

day03

day03

永久关闭防火墙和selinux

CI/CD

配置jenkins

软件版本管理

配置jenkins访问gitlab代码仓库

测试下载

下载到子目录

准备两台web服务器

部署代码到web服务器

自动化部署流程

在Jenkins上配置FTP服务器

配置jenkins把gitlab下载的代码打包

web服务自动部署

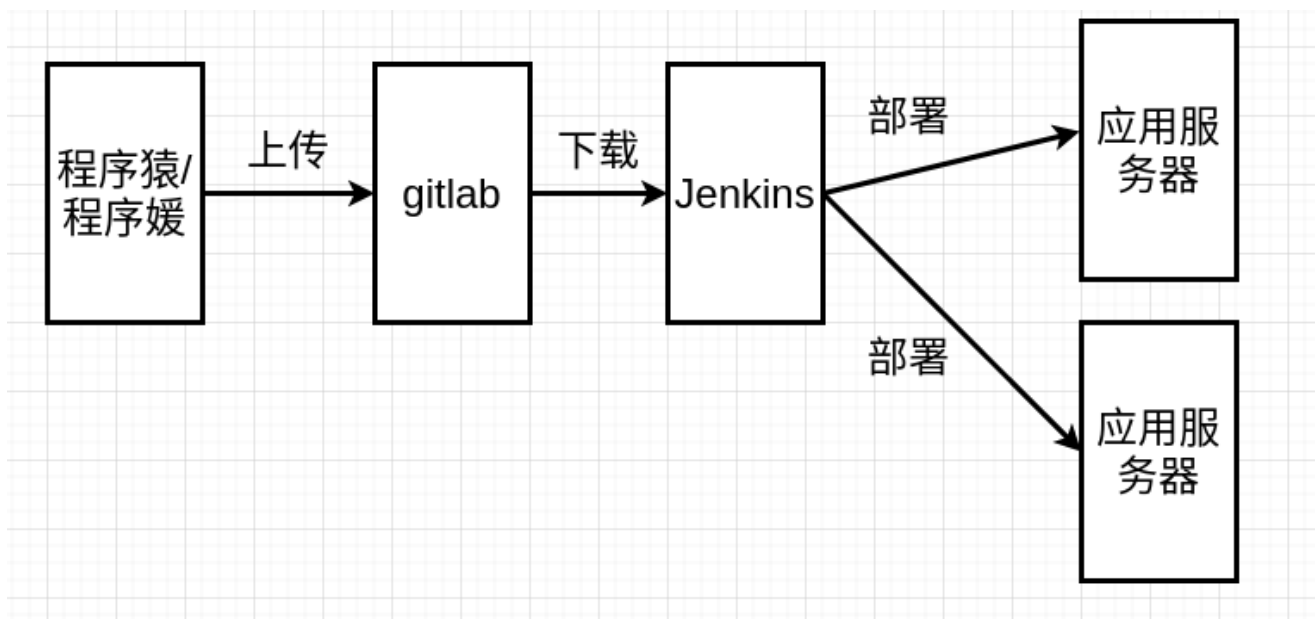
安装httpd服务

编写自动上线脚本

永久关闭防火墙和selinux

```
[root@git ~]# systemctl stop firewalld
[root@git ~]# systemctl disable firewalld
[root@git ~]# setenforce 0
[root@git ~]# vim +7 /etc/selinux/config
7 SELINUX=disabled
```

CI/CD



- 启动之前准备好的机器：192.168.4.10、192.168.4.20、192.168.4.30
- 设置gitlab容器开机自启：/etc/rc.local是开机后会自动运行的脚本，写到这个文件中的命令，开机后都会自动运行

```
[root@git ~]# vim /etc/rc.d/rc.local    # 在文件尾部追加  
一行内容如下:
```

```
... ..
```

```
podman start gitlab
```

```
[root@git ~]# chmod +x /etc/rc.d/rc.local
```

配置jenkins

- 访问<http://192.168.4.30:8080>，用户名是admin
- 安装插件：jenkins的很多功能都是能过插件实现的，比如发邮件、比如中文支持。

```
[root@jenkins ~]# yum install -y tar
```

```
[root@jenkins ~]# tar xf jenkins_plugins.tar.gz
```

```
# 拷贝文件的时候，注意选项，-r可以拷贝目录，-p保留权限
```

```
[root@jenkins ~]# cp -rp jenkins_plugins/*
```

```
/var/lib/jenkins/plugins/
```

```
[root@jenkins ~]# systemctl restart jenkins
```

```
# 刷新web页面，如果出现中文，则插件安装成功
```

