssh+machine: docker
```

编写gitlab-ci.yml

```
image: maven:latest

cache:
  paths:
    - .m2/repository

variables:
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"

stages:
  - build
  - package

build:
  stage: build
  tags:
    - maven
  script:
    - echo "=====开始编译构建任务======"
    - mvn compile

package:
  stage: package
  tags:
    - maven
  script:
    - echo "=====开始打包任务======"
    - mvn clean package -Dmaven.test.skip=true

deploy:
  stage: deploy
  image: ubuntu
  tags:
    - maven
  script:
    - echo "=====开始部署任务======"
    - "which sshpass || (apt-get update -y && apt-get install sshpass -y)"
    - sshpass -p $PASSWORD scp -o StrictHostKeyChecking=no target/hellodemo-
0.0.1-SNAPSHOT.jar $SSH_USERNAME@$SSH_KNOWN_HOSTS:/home/workspace/hellodemo.jar
    - sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no
$SSH_USERNAME@$SSH_KNOWN_HOSTS "bash /home/workspace/start.sh /home/workspace
hellodemo.jar"
```

变量




[收起](#)

环境变量通过Runner应用于环境中。您可以通过环境变量来存储密码，密钥等等。如果您需要把变量用于最终运行的应用，您可以在变量名前添加 `K8S_SECRET_`。您可以将变量设置为：

- 受保护的 变量仅可用于受保护的分支或标签中。
- 隐藏的 变量在作业日志中隐藏(虽然它们必须符合某些正则表达式要求)。

[更多信息](#)

环境变量已被管理员配置为受保护的(默认)

类型	↑ 键	值	受保护	隐藏	环境	
变量	PASSWORD	*****	×	×	全部(默认)	
变量	SSH_KNOWN_HOSTS	*****	×	×	全部(默认)	
变量	SSH_USERNAME	*****	×	×	全部(默认)	

[显示值](#)[添加变量](#)

对应gitlab-ci.yml文件的\$PASSWORD \$SSH_USERNAME \$SSH_KNOWN_HOSTS

```
#!/bin/bash
# 基础路径
BASE_PATH=$1
# 上传jar包路径
SOURCE_PATH=$BASE_PATH/build
# 应用名称
APP_NAME=$2
# 环境
PROFILES_ACTIVE=$3

DATE=$(date +%Y%m%d%H%M)

# heapError 存放路径
HEAP_ERROR_PATH=$BASE_PATH/heapError
# JVM 参数
JAVA_OPS="-Xms1024m -Xmx1024m -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=$HEAP_ERROR_PATH"
# JavaAgent 参数。可用于配置 skywalking 等链路追踪
JAVA_AGENT=

# 备份代码
function backup() {
    # 如果不存在，则不备份
    if [ ! -f "$BASE_PATH/$APP_NAME.jar" ]; then
        echo "[backup] $BASE_PATH/$APP_NAME.jar 不存在，跳过备份"
    else
        echo "[backup] 开始备份 $APP_NAME ..."
        cp $BASE_PATH/$APP_NAME.jar $BASE_PATH/backup/$APP_NAME-$DATE.jar
        echo "[backup] 备份 $APP_NAME 完成"
    fi
}

# 更新代码
function transfer() {
    echo "[transfer] 开始转移 $APP_NAME.jar"

    # 删除原 jar 包
    if [ ! -f "$BASE_PATH/$APP_NAME.jar" ]; then
```

```

        echo "[transfer] $BASE_PATH/$APP_NAME.jar 不存在, 跳过删除"
    else
        echo "[transfer] 移除 $BASE_PATH/$APP_NAME.jar 完成"
        rm $BASE_PATH/$APP_NAME.jar
    fi

    # 复制新 jar 包
    echo "[transfer] 从 $SOURCE_PATH 中获取 $APP_NAME.jar 并迁移至 $BASE_PATH ...."
    cp $SOURCE_PATH/$APP_NAME.jar $BASE_PATH

    echo "[transfer] 转移 $APP_NAME.jar 完成"
}

# 停止
function stop() {
    echo "[stop] 开始停止 $BASE_PATH/$APP_NAME"
    PID=$(ps -ef | grep $BASE_PATH/$APP_NAME | grep -v "grep" | awk '{print $2}')
    # 如果 Java 服务启动中, 则进行关闭
    if [ -n "$PID" ]; then
        # 正常关闭
        echo "[stop] $BASE_PATH/$APP_NAME 运行中, 开始 kill [$PID]"
        kill -15 $PID
        # 等待最大 60 秒, 直到关闭完成。
        for ((i = 0; i < 60; i++))
        do
            sleep 1
            PID=$(ps -ef | grep $BASE_PATH/$APP_NAME | grep -v "grep" | awk '{print $2}')
            if [ -n "$PID" ]; then
                echo -e ".\c"
            else
                echo '[stop] 停止 $BASE_PATH/$APP_NAME 成功'
                break
            fi
        done

