[Jenkins基础 1](#_Toc943762)

[安装Jenkins 2](#_Toc943763)

[准备git仓库 5](#_Toc943764)

[分发服务器管理 11](#_Toc943765)

CI/CD:持续集成/持续交互

web服务搭建:

ppt是devops\_05

**CICD概述**

Php python是解释执行的，把代码放到运行环境下就行了。

JAVA需要编译。Tomcat里面有个war包。获得git代码的地址，拉去代码，使用ant或者marvin工具把他生成一个测试，测试通过生成一个war包。把war包放到tomcat运行环境，重启tomcat就好了。Tomcat会自动解压，java程序就上线了。CICD需要把这些操作自动完成了。Jenkins把代码打包成war，放到部署服务器上。写个python脚本，定期去检查部署服务器，如果有新的版本，就自动部署新版本。

Gitlab服务器只有一个，广东，杭州等都有服务器。做法是： 每个IDC机房都有一个jenkins服务器，自动拉取gitlab代码，自动编译，生成war包。应用服务器使用python脚本，获取war包，部署war包就行了。

部署上线后，测试部门测试。测试通过之后，上线，测试部门再测试。可以使用docker，保证测试，生产环境是相同环境

Docker使用k8s监控

# Jenkins基础

Jenkins概述

• Jenkins是由java编写的一款开源软件

• 作为一款非常流行的CI(持续集成)工作,用于构建和测试各种项目

• Jenkins 的主要功能是监视重复工作的执行,例如软件工程的构建或在 cron下设置的 jobs

自动化运维的重要工具

持续集成

• 持续集成(CI)是当下最为流行的应用程序开发实践方式

• 程序员在代码仓库中集成了修复bug、新特性开发或是功能革新

• CI工具通过自动构建和自动测试来验证结果。这可以检测到当前程序代码的问题,迅速提供反馈

程序员提交代码更新到软件仓库(SVN/GIT)-------> CI服务器基于计划任务查询仓库,并下载代码------> CI服务器运行构建过程并生成软件包 ------> CI服务器进行单元和集成测试,存储测试结果----如果有错误----> 向开发团队发送构建通知

Jenkins特点

• 简单、可扩展、用户界面友好

• 支持各种SCM(软件配置管理)工具,如SVN、GIT、CVS等

• 能够构建各种风格的项目

• 可以选择安装多种插件

• 跨平台,几乎可以支持所有的平台

## 安装Jenkins

下载网站 **jenkins.io**

1、虚拟机上网，把网卡改成nat的那个网卡

2、将虚拟机的ip地址改为自动获得Automatic

3、将虚拟机的virbr0网卡删除

ifconfig virbr0 down 删虚拟网卡

brctl delbr virbro 删虚拟网桥

[root@node1 ~]# yum -y install jenkins-2.121-1.1.noarch.rpm

安装java依赖包

[root@node1 ~]# yum -y install java-1.8.0-openjdk

启动服务

[root@node1 ~]# systemctl start jenkins

[root@node1 ~]# systemctl enable jenkins

[root@node1 ~]# /sbin/chkconfig jenkins on

**配置Jenkins**

初始化jenkins

firefox访问虚拟机的8080端口. jenkin默认开启8080端口。页面会提示密码写在/var/lib/jenkins/secrets/initialadminpassword下面



