



# Jenkins+Gitlab+K8S 代码自动集成和发布

北京元鼎时代科技股份有限公司

2018-10-30

---

# 目 录

<b>1</b>	<b>规划.....</b>	<b>1</b>
1.1	总体实验流程.....	1
1.2	实验环境.....	2
1.3	本地程序代码说明 .....	3
<b>2</b>	<b>基础环境安装 .....</b>	<b>4</b>
2.1	Docker 及 K8s 集群安装 .....	4
2.2	Docker Registry 安装.....	4
2.2.1	【方式一】安装私有镜像仓库（无密码，免登陆） .....	4
2.2.2	【方式二】安装私有镜像仓库（有密码，需验证） .....	5
2.3	Jenkins 安装 .....	9
2.3.1	执行安装 .....	9
2.3.2	Jenkins 初始化 .....	9
2.3.3	安装插件 .....	13
2.4	Gitlab 安装.....	14
2.4.1	执行安装 .....	14
2.4.2	gitlab 初始化.....	14
2.4.3	Gitlab 中创建项目.....	18
2.4.4	Git 客户端配置 .....	21
2.4.5	确认 gitlab 代码仓库中的内容.....	26
<b>3</b>	<b>Jenkins 配置流水线.....</b>	<b>27</b>
3.1	方案总览.....	27
3.2	全局配置.....	27
3.2.1	配置到 gitlab 代码仓库的连接.....	27
3.2.2	配置到 k8s-master 操作系统的 ssh 连接 .....	28
3.2.3	配置 webhook .....	29
3.3	配置代码同步流水线.....	32
3.3.1	新建任务 .....	32
3.3.2	配置 gitlab 代码仓库 .....	32
3.3.3	构建触发器 .....	34
3.3.4	Gitlab 中配置 webhook.....	34
3.3.5	配置构建环境 .....	36
3.3.6	【方式一】同步代码到 k8s 集群中 .....	37

---

---

3.3.7	【方式二】将代码打包到镜像中 .....	43
3.4	触发自动构建任务 .....	48
4	其他说明.....	49

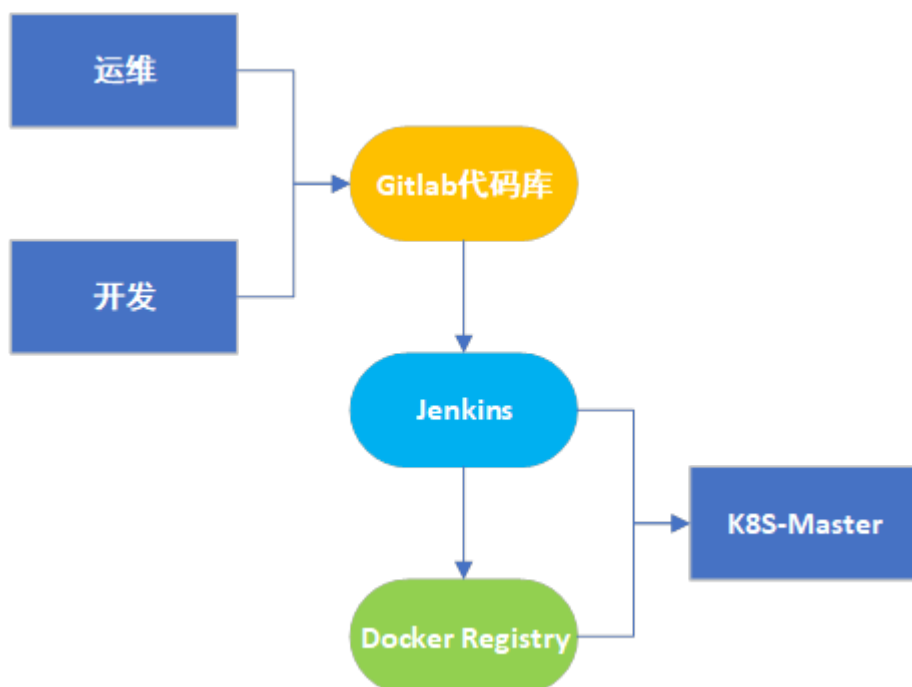
---

---

# 1 规划

## 1.1 总体实验流程

个人电脑作为程序源代码的开发机，开发一个网站程序，当代码提交到 gitlab 代码库之后，可以通过 jenkins 自动识别到代码的变化，将代码同步到 k8s 集群中，并利用 nginx 镜像，在一个容器中运行起来。



- 提交代码

- 开发人员在开发机上开发代码后提交到 gitlab 代码库；
- 运维人员运行该代码所需的环境配置文件（镜像配置-Dockerfile、K8S 资源配置-YAML）提交到 gitlab 代码库；

- 代码构建

- Gitlab 通过 webhook 插件触发 jenkins 进行代码的构建，可以将 gitlab 代码仓库中的代码同步到 jenkins 的工作目录中；

- 根据 Dockerfile 配置，将程序代码打包到 docker 镜像中（也有可能这些代码不需要打包到镜像中，只要容器运行时使用到即可），之后将 docker 镜像传到 docker 镜像仓库中；
- Jenkins 同时可以将 yaml 文件中使用的镜像等变量替换成本次构建时的实际值；