        # 如果正常关闭失败, 那么进行强制 kill -9 进行关闭
        if [ -n "$PID" ]; then
            echo "[stop] $BASE_PATH/$APP_NAME 失败, 强制 kill -9 $PID"
            kill -9 $PID
        fi
    # 如果 Java 服务未启动, 则无需关闭
    else
        echo "[stop] $BASE_PATH/$APP_NAME 未启动, 无需停止"
    fi
}

function start() {
    # 开启启动前, 打印启动参数
    echo "[start] 开始启动 $BASE_PATH/$APP_NAME"
    echo "[start] JAVA_OPS: $JAVA_OPS"
    echo "[start] PROFILES: $PROFILES_ACTIVE"

    # 开始启动
    BUILD_ID=dontKillMe nohup java -server $JAVA_OPS -jar
$BASE_PATH/$APP_NAME.jar --spring.profiles.active=$PROFILES_ACTIVE >
$BASE_PATH/start.log &

```



```

    echo "[start] 启动 $BASE_PATH/$APP_NAME 完成"
}

# 部署
function deploy() {
    cd $BASE_PATH
    # 备份原 jar
    backup
    # 停止 Java 服务
    stop
    # 部署新 jar
    transfer
    # 启动 Java 服务
    start
}

deploy

```

对应gitlab-ci.yml文件中的start.sh

集成代码规范检查

```

mkdir /root/postgres
cd /root/postgres
vim docker-compose.yml

```

```

version: '3'
services:
  postgres:
    image: postgres
    restart: always
    container_name: postgres
    ports:
      - 5432:5432
    environment:
      POSTGRES_PASSWORD: sonar
      POSTGRES_USER: sonar
      POSTGRES_DB: sonar
    volumes:
      - $PWD/postgres.conf:/etc/postgresql/postgresql.conf
      - $PWD/data:/var/lib/postgresql/data

```

```

[root@iz2zeewcelgkyywlttadd5Z postgres]# docker ps
CONTAINER ID   IMAGE     NAMES   COMMAND                                CREATED      STATUS      PORTS
965e6de03dd6   postgres postgres "docker-entrypoint.s..."  2 seconds ago Up 1 second  0.0.0.0:5432->5432/tcp

```

安装成功postgresql