安装插件

• 如果网速较快,选择推荐插件,否则选择自定义,只选中GIT即可管理用户（有一个没安装成功，不过我们用不到那么多）

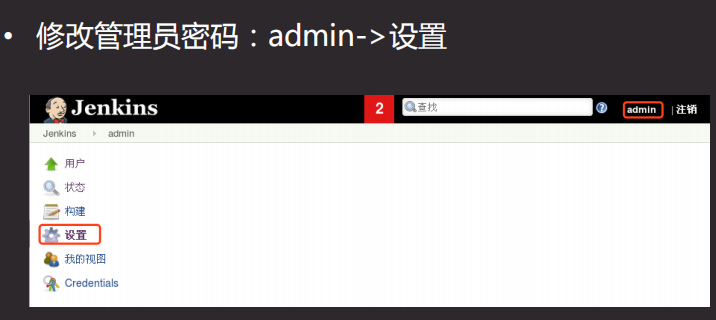


• 可以直接使用Admin登陆



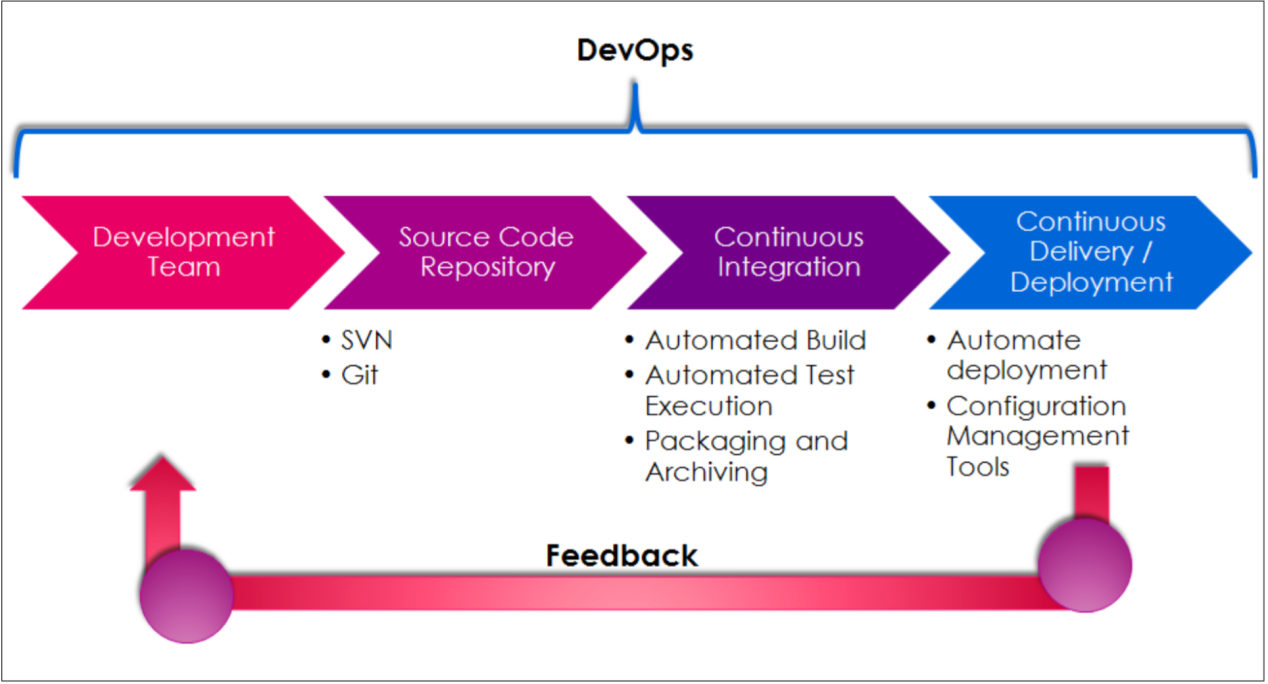


• 修改管理员密码:admin->设置



CI/CD流程

• 程序开发的生命周期内,需要程序员和运维共同协作



过程:

1、程序员在他的电脑上编写程序php（这里用wordpress代替）

2、程序员将写好的代码推到gitlab服务器上

3、运维工程师在jenkins上接取并测试git的代码，生成最终软件包 (\*\*\*.tar.gz)

4、运维工程师在应用服务器上将最终软件包下载并部署到生产服务器

## 准备git仓库

一、配置wordpress

1、初始化wordpress项目

• 解压wordpress 4.8版本

[root@node2 wordpress]# unzip wordpress-4.8-zh\_CN.zip

• 初始化git仓库

[root@node2 wordpress]# cd wordpress/

[root@node2 wordpress]# git init

[root@node2 wordpress]# git add .

[root@node2 wordpress]# git commit -m 'wordpress init'

[root@node2 wordpress]# git status

# 位于分支 master

无文件要提交，干净的工作区

1. 给当前的wordpress打个1.0版本

tag标签

• 如果达到一个重要的阶段,并希望永远记住那个特别的提交快照,可以使用 git tag 给它打上标签

• 将初始化完毕的wordpress打标签v1.0

[root@node2 wordpress]# git tag v1.0tag

[root@node2 wordpress]# git tag

v1.0

3、升级wordpress，将wordpress新版本解压到项目中

[root@node2 wordpress]# cd ..

[root@node2 wordpress]# unzip wordpress-4.9-zh\_CN.zip

4、更新Git仓库

[root@node2 wordpress]# cd wordpress/

[root@node2 wordpress]# git status

[root@node2 wordpress]# git add .