软件版本管理

- 可以在git中使用tag标记将某一次commit提交标识为某一版本

```
[root@develop ~]# cd myproject/    # 进入项目目录
[root@develop myproject]# git tag  # 查看标记，默认没有
标记
[root@develop myproject]# git tag 1.0  # 将当前提交，标
识为1.0
[root@develop myproject]# git tag
1.0
[root@develop myproject]# echo 'hello world' >
index.html
[root@develop myproject]# git add .
[root@develop myproject]# git commit -m "add
index.html"
[root@develop myproject]# git tag 1.1

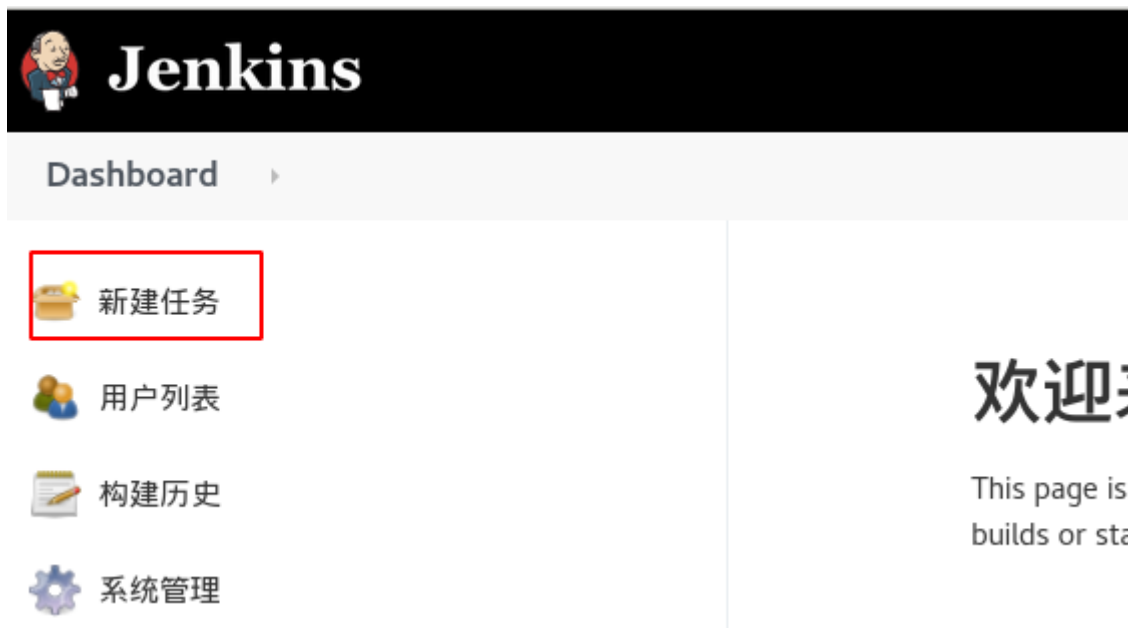
# 将本地文件和tag推送到gitlab服务器
[root@develop myproject]# git push    # 只推送文件，不
推送标记
[root@develop myproject]# git push --tags
```

在gitlab上查看标记：





配置jenkins访问gitlab代码仓库



Dashboard > 所有 >

输入一个任务名称

myproject

» 必填项



构建一个自由风格的软件项目

这是Jenkins的主要功能.Jenkins将会结合任何SCM和任何构建系统来构建你的项目, 甚至可以构建软件以外的系统.