```

mkdir /root/sonarqube
cd /root/sonarqube
vim docker-compose.yml

```

```

version: '3'
services:
  sonarqube:

```

```
image: sonarqube:8.7-community
restart: always
container_name: sonarqube
ports:
  - 9000:9000
environment:
  "sonar.jdbc.username": sonar
  "sonar.dbc.password": sonar
  "sonar.jdbc.url": jdbc:postgresql://39.105.47.81:5432/sonar
volumes:
  - $PWD/conf:/opt/sonarqube/conf
  - $PWD/extensions:/opt/sonarqube/extensions
  - $PWD/logs:/opt/sonarqube/logs
  - $PWD/data:/opt/sonarqube/data
```

启动sonarqube问题解决

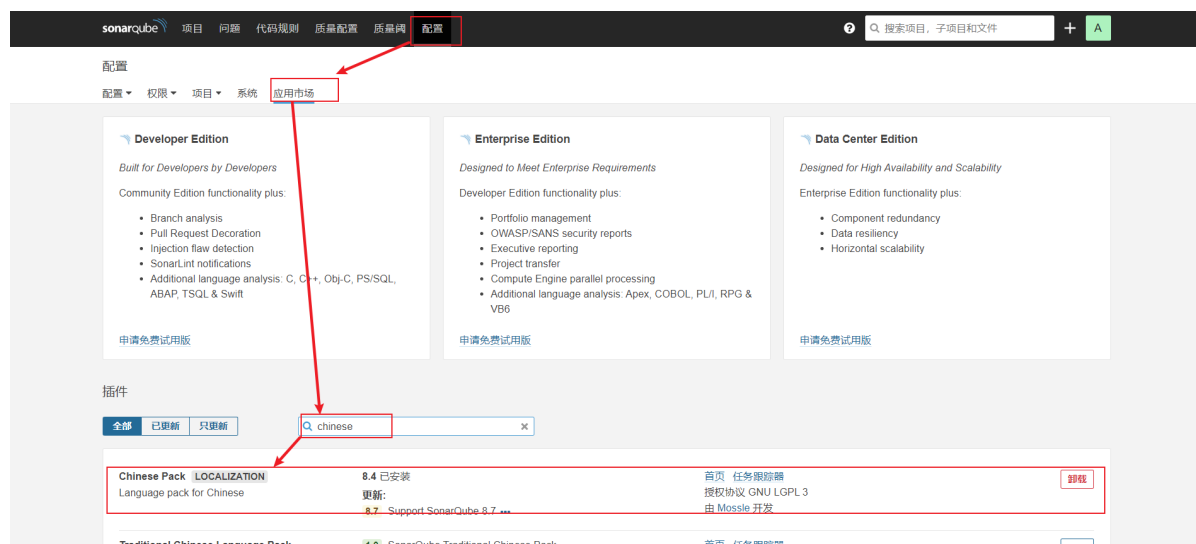
```
vim /etc/sysctl.conf
```

添加

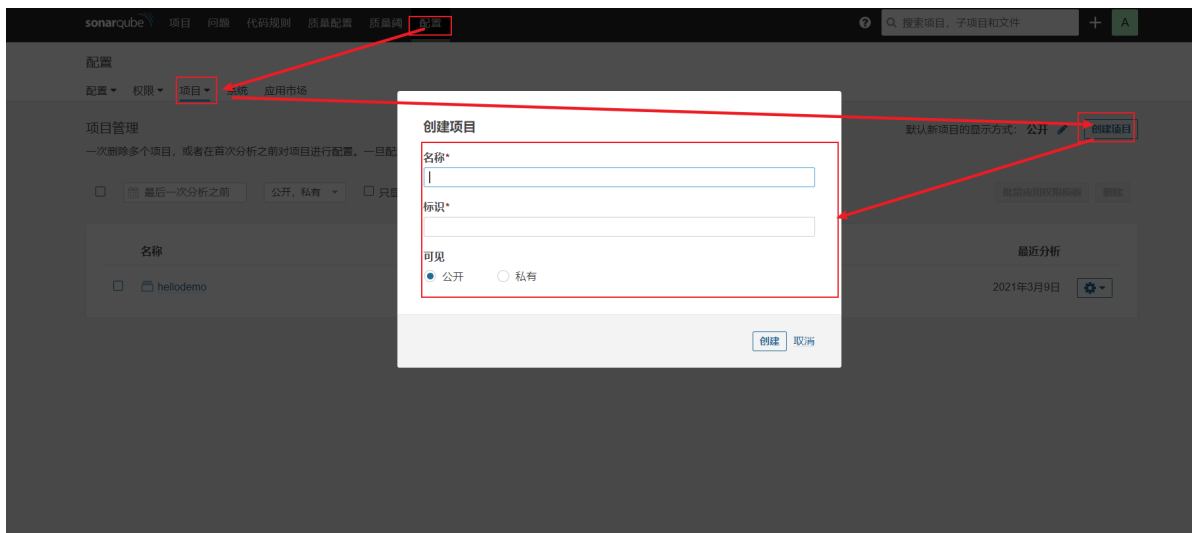
```
vm.max_map_count=262144
```

配置生效

```
sysctl -p
```



重启，显示中文



创建sonarqube项目



创建令牌



选择java maven 复制mvn命令

```
sonar:
  stage: sonar
  tags:
    - maven
  script:
    - echo "=====开始代码检查===== "
    - mvn sonar:sonar -Dsonar.projectKey=hellodemo -
      Dsonar.host.url=http://39.105.47.81:9000 -
      Dsonar.login=7c8174f48e315bcbae1577fbe703920e3bb020c8
```

配置到.gitlab-ci.yml中 提交代码流水线通过

打开sonarqube项目地址，显示代码经过检查