## ● 代码发布

- Jenkins 可以将代码同步到 k8s 集群中的指定目录中；
- 之后在 k8s 集群中根据 yaml 文件，进行相关的资源的创建，从私有镜像仓库拉取镜像，启动容器，实现代码的发布运行。

## 1.2 实验环境

实验环境为 google 云上的一个 k8s 集群

序号	集群节点	IP 地址	用途
1	k8s-master	外部：35.231.203.253 内部：10.142.0.2	K8s 集群的 Master 节点，同时运行本地镜像仓库、jenkins、gitlab /nfs 目录为 nfs 服务
2	K8s-node1	10.142.0.3	挂载来自 k8s-master 的/nfs 目录
3	K8s-node2	10.142.0.5	挂载来自 k8s-master 的/nfs 目录

序号	程序	IP 地址	用途
1	k8s 集群	35.231.203.253:6443	K8s 集群的 Master 节点
2	Jenkins	35.231.203.253:32080	以 docker 方式运行的 jenkins
3	Docker registry	10.142.0.2:5000	以 docker 方式运行的镜像仓库

4	Gitlab	35.231.203.253:32480 10.142.0.2:32480	以 docker 方式运行的代码仓库
---	--------	--	--------------------

## 1.3 本地程序代码说明

本次实验的本地项目 **myweb** 是一个简单的网站程序, 可以使用 **nginx** 运行, 具体结构如下:

```
root@chilone:/mnt/f/localgit# tree myweb/
myweb/
├── README.md
├── dockerfile
│   └── Dockerfile
├── src
│   ├── 404.html
│   ├── css
│   │   ├── css.css
│   │   ├── font-awesome.min.css
│   │   └── style.css
│   ├── images
│   │   ├── image1.jpg
│   │   ├── image5.jpg
│   │   ├── image6.jpg
│   │   └── image6.jpg-bak
│   ├── index.html
│   └── js
│       ├── jquery.min.js
│       ├── respond.min.js
│       ├── skel.min.js
│       └── util.js
└── yaml
    ├── k8s-myweb.yaml
    ├── k8s-myweb1.yaml
    └── pv-myweb.yaml

6 directories, 18 files
root@chilone:/mnt/f/localgit#
```

其中 **src** 目录中是网站的代码程序, **dockerfile** 是存放建立镜像的 **Dockerfile**, **yaml** 中是 **k8s** 资源的文件。具体文件内容后续进行说明。

---

## 2 基础环境安装

### 2.1 Docker 及 K8s 集群安装

Docker 及 K8S 集群安装方式略。

### 2.2 Docker Registry 安装

私有镜像仓库有两种形式，一种是无密码、免登录的，属于不安全的镜像仓库，另外一种是有安全验证措施的，属于安全的镜像仓库。下面分别进行介绍。

#### 2.2.1 【方式一】安装私有镜像仓库（无密码，免登陆）

##### 2.2.1.1 安装镜像仓库服务器端

在 k8s-master 上安装私有的 docker 镜像仓库，挂载/nfs/docker\_registry 目录给镜像仓库，该镜像仓库没有设置登录密码，可以直接使用：

```
# mkdir -p /nfs/docker_registry
# docker run -d -p 5000:5000 -v /nfs/docker_registry:/var/lib/registry --restart=always --privileged=true --name local_registry registry
```

##### 2.2.1.2 docker 客户端将仓库加入不安全镜像仓库列表

由于本次实验中没有为镜像仓库设置登录密码及其他安全验证方式，且使用的是 http 协议，因此在其他服务器使用该私有镜像仓库时，需要调整 docker 的配置参数，将私有仓库的地址加入到不安全仓库列表中，之后可以直接使用该镜像仓库中的镜像，具体方法如下：

```
修改 /etc/docker/daemon.json 文件，加入一行 "insecure-registries": ["10.142.0.2:5000"]
如果该文件中已经有其他的内容，在文件末尾的“}”之前加入这一行，同时上一行的行尾要加上分号“;”。
# cat /etc/docker/daemon.json
{
  "insecure-registries": ["192.168.50.120:5000"]
}
```

```
}
```

重启 docker 进程

```
# systemctl daemon-reload
```

```
# systemctl restart docker
```

### 2.2.1.3 访问私有镜像仓库

这种无验证的镜像仓库，可以直接访问，采用此方式时，私有镜像仓库的访问地址为：10.142.0.2:5000

之后将镜像上传到私有镜像仓库的步骤如下：

为镜像打 tag

```
# docker tag image_name:version 10.142.0.2:5000/image_name:version
```