[root@node2 wordpress]# git commit -m 'upgrade to new version'

[root@node2 wordpress]# git status

# 位于分支 master

无文件要提交，干净的工作区

[root@node2 wordpress]# git tag v2.0

• 查看所有标签

[root@node2 wordpress]# git tag

v1.0

v2.0

• 切换至v1.0标签

[root@node2 wordpress]# git checkout v1.0

• 切换到最新状态

[root@node2 wordpress]# git checkout master

之前的 HEAD 位置是 2818b5b... wordpress init

切换到分支 'master'

二、将WordPress推送到gitlab

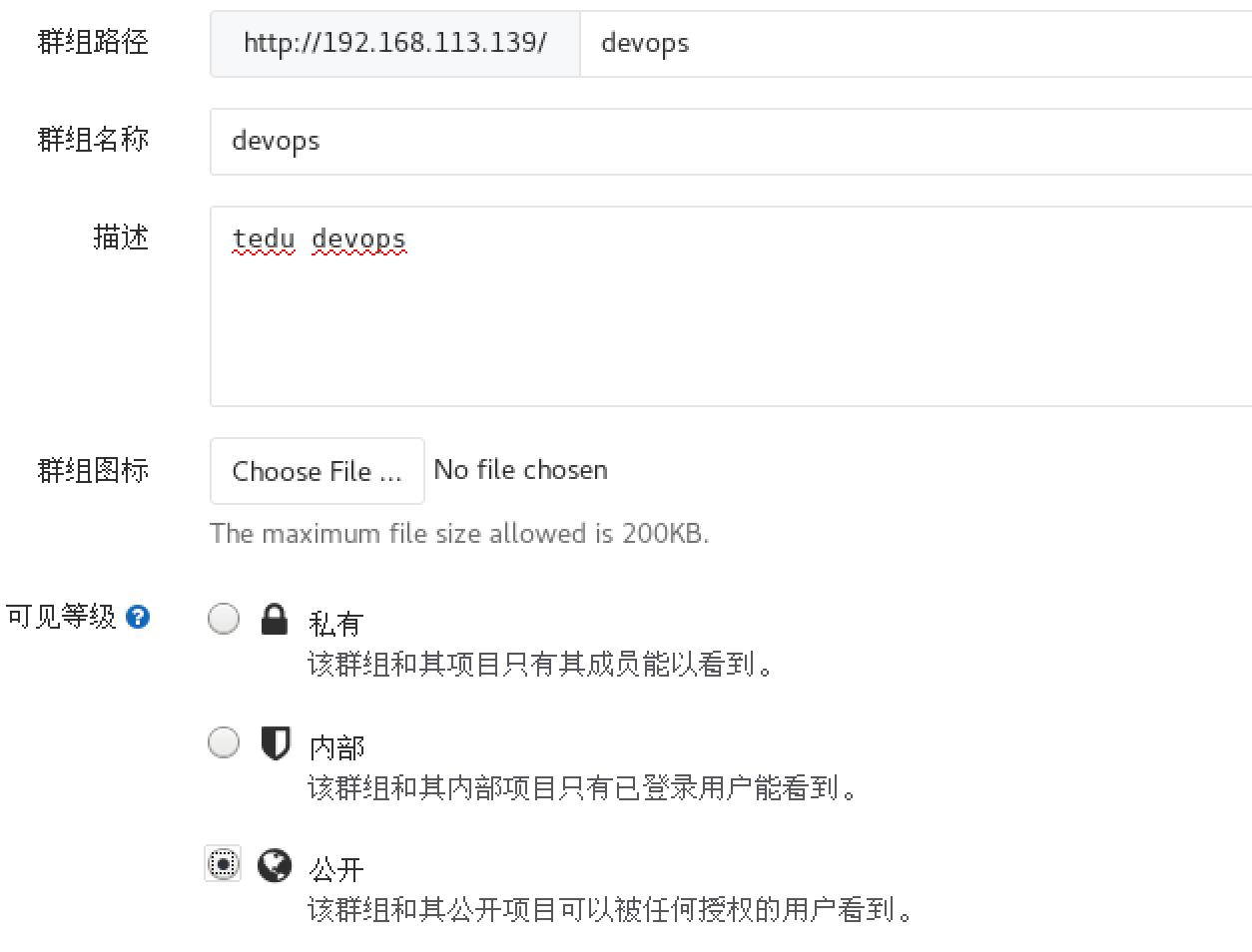
1、远程仓库，配置gitlab

参见ppt

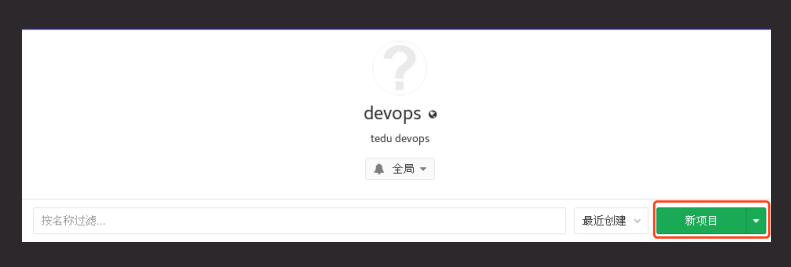
• 创建名为devops的群组



• 创建名为wordpress的项目



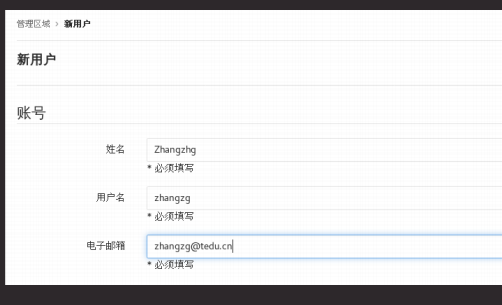
创建wordpress项目

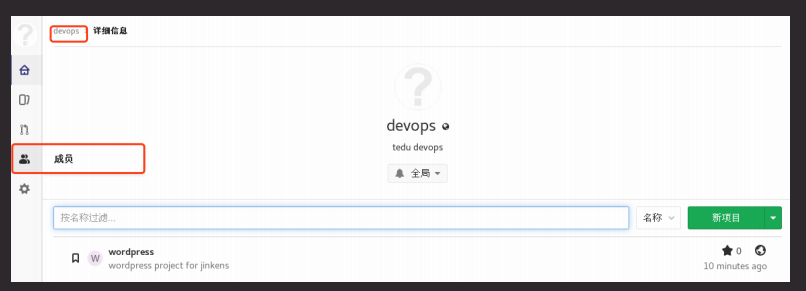




• 为wrodpress项目创建主程序员用户









• 用户生成ssh密钥（需要用户自己登录后添加）

[root@node2 wordpress]# ssh-keygen -t rsa -C 'gaopf@tedu.cn' -b 4096

[root@node2 wordpress]# ssh-agent bash --login -i

[root@node2 wordpress]# ssh-add

[root@node2 wordpress]# cat /root/.ssh/id\_rsa.pub 结果复制到网页的ssh部署里面

1. 推送wordpress代码