确定

参数化构建过程中, “名称” 是自己定义的变量名, 用于标识tag或分支

General 源码管理 构建触发器 构建 构建后操作

☒ 参数化构建过程

Git 参数

名称 web

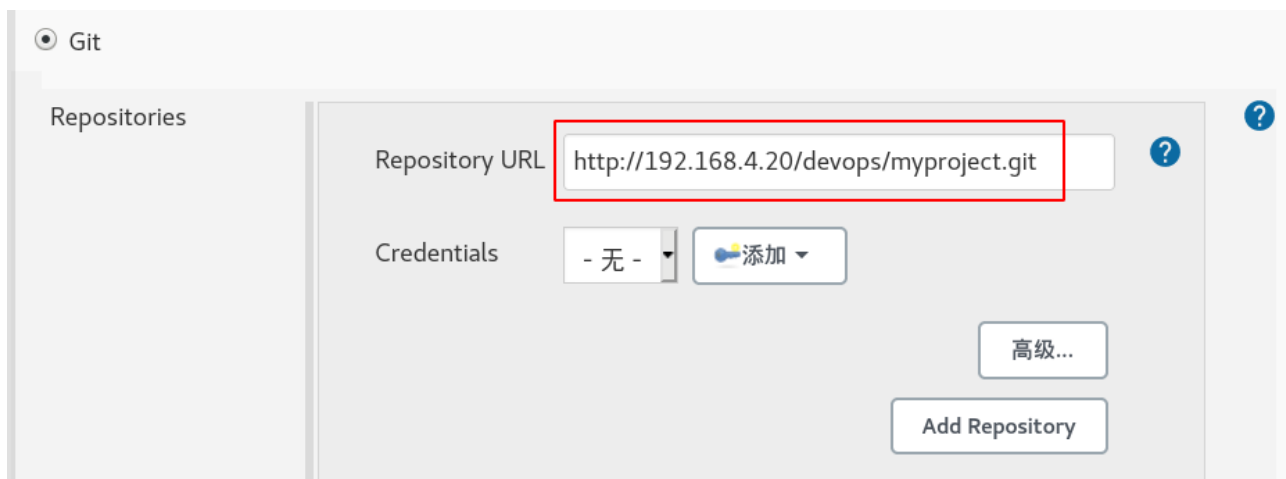
描述

[纯文本] 预览

参数类型 分支或标签

默认值 origin/master

git仓库地址, 在gitlab上找到myproject仓库的http地址, 注意将gitlab名称改为IP地址



指定分支构建的时候，使用上面步骤创建的变量\$web



点击保存。

在项目页面， 可以进行构建测试。



Dashboard

myproject

 返回面板

 状态

 修改记录

 工作空间

 Build with Parameters

 配置

 删除 工程

工程 myproject

需要如下参数用于构建项目:

web

1.0
1.1
origin/master

开始构建

测试下载

 Jenkins

查找



Dashboard

新建任务

用户列表

构建历史

系统管理

我的视图

所有

名称 ↓

上次成功

上次失败

上次持续时间

		myproject	没有	无	无	
---	---	-----------	----	---	---	---

图标:


小 中 大

图例

Atom feed 全部

Atom feed 失败

Dashboard > myproject >

 返回面板

 状态

 修改记录

 工作空间

 Build with Parameters

 配置

 删除 工程

工程 myproject

 工作区

 最新修改记录

相关链接

工程 myproject

需要如下参数用于构建项目:

web


1.0


1.1

origin/master

开始构建

构建过程中，边栏左下角会有一个闪烁的灰球，构建成功是蓝球，失败是红球。点击它，可以看详情。


 工作空间

 Build with Parameters

 配置

 删除 工程

 重命名

 Build History

构建历史 ^

find x

 #1

2021年10月26日 上午11:20

 工作区

 最新修改记录


相关链接

- [最近一次构建\(#1\),17 秒之前](#)
- [最近稳定构建\(#1\),17 秒之前](#)
- [最近成功的构建\(#1\),17 秒之前](#)
- [最近完成的构建\(#1\),17 秒之前](#)



🔍 查找



 返回到工程

 状态集

 变更记录

 控制台输出

 编辑编译信息

 删除构建 '#1'

 参数

构建 #1 (2021年10月26日 上午11:20)

 添



No changes.



启动用户 [admin](#)



Revision: a7143b5eb14a0f8c6b315eea089cdc4686429959

- 1.0

Jenkins

Dashboard > myproject > #1

控制台输出

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/myproject
No credentials specified
Cloning the remote Git repository
Cloning repository http://192.168.4.20/devops/myproject.git
> git init /var/lib/jenkins/workspace/myproject # timeout=10
Fetching upstream changes from http://192.168.4.20/devops/myproject.git
> git --version # timeout=10
> git fetch --tags --progress -- http://192.168.4.20/devops/myproject.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url http://192.168.4.20/devops/myproject.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url http://192.168.4.20/devops/myproject.git # timeout=10
Fetching upstream changes from http://192.168.4.20/devops/myproject.git
```

在jenkins上查看下载的内容：