将镜像上传到私有镜像仓库

```
# docker push 10.142.0.2:5000/image_name:version
```

## 2.2.2 【方式二】安装私有镜像仓库（有密码，需验证）

### 2.2.2.1 安装镜像仓库服务器端

在 k8s-master 上安装私有的 docker 镜像仓库，挂载/nfs/docker\_registry 目录给镜像仓库，该镜像仓库设置有登录密码：

如果创建了签名证书，且启动 docker registry 时需要用证书验证，那么需要将镜像仓库的域名 local-registry 进行解析，且要将 registry.crt 证书放到/etc/docker/certs.d/local-registry:5000/目录下。且登录时必须使用域名登录。



---

创建存放密码的目录

```
# mkdir -p /nfs/docker_registry/auth
```

创建存放证书的目录

```
# mkdir -p /nfs/docker_registry/certs
```

创建存放数据的目录

```
# mkdir -p /nfs/docker_registry/data
```

创建登录的用户名和密码: chilone/12345678 , 存放到/nfs/docker\_registry/auth/htpasswd 文件中

```
# docker run --entrypoint htpasswd registry -Bbn chilone 12345678 > /nfs/docker_registry/auth/htpasswd
```

创建签名证书, 存放到/nfs/docker\_registry/certs 目录中。其中/CN=local-registry/ 是私有镜像仓库的域名, 后续需要通过 dns 或者/etc/hosts 解析。

```
# openssl req -x509 -days 3650 -subj '/CN=local-registry/' -nodes -newkey rsa:2048 -keyout /nfs/docker_registry/certs/registry.key -out /nfs/docker_registry/certs/registry.crt
```

```
# docker run -d -p 5000:5000 --restart=always \
--name local_registry \
-v /nfs/docker_registry/auth:/auth \
-e "REGISTRY_AUTH=htpasswd" \
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry on local-registry" \
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
-v /nfs/docker_registry/certs:/certs \
-v /nfs/docker_registry/data:/var/lib/registry \
-e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/registry.crt \
-e REGISTRY_HTTP_TLS_KEY=/certs/registry.key \
registry
```

在所有需要使用私有镜像仓库的 docker 客户端服务器上设置如下内容，保证可以正常登录使用私有镜像仓库。【本次实验中，需要在 k8s-master、k8s-node1、k8s-node2 服务器上设置】

添加/etc/hosts 解析（如果有 DNS 解析，则在 DNS 中配置，不用在此配置）

将上面创建证书时/CN=**local-registry**/中设置的域名解析成主机的 IP。

```
# vi /etc/hosts
```

```
10.142.0.2    local-registry
```

创建客户端证书目录

```
# mkdir -p /etc/docker/certs.d/local-registry:5000/
```

复制证书 registry.crt 到客户端证书目录中

K8s-master 本机复制：

```
# cp /nfs/docker_registry/certs/registry.crt /etc/docker/certs.d/local-registry:5000/registry.crt
```

跨主机复制，在对应的主机上操作：

```
# scp 10.142.0.2:/nfs/docker_registry/certs/registry.crt /etc/docker/certs.d/local-registry:5000/registry.crt
```

### 2.2.2.1 访问私有镜像仓库

Docker 镜像仓库的地址是 local-registry:5000。

之后将镜像上传到私有镜像仓库的步骤如下：

①登录镜像仓库，之后输入用户名 chilone，密码 12345678

```
# docker login local-registry:5000
```

Username: chilone

Password:

Login Succeeded

一旦登录成功之后，将在 ~/.docker/config.json 文件中添加相关的认证记录

```
# cat ~/.docker/config.json
```

```
{
  "auths": {
    "local-registry:5000": {
      "auth": "Y2hpbG9uZToxMjM0NTY3OA=="
    }
  },
  "HttpHeaders": {
```

```
"User-Agent": "Docker-Client/17.12.1-ce (linux)"  
}
```

②创建允许 k8s 集群中 ci 命名空间可以下载镜像的 secret，名称为 **regcred**，该 secret 将在 yaml 文件的 **ImagePullSecrets** 参数中用到

```
# kubectl create secret docker-registry regcred --namespace=ci --docker-server=http://local-registry:5000 --docker-username=chilone --docker-password=12345678 --docker-email=zhangzhilong@yuandingit.com
```

注意：如果在 jenkins 中使用的话，写好配置后，会自动创建这个 secret。

③为镜像打 tag

```
# docker tag image_name:version local-registry:5000/image_name:version
```

④将镜像上传到私有镜像仓库

```
# docker push local-registry:5000/image_name:version
```

⑤登出，一旦登出之后，则会删除 ~/.docker/config.json 文件中相关的认证记录

```
# docker logout local-registry:5000
```

```
Removing login credentials for local-registry:5000
```

---

## 2.3 Jenkins 安装

### 2.3.1 执行安装

在 k8s-master 上安装 jenkins:

```
# docker run \  
--name jenkins \  
-u root \  
-d \  
--restart always \  
-p 32080:8080 \  
-p 50000:50000 \  
-v /nfs/jenkins:/var/jenkins_home \  
-v /var/run/docker.sock:/var/run/docker.sock \  
jenkinsci/blueocean
```