注意git推送有两种方式. 1. ssh 需要用户提供秘钥,以后提交就不需要提供用户名密码了2. http.每次提交需要提供用户名,密码. 两种方式的地址是不一样的.git remote add XXX 地址是不一样的

• 因为本地wordpress已经是git版本库了,所以采用以下方式进行上传:

[root@node2 wordpress]# cd wordpress/

[root@node2 wordpress]# git remote rename origin old-origin

error: 不能重命名配置小节 'remote.origin' 到 'remote.old-origin'

上述错误忽略

[root@node2 wordpress]# git remote add origin git@192.168.1.2:devops/wordpress.git

（这里填写gitlab服务器的ip地址，也就是自己的）或者修改hosts文件， 192.168.1.2 gitlab

git remote add origin git@gitlab:devops/wordpress.git

[root@node2 wordpress]# git push -u origin --all

[root@node2 wordpress]# git push -u origin --tags

Total 0 (delta 0), reused 0 (delta 0)

To git@gitlab:devops/wordpress.git

\* [new tag] v1.0 -> v1.0

\* [new tag] v2.0 -> v2.0

3、应用jenkins(jenkins服务器可以每个机房一台)

下载git插件

• 为了使得Jenkins可以使用git的tag,需要下载gitparameter插件

• 点击“系统管理”->“管理插件”

参见ppt

• 新建任务

• 选择自由风格

• 参数化构建过程 添加Git Parameter参数

• 源码采用git 这里如果报错就在jenkins服务器上执行一下那个报错的命令(jenkins服务器上需要安装git)