```
[root@jenkins ~]# ls
/var/lib/jenkins/workspace/myproject/
README.md  hosts  passwd
```

下载到子目录

- jenkins下载不同的版本到自己的子目录，不共享相同目录

Jenkins

Dashboard

新建任务

用户列表

构建历史

系统管理

我的视图

所有 +

S	W	名称 ↓	上次成功	上次失败	上次持续时间
		myproject	2 分 53 秒 - #3	无	90 毫秒

图标: 小 中 大

图例

Atom feed 全部

Atom feed 失败

Atom feed 最新的构建

Dashboard > myproject >

返回面板

状态

修改记录

工作空间

Build with Parameters

配置

删除工程

重命名

工程 myproject

工作区

最新修改记录

相关链接

- 最近一次构建(#3),3 分 20 秒之前
- 最近稳定构建(#3),3 分 20 秒之前
- 最近成功的构建(#3),3 分 20 秒之前

新增时，如果没有中文，英文是“checkout to a sub directory”

Dashboard > myproject >

General 源码管理 构建触发器 构建 构建后操作

Additional Behaviours

新增

检出到子目录

检出到指定的本地分支

高级的克隆行为

高级的子模块行为

高级的检出行为

构建触发器

☐ 触发远程构建 (例如,使用SSH)

☐ 其他工程构建后触发

☐ 定时构建

☐ 轮询 SCM

构建

保存

Additional Behaviours

检出到子目录

仓库的本地子目录 myproject-\$web

新增

点击保存。

测试：

```
# 删除之前下载的内容  
[root@jenkins ~]# rm -rf  
/var/lib/jenkins/workspace/myproject/
```

执行多次构建，构建不同版本：



查看下载目录：

```
[root@jenkins ~]# ls  
/var/lib/jenkins/workspace/myproject/  
myproject-1.0  myproject-1.1
```

准备两台web服务器

- web1: 192.168.4.100, 配置YUM, 关闭SELINUX/防火墙
- web2: 192.168.4.200, 配置YUM, 关闭SELINUX/防火墙

部署代码到web服务器

自动化部署流程

1. 程序员编写代码, 推送到gitlab服务器
2. Jenkins服务器从gitlab上下载代码
3. Jenkins处理下载的代码
 - 删除下载目录的版本库
 - 将下载的代码打包
 - 计算程序压缩包的md5值
 - 在Jenkins上安装ftp服务, 共享程序压缩包
4. web服务器下载软件包, 并应用 (通过脚本实现)
5. 访问测试

在Jenkins上配置FTP服务器

```
# 安装vsftpd
[root@jenkins ~]# yum install -y vsftpd

# 启用ftp的匿名访问
```

```
[root@jenkins ~]# vim +12 /etc/vsftpd/vsftpd.conf  
anonymous_enable=YES
```

起服务

```
[root@jenkins ~]# systemctl enable vsftpd --now
```

ftp的数据目录默认是/var/ftp。

在ftp上创建保存压缩包的路径

```
[root@jenkins ~]# mkdir -p /var/ftp/deploy/packages
```

因为jenkins服务需要向该目录保存文件，所以设置jenkins对它有权限

```
[root@jenkins ~]# chown -R :jenkins /var/ftp/deploy
```

```
[root@jenkins ~]# chmod -R 775 /var/ftp/deploy/
```

配置jenkins把gitlab下载的代码打包

在jenkins上修改myproject项目



命令

```
pkg_dir=/var/ftp/deploy/packages
cp -r myproject-$web $pkg_dir
rm -rf $pkg_dir/myproject-$web/.git
cd $pkg_dir
tar czf myproject-$web.tar.gz myproject-$web
rm -rf myproject-$web
md5sum myproject-$web.tar.gz | awk '{print $1}' > myproject-$web.tar.gz.md5
cd ..
echo -n $web > ver.txt
```

查看 [可用的环境变量列表](#)

```
# 定义存储软件包路径的变量
pkg_dir=/var/ftp/deploy/packages
# 将下载的代码目录拷贝到下载目录
cp -r myproject-$web $pkg_dir
# 删除下载目录的版本库，不是必须的，只是为了严谨
rm -rf $pkg_dir/myproject-$web/.git
cd $pkg_dir # 切换到下载目录
# 将下载的目录打包
tar czf myproject-$web.tar.gz myproject-$web
# 下载目录已打包，目录就不需要了，删除它
rm -rf myproject-$web
# 计算压缩包的md5值，保存到文件
md5sum myproject-$web.tar.gz | awk '{print $1}' >
myproject-$web.tar.gz.md5
cd ..
echo -n $web > ver.txt # 将版本号写入文件
```

以上步骤改好后，保存。

测试修改的任务。

 **Jenkins**

查找


Dashboard > myproject >

 返回面板

 状态

 修改记录

 工作空间

 Build with Parameters

 配置

 删除 工程

工程 myproject

需要如下参数用于构建项目:

web

1.0

1.1

origin/master

开始构建

ftp://192.168.4.30/deploy/packages/

Index of ftp://192.168.4.30/deploy/packages/

Up to higher level directory

Name	Size	Last Modified
File: myproject-1.0.tar.gz	1 KB	2021/10/26 GMT+8 下午3:16:00
File: myproject-1.0.tar.gz.md5	1 KB	2021/10/26 GMT+8 下午3:16:00

web服务自动部署

安装httpd服务