Jenkins 的访问地址是: <http://35.231.203.253:32080>

### 2.3.2 Jenkins 初始化

#### 2.3.2.1 解锁 jenkins

浏览器访问: <http://35.231.203.253:32080> , 可以看到 jenkins 正在启动中的提示, 等待启动完成, 进入解锁 jenkins 页面, 可执行如下命令找到管理员密码:

```
# docker logs Jenkins  
找到如下的内容: b7818539ba1642c382510214256bc87b 就是解锁密码  
  
*****  
*****  
*****  
  
Jenkins initial setup is required. An admin user has been created and a password generated.
```

Please use the following password to proceed to installation:

**b7818539ba1642c382510214256bc87b**

This may also be found at: `/var/jenkins_home/secrets/initialAdminPassword`

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*

入门

## 解锁 Jenkins

为了确保管理员安全地安装 Jenkins，密码已写入到日志中（[不知道在哪里?](#)）该文件在服务器上：

`/var/jenkins_home/secrets/initialAdminPassword`

请从本地复制密码并粘贴到下面。

管理员密码

### 2.3.2.2 安装默认插件

选择安装推荐的插件，安装完成之后，可以正式进入 jenkins



# 自定义Jenkins

插件通过附加特性来扩展Jenkins以满足不同的需求。

## 安装推荐的插件

安装Jenkins社区推荐的插件。

## 选择插件来安装

选择并安装最适合的插件。

## 新手入门



<input checked="" type="checkbox"/> Folders Plugin	<input type="checkbox"/> OWASP Markup Formatter Plugin	<input type="checkbox"/> Build Timeout	<input type="checkbox"/> Credentials Binding Plugin
<input type="checkbox"/> Timestamper	<input type="checkbox"/> Workspace Cleanup	<input type="checkbox"/> Ant	<input type="checkbox"/> Gradle
<input checked="" type="checkbox"/> Pipeline	<input type="checkbox"/> GitHub Branch Source Plugin	<input checked="" type="checkbox"/> Pipeline: GitHub Groovy Libraries	<input type="checkbox"/> Pipeline: Stage View
<input type="checkbox"/> Git plugin	<input type="checkbox"/> Subversion	<input type="checkbox"/> SSH Slaves	<input checked="" type="checkbox"/> Matrix Authorization Strategy Plugin
<input type="checkbox"/> PAM Authentication	<input type="checkbox"/> LDAP	<input type="checkbox"/> Email Extension	<input type="checkbox"/> Mailer Plugin

Folders

### 2.3.2.3 创建管理员用户

创建一个管理员用户，或者使用 admin 用户。

我这里选择创建了一个管理员用户，【用户名：zzl，密码：12345678，邮箱：zhangzhilong@yuandingit.com】

新手入门

创建第一个管理员用户

用户名:

密码:

确认密码:

全名:

电子邮件地址:

Jenkins 2.138.2

使用admin账户继续

保存并完成

### 2.3.2.4 确认 jenkins 的 URL

最后确认 jenkins 的登录 URL：<http://192.168.50.120:8080/>，当出现“Jenkins 已就绪”，点击“开始使用 jenkins”即可完成初始化。

新手入门

实例配置

Jenkins URL:

Jenkins URL 用于给各种Jenkins资源提供绝对路径链接的根地址。这意味着对于很多Jenkins特色是需要正确设置的，例如：邮件通知、PR状态更新以及提供给构建步骤的BUILD\_URL环境变量。

推荐的默认值显示在尚未保存，如果可能的话这是根据当前请求生成的。最佳实践是要设置这个值，用户可能会需要用到。这将会避免在分享或者查看链接时的困惑。

Jenkins 2.138.2

现在不要

保存并完成

# Jenkins已就绪!

Jenkins安装已完成。

开始使用Jenkins

Jenkins 2.138.2

## 2.3.3 安装插件

由于本次实验使用的 **gitlab** 作为代码仓库，使用 **docker** 镜像，使用 **k8s** 集群，因此需要安装这三者相关的插件。如下：

序号	名称	用途
1	Gitlab	配置 gitlab 的代码仓库。
2	Gitlab hook	Gitlab 的 web hook，当代码仓库中代码有变化的时候，会自动触发 jenkins 构建任务。
3	Publish over ssh	可以将代码文件同步到 ssh 服务器（如 k8s 的 master）上。
4	kubernetes continuous deploy	配置 k8s 集群的信息，自动在集群上发布 k8s 资源。
5	CloudBees Docker Build and Publish	允许进行 docker 镜像的打包操作
6	Kubernetes	允许 jenkins 通过在 kubernetes 运行 slave pod 来执行部署任务。
7	Kubernetes Cli	可在 jenkins 中使用 kubernetes 命令