• 将源码checkout到子目录

• 构建工程

• 选择指定的标签(如果没有标签,检查有没有把标签上传到gitlab)

• 控制台输出，查看日志输出

查看本地结果

• 构建好的项目出现在/var/lib/jinkens目录下

[root@node1 ~]# cd /var/lib/jenkins/workspace/

[root@node1 workspace]# ls wpbuild/wordpress-v1.0/

[root@node1 workspace]# cd wpbuild/wordpress-v1.0/

[root@node1 wordpress-v1.0]# git tag

v1.0

v2.0

[root@node1 wordpress-v1.0]# git log

## 分发服务器管理

优化构建工程

• 在Jenkins服务器上安装apache,用于分发应用程序

• 为了方便应用服务器下载,Jenkins构建的工程应该打包成为一个文件

• 为了应用服务器可以获知下载的程序文件是没有损坏的,应该为其生成md5值

配置分发服务器

• 通过web服务为应用服务器提供应用程序

• 下载目录为/var/www/deploy/packages

[root@node1 wpbuild]# yum -y install httpd

[root@node1 wpbuild]# systemctl restart httpd

[root@node1 wpbuild]# systemctl enable httpd

[root@node1 wpbuild]# mkdir -pv /var/www/html/deploy/packages

[root@node1 wpbuild]# chown -R jenkins.jenkins /var/www/html/deploy/

• 为下载的应用打包,以及生成md5可以能过在工程中增加构建步骤完成

工程里面配置网页 参考ppt

• 检查分发服务器的相关目录

[root@node1 wordpress-v1.0]# cd /var/www/html/deploy/packages/

[root@node1 packages]# ls

wordpress-v1.0.tar.gz wordpress-v1.0.tar.gz.md5

[root@node1 packages]# cat wordpress-v1.0.tar.gz.md5

7b2e0976f947d0d7822a48a34ad8ef2c

创建版本文件

• 创建两个版本文件

– live\_version:表示当前使用版本

– last\_version:表示上一个版本

• 应用服务器可以查看live\_version决定是不是要发布新版本

• 如果新版本有问题,应用服务器可以根据last\_version回滚到前一版本

• 为了记录应用的当前版本和前一个版本,创建两个工程:

– 创建live\_version,记录应用程序当前版本

– 创建last\_version,记录应用程序前一个版本

• 查看构建结果

[root@localhost ~]# ls /var/www/html/deploy/

live\_version packages

[root@localhost ~]# cat /var/www/html/deploy/live\_version

v1.0

自动化部署

服务器规划

• 为了方便版本的切换,可以规划如下目录

– /var/www/download用于存储下载的应用

– /var/www/deploy用于存储解压的应用

• 创建/var/www/html/current软链接,指向需要部署的应用版本

下载应用

• 编写下载应用的功能代码

– 通过位置参数指定要下载的版本

– 位置参数是live下载当前版本

– 位置参数是last下载前一个版本

– 如果已经下载,则不要重复下载

校验文件

• 编写校验文件代码

– 计算指定文件的md5值

– 将md5值与发布服务器提供的md5值进行比较,以确

认下载的文件无误

发布应用

• 编写应用发布代码

– 根据指定的版本,创建/var/www/html/current链接,指向到不同的发布版本

import requests

import os

import hashlib

import tarfile

import re

from urllib.request import urlopen

def get\_pack\_name(version):

'返回下载的软件包url'

version\_url = 'http://192.168.1.1/deploy/%s\_version' % version

r = requests.get(version\_url)

ver =r.text.strip()

url = 'http://192.168.1.1/deploy/packages/wordpress-%s.tar.gz' % ver

return url

def download(url):

'下载软件包,返回软件包的绝对路径'

with urlopen(url) as html :

fname = url.split('/')[-1]

fname = os.path.join('/var/tmp',fname)

with open(fname,'wb') as fobj:

while True:

data = html.read(4096)

if not data:

break

fobj.write(data)

return fname

def check\_md5(fname):

m = hashlib.md5()

with open(fname,'rb') as fobj:

while True:

data = fobj.read(4096)

if not data:

break

m.update(data)

return m.hexdigest()

def check\_package(fname):

'校验软件包是否完好,没问题就返回True,否则返回False'

local\_md5 = check\_md5(fname)

local\_name = os.path.basename(fname)

r = requests.get('http://192.168.1.1/deploy/packages/' + local\_name + '.md5')

remote\_md5 = r.text.strip()

if local\_md5 == remote\_md5 :

return True

return False

def deploy(fname):