```
[root@web1 ~]# yum install -y httpd tar wget
[root@web1 ~]# systemctl enable httpd --now
[root@web1 ~]# ss -tlnp | grep :80
LISTEN      0          128          *:80
              *:~*
              users:
(("httpd",pid=9721,fd=4),("httpd",pid=9720,fd=4),
("httpd",pid=9719,fd=4),("httpd",pid=9717,fd=4))
```

编写自动上线脚本

- 下载软件包
- 检查软件包是否损坏
- 解压、部署到web服务器

```
[root@web1 ~]# vim /usr/local/bin/web.sh
#!/bin/bash

# 定义软件包服务器和本地路径
ftp_url=ftp://192.168.4.30/deploy
deploy_dir=/var/www/deploy
dest=/var/www/html/tedu-cloud

# 创建用于部署的函数
down_file(){
    # 获取要下载的软件版本
```

```
version=$(curl -s $ftp_url/ver.txt)
# 下载版本文件到本地
wget -q $ftp_url/ver.txt -O $deploy_dir/ver.txt
# 下载软件压缩包
wget -q
$ftp_url/packages/myproject-$version.tar.gz -O
$deploy_dir/myproject-$version.tar.gz
# 计算本地压缩包Md5值
hash=$(md5sum
$deploy_dir/myproject-$version.tar.gz | awk '{print
$1}')
# 获取网上md5文件中的md5值
ftp_hash=$(curl -s
$ftp_url/packages/myproject-$version.tar.gz.md5)
# 如果文件未损坏则解压
if [ "$hash" == "$ftp_hash" ]; then
    tar xf
$deploy_dir/myproject-$version.tar.gz -C $deploy_dir
fi
# 如果存在目标软链接，先删除它
if [ -e "$dest" ]; then
    rm -f $dest
fi
# 创建软链接
ln -s $deploy_dir/myproject-$version $dest
}

# 如果$deploy_dir不存在，先创建它
if [ ! -e "$deploy_dir" ]; then
    mkdir $deploy_dir
fi
```

如果本地不存在版本文件，则意味着是新服务器，要部署软件

```
if [ ! -f $deploy_dir/ver.txt ]; then
    down_file
fi
```

如果本地存在版本文件，但是和服务器的版本文件不一样，则要部署新版本

```
if [ -f $deploy_dir/ver.txt ]; then
    ftp_ver=$(curl -s $ftp_url/ver.txt)
    local_ver=$(cat $deploy_dir/ver.txt)
    if [ "$ftp_ver" != "$local_ver" ]; then
        down_file
    fi
fi
```

fi

```
[root@web1 ~]# chmod +x /usr/local/bin/web.sh
```

```
[root@web1 ~]# yum install -y wget
```

```
[root@web1 ~]# web.sh
```

访问<http://192.168.4.100/tedu-cloud/>可以看到部署的文件

```
[root@web1 html]# ls /var/www/html/
tedu-cloud
```

■ 完整测试流程：

- 程序员编写新版本并推送到服务器
- Jenkins上构建新版本
- web服务器上执行web.sh部署新版本

程序员编写新版本


```
[root@develop myproject]# vim index.html
<marquee>Welcome to tedu</marquee>
[root@develop myproject]# git add .
[root@develop myproject]# git commit -m "modify
index.html"
[root@develop myproject]# git tag 2.0
```

程序员推送到服务器

```
[root@develop myproject]# git push
[root@develop myproject]# git push --tags
```

工程 myproject

需要如下参数用于构建项目:



web 1.0
1.1
2.0
origin/master

开始构建

web服务器上执行`web.sh`部署新版本

```
[root@web1 html]# web.sh
[root@web1 html]# ls /var/www/deploy/
myproject-1.1          myproject-2.0          ver.txt
myproject-1.1.tar.gz  myproject-2.0.tar.gz
# 访问http://192.168.4.100/tedu-cloud
```