插件的安装步骤如下：

“系统管理” → “插件管理” → “可选插件”（选择对应的插件）→ “直接安装”。



---

## 2.4 Gitlab 安装

### 2.4.1 执行安装

以 docker 方式运行 gitlab，执行如下命令：

```
建立本地的目录
#
mkdir -p /nfs/gitlab/config
mkdir -p /nfs/gitlab/data
mkdir -p /nfs/gitlab/logs

运行 docker 命令，可以将容器的配置文件都放到刚才建立的目录中
# docker run --detach \
    --hostname gitlab.example.com \
    --publish 32443:443 --publish 32480:80 --publish 32422:22 \
    --name gitlab \
    --restart always \
    --volume /nfs/gitlab/config:/etc/gitlab \
    --volume /nfs/gitlab/logs:/var/log/gitlab \
    --volume /nfs/gitlab/data:/var/opt/gitlab \
    gitlab/gitlab-ce:latest
```

Gitlab 安装之后，需要几分钟才能正式启动，正式启动之后的访问地址是 <http://35.231.203.253:32480>

### 2.4.2 gitlab 初始化

#### 2.4.2.1 更改管理员用户密码

当可以访问 <http://35.231.203.253:32480> 之后，第一个页面是要求更改用户的密码。输入新的密码（本次实验设置的密码是 12345678）并确认，之后才能够到登陆或注册的页面。

【有的时候不会出现该界面，因此就没有更改管理员的密码。默认的管理员

用户是 `admin@example.com` ，默认密码是 `5iveL!fe`】



Please create a password for your new account.

## GitLab Community Edition

### Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

#### Change your password

New password

Confirm new password

Change your password

Didn't receive a confirmation email? [Request a new one](#)

Already have login and password? [Sign in](#)

### 2.4.2.2 注册或登陆

可以使用默认的管理员用户登录或者注册新的用户登录。

本次实验中注册了一个新的用户：`zzl`，密码：`12345678`



## GitLab Community Edition

### Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Sign in

Register

Username or email

Password

☐ Remember me

[Forgot your password?](#)

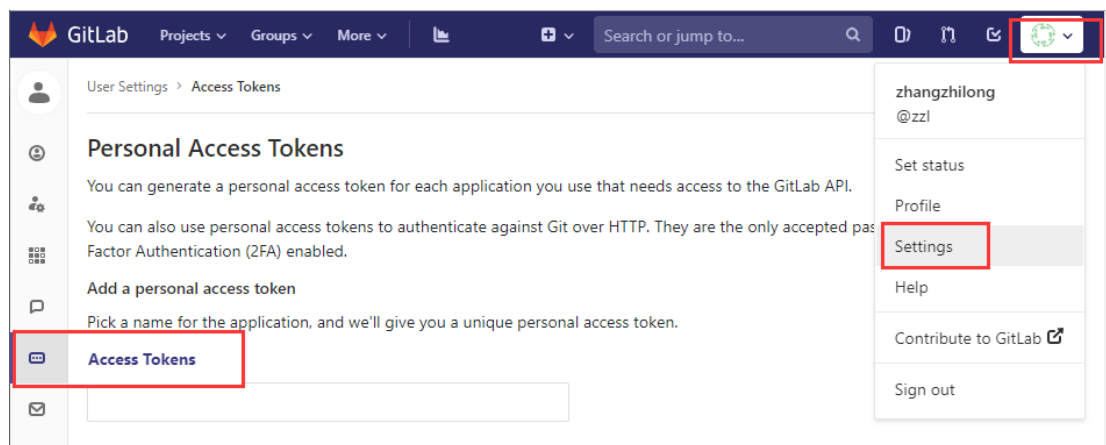
Sign in

### 2.4.2.3 获取用户的 token

后续 `jenkins` 配置与 `gitlab` 的连接时，需要用到 `gitlab` 用户的 `token`，因此

在这里建立 token 并记录下来【一定要记录下来，因为创建之后关掉页面就再也看不到 token 的值了】，本次实验获取的 zzi 用户的 token 值是“xoZbZrGBfxAHFjqZNDnH”。

在页面右上角点击用户图标→“settings”→“Access Token”，之后填写 token 的名称，勾选为“API”，然后创建 token。



You can also use personal access tokens to authenticate against Git over HTTP. The Factor Authentication (2FA) enabled.

Pick a name for the application, and we'll give you a unique personal access token.

jenkins

YYYY-MM-DD

☒ api

Grants complete read/write access to the API, including all groups and projects.








☐ read user

Grants read-only access to the authenticated user's profile through the `/user AP` full name. Also grants access to read-only API endpoints under `/users`.

☐ read\_repository

Grants read-only access to repositories on private projects using Git-over-HTTP

## Create personal access token



Your new personal access token has been created.

## Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the repository.

You can also use personal access tokens to authenticate against Git over HTTP. They are the same as OAuth2 tokens if Two Factor Authentication (2FA) is enabled.

**Your New Personal Access Token**

xoZbZrGBfxAHFjqZNDnH

Make sure you save it - you won't be able to access it again.

You can generate a personal access token for each application you use that needs access to the

You can also use personal access tokens to authenticate against Git over HTTP. They are the only way to authenticate if Two Factor Authentication (2FA) is enabled.