'将下载的软件包部署到web服务器'

web\_root = '/var/www'

os.chdir(web\_root)

with tarfile.open(fname,'r:gz') as tar:

tar.extractall()

dst\_fname = fname.replace('.tar.gz', '')

dst\_fname = os.path.basename(dst\_fname)

m = re.search('(\w+)-.\*',dst\_fname)

link = os.path.join('%s/html/' % web\_root,m.groups()[0])

if os.path.islink(link):

os.unlink(link)

os.symlink('%s/%s' % (web\_root,dst\_fname),link)

if \_\_name\_\_ == '\_\_main\_\_':

promt = '''(0) 最新版本

(1) 上一个版本

请选择(0/1)'''

choice = input(promt)

if choice == '0':

version = 'live'

elif choice == '1':

version = 'last'

url = get\_pack\_name(version)

fname = download(url)

fileok = check\_package(fname)

if fileok :

deploy(fname)

else:

print('下载的文件已经损坏,请重新下载')

# 检查项概述

C,c++ : pclint,kora,klocwork,cppcheck

Java: findbugs,pmd,checkstyle,nc,NSIQ

Python: pylint,paycheck

公共： csecheck,sourcemonitor,simian,cct,sharpcount,purufy,metric,checkin,stakcheck,recusion

其他：安全扫描，敏感词汇，二进制合法性，文件完整性，特殊字符检查

LLT ： cunit ,cppunit,cobertura

开发架构：UADP-guarding

开源认证扫描：fossid扫描

完整性校验：signatureVerify部署

数字签名： 数字签名服务器客户端

BEP插件部署： 时间戳，消除BEP部署

冒烟测试： CI集成冒烟测试（前提：准备冒烟测试用例）

全量测试： CI集成全量测试）（前提，准备全量测试用例）

构建结果邮件提醒

分类： push merge daily 全部检查 增量检查

# 部署分布式jenkins环境

## 一、安装JDK

## 二、安装jenkins插件

## 三、 下载crt文件

下载站点crt文件，放在/Jenkins/update-center-root CAS 的目录里面，登录jenkens master主页，在 系统-》管理插件-》高级中，把URL修改为目标地址，点击立即获取

## 四、 slave节点

1. 修改全局配置

TCP port for JNLP 随即

代理协议： java web start agent protocol 3

2. 新建节点。在 系统设置-》节点管理 中选择新建节点，选固定节点

名称： test

并发构建：6

远程工作目录： /Jenkins

标签：linux

用法：只允许运行绑定到这台机器的job

启动方式：通过java web 启动代理 （另一种是通过master命令启动）

标签的作用是可以指定某个任务只能在指定标签的机器上面跑

3. 启动slave

从jenkins master 处下载agent.jar（从新建的节点点进去），并保存到/jenkins下面，修改rc.local文件

Nohup sleep 60 && source /etc/profile && java –Dorg.jenkinsCI.plugin.gitclient.git.timeout=60 -jar /Jenkins/agent.jar -jnlpurl <http://master.ip/:8080/computer/linux5/slave-agent.jnlp> --secret 一串数字 -workdir “/jenkins” &

## 五、 启动jenkins

Master节点： 修改/etc/rc.local

Export JENKINS.HOME=/Jenkins

Nohup java -Dorg.kenjins.plugins.gitclient.git.timeout=60 -jar /usr/share/Jenkins/Jenkins.jar &

解释： 终端关闭后服务还存活 git超时时间为60分，jenkins.jar是从官网下载的，放入后台执行

## 六、 配置gitlab，在push，merge时触发构建

1. 安装插件 gitlab plugin, git server plugins, git plugin,git client plugin

2. 配置git及sshkey

Git config --global user.name=yourname

Git config --global user.email=youremail

3. ssh-keygen 生成米有

4. Jenkins 配置sshkey秘钥

5. gitlab配置秘钥

5. 创建push工程jenkins

Branch to build : \*\*

触发器： build when a change is push to gitlab. Gitlab CI server : <http://master.ip/projectname>

Enable trigger: push Event (或者merge event)

6. codeclub/gitlab 配置对接

Jenkins CI： active

Project URL : XXX

Username: root

Changepassword:

Intergration : push events

Enable SSL verification

## 七、 配置静态检查

### 1. wine

安装wine用于在linux下执行windows程序

### 2. Cppcheck

**Linux下:** 解压cppcheck-tar.gz

Make CFGDIR=/cppcheckcfg

Make install CFGDIR=/cppcheckcfg

**Windows: 有mis安装包**

配置 构建-》执行shell

Cppcheck --enable=all --inconclusive --xml --xml-version=2 CSLB 2 > cpcheck.xml

使用所有检查 待检查的目录 检查结果

构建后-》 pushcppcheckresult

Cppcheck report: report cppcheck.xml

### 3. pclint

Pclint工具是hisi提供的，拷贝到/Jenkins/pclint，并设置LINT-PATH=/Jenkins/pclint

Pclint的命令格式 ：

LINT-NT.EXE env-xml.lint mdc.lint filelist.lint > report.xml

Env-xml.lint: 写在群文件里面的配置参数，可以对所有工程生效

MDC.lint: 本项目的规则文件，包括本项目不需要输出的warning，需要特别输出的warning，头文件路径等。

Filelist.lint 待检查的列表

Env-xml.lnt文件解析

-v //turn off verbsity

-width(0) //don’t insert line breaks

+xml (?xml version=”1.0”) //add version info

+xml(doc) //turn on xml escapes the whole is bracked with <doc> …</doc>

-format=<message><file>

……

**Windows下配置: bat脚本**

Rem “检查规则为mcu的规则文件”

@echo off

Set LINT-PATH=d:\pclint\ //设置工具所在路径

Set S32-SDK-PATH=%projectDirPath \SDK\mcu\SDK //设置工程所在路径

Set S32-STDC-HEADER=c:\xxx\xxx //设置头文件所在路径

@dir %ProjDirPath%\\*.cpp %ProjDirPath%\\*.c %ProjDirPath%\\*.cc /s /b /a :-d >filelist.lnt //生成待检查文件的路径集合

@%LINT-PATH%\LINT-NT.exe env-xml.lnt filelist.lnt | report.xmml

**Linux 略**

### 4. sourcemonitor

命令格式 linux下：

Wine ./2.6/sourcemonitor.exe /c++ sourcemonitor-commands-xml

Sourcemonitor-commands.xml 解析

<?xml version=1.0 encoding=UTF-8 standalone=no?>

<sourcemonitor-commands>

<write-log>true</write-log>

<command>

<projeckfile>/Jenkins/xxx/report/source.smp</projectfile> 指定工程名

<projeck-language>C++</project-language> 指定语言

<source-directory>/Jenkins/xxx</soure-directory>指定待检查的文件路径

<source-sublist>

<exclude-subdirectory>true</exclude-subdirectory> 如果true,则后面的文件被踢出检查，目录下的其他文件全部被包含进来。如果为false，或者确实，则只包含后面的目录

<source.subtree>dependency</source.subtree> 前面的设置对子目录也生效，soutcesubtree 关键字source-subdirectory，则前面的设置对自布鲁不生效。注意，这里的路径必须是反斜杠，或者没有斜杠

</source-subdirectory-list>

<include-subdirectories>true<include-directoried>

<file-extension>\*.h,\*.c,\*.cpp<fileextension>

<export>

<export-file>/jenkins/report/xxx.xml</exportfile>

<exporttype>2(projectdetail as xml)</export.type>

<export-option>include method metric:options</export-option>

</export>

</command>

</sourcemonitor>

### 5. codedex （converity fortify） 前提：编译能通过

Linux下：

1. 删除原有配置。Config目录下（converity）

Rm –rf \*-config-\* covenity-config.xml

1. 重新注册编译器（需要用到的编译器）

Cov-config --gcc

Cov-configure --comptype gcc --compiler arch64-him-ix100-linux-gcc --template

Cov-configure --template --compiler C++ --comtype gcc

重新注册fortify：

修改fortify/core/config下的fortify.sca.properties文件

Com.fortify.projectRoot=/Jenkins/xxx/for-ret 输出存放的目录

Com.fortify.sca.compulers.aarch64-himix100-linux-gcc=com.fortify.sca.util.cmpilers.gcccompiler 查看原有的配置的写法，这是注册成c的编译器

Comfrtify.sca.util.compiler.aarch64-himix100-linux-gcc=com.fortify.sca.util.compiler.gppcompiler注册c++的编译器

Com.fortify.sca.compiler.aarch64-himix100-linux-ld=comfortify.sca.compilers.ldcompiler 注册链接库