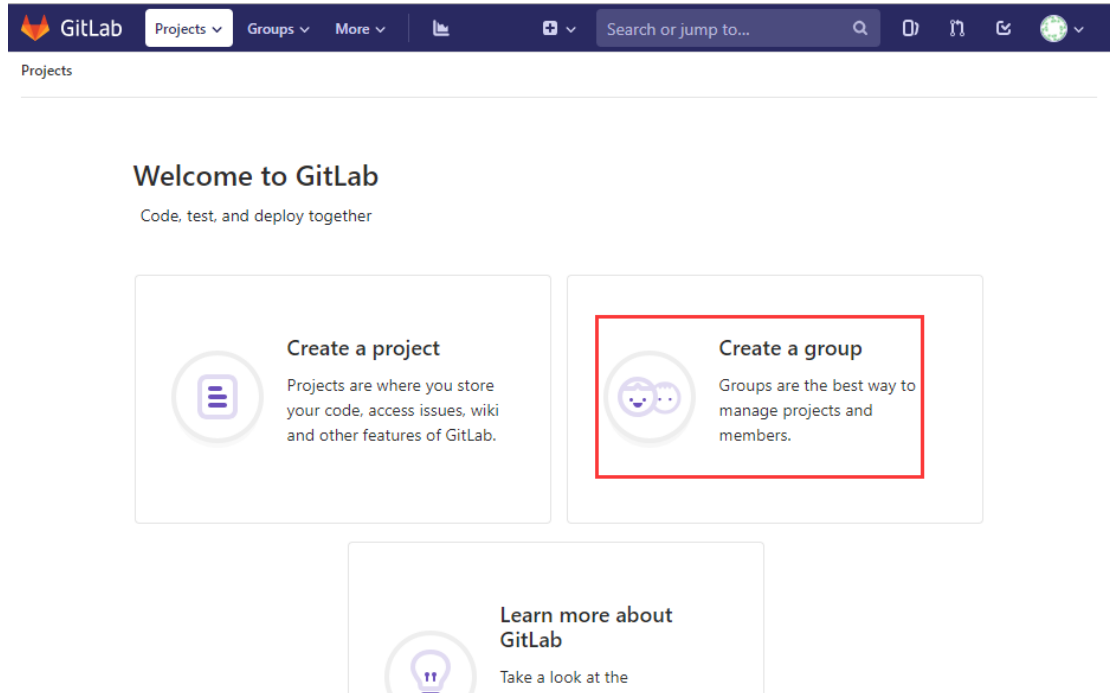
xoZbZrGBfxAHFjqZNDnH

Make sure you save it - you won't be able to access it again.


---

### 2.4.3 Gitlab 中创建项目



首先创建一个组“CICD”，表明这个组是用于做 cicd 实验的，在 cicd 组内创建一个名为“myweb”的项目。









 **GitLab**

Projects ▾ Groups ▾ More ▾

  ▾ Search or jump to...

    ▾

Projects

## New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), [among other things](#).

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

**Tip:** You can also create a project from the command line. [Show command](#)

Blank project

Create from template

Import project

Project name

myweb

Project URL

http://35.231.203.253:32480/

cicd ▾


Project slug

myweb

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)

Description format

Visibility Level 

☐ Private

Project access must be granted explicitly to each user.

☒ Internal

The project can be accessed by any logged in user.

☐ Public

This project cannot be public because the visibility of cicd is internal. To make this project public, you must first [change the visibility](#) of the parent group.


☒ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

记录下这个项目的 gitlab 地址，本次实验的是 <http://gitlab.example.com/cicd/myweb.git>，这个域名是不能用的，所以需要换成 IP 地址，则可用的 gitlab 地址是 <http://35.231.203.253:32480/cicd/myweb.git>

M myweb Internal  Add license

Project ID: 1

0

☆ Star

0

🍴 Fork

HTTP ▾ http://gitlab.example.c



 ▾

 ▾

 Global ▾

Readme

Files (0 Bytes)

Commits (0)

Branch (1)

Tags (0)

Auto DevOps enabled

Add Changelog

Add Contribution guide

Add Kubernetes cluster

## 2.4.4 Git 客户端配置

### 2.4.4.1 Git 客户端安装

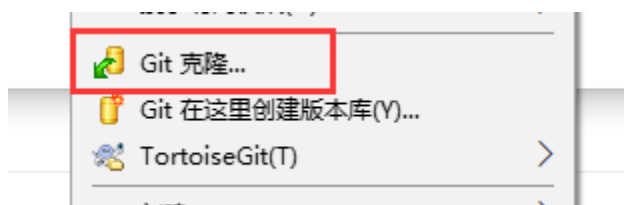
在本地的个人电脑上安装 git 客户端，之后可以进行代码的开发和上传工作。

本次实验中安装了三个软件，如下：

序号	软件名称	用途
1	Git	Git 的 UI 和 BASH 工具
2	TortoiseGit	针对 git 的 Windows 窗口方式界面工具
3	TortoiseGit-LanguagePack	TortoiseGit 的中文语言包

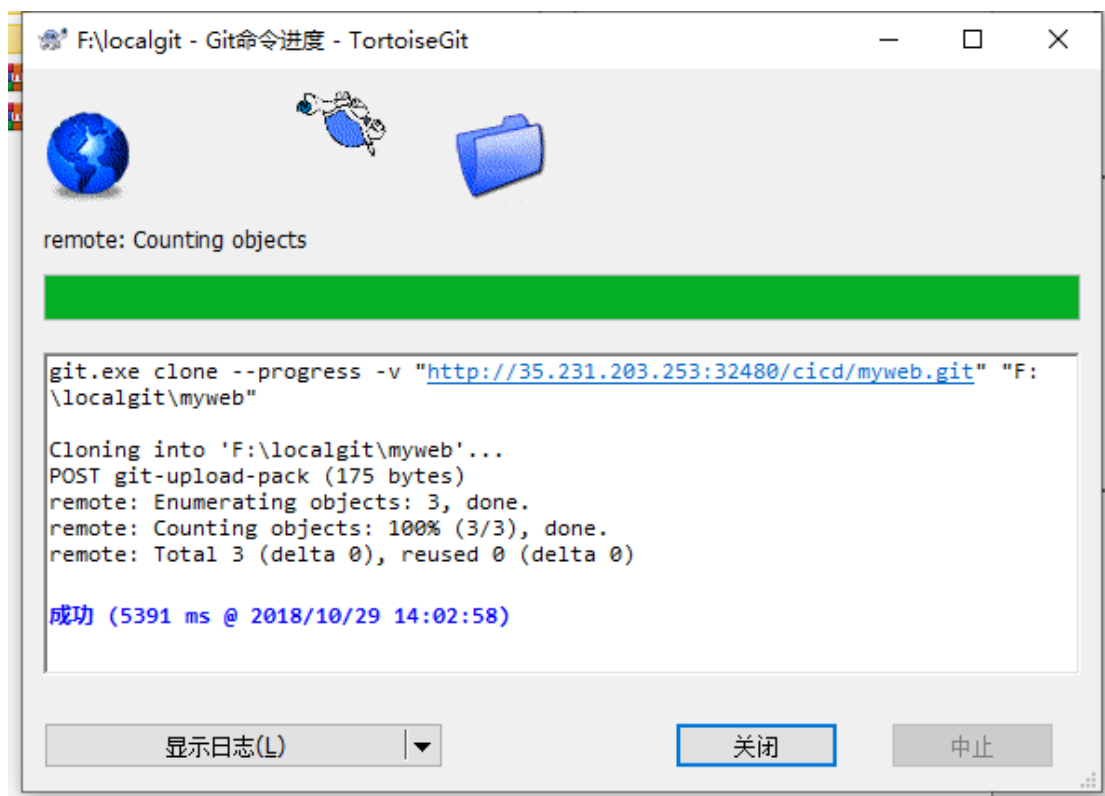
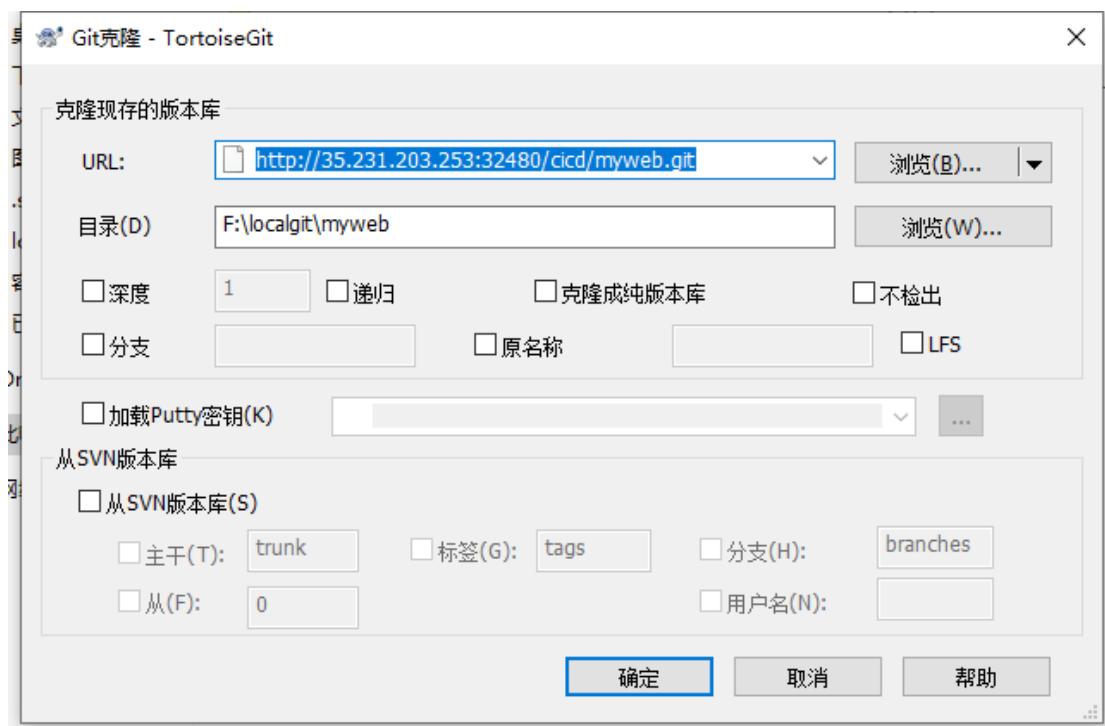
### 2.4.4.2 克隆远端代码仓库到本地

进入到本地电脑的需要作为代码仓库的目录中，点右键“git 克隆”将上面创建的远程代码仓库克隆到本地，形成本地的代码仓库。



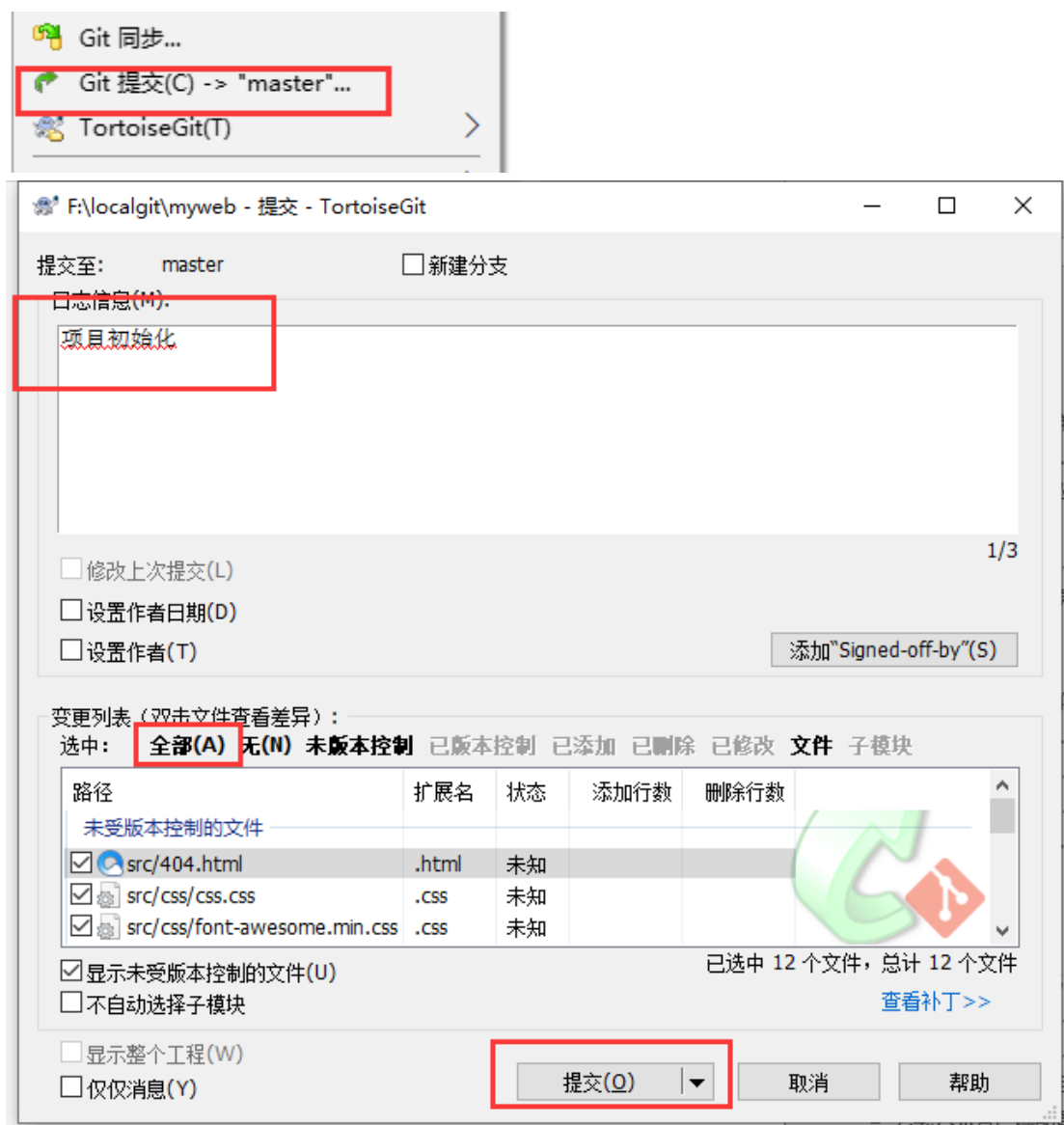
URL 填写上面创建的那个 myweb 项目的地址：<http://35.231.203.253:32480/cicd/myweb.git>。会在本地的目录中创建一个 myweb 的目录。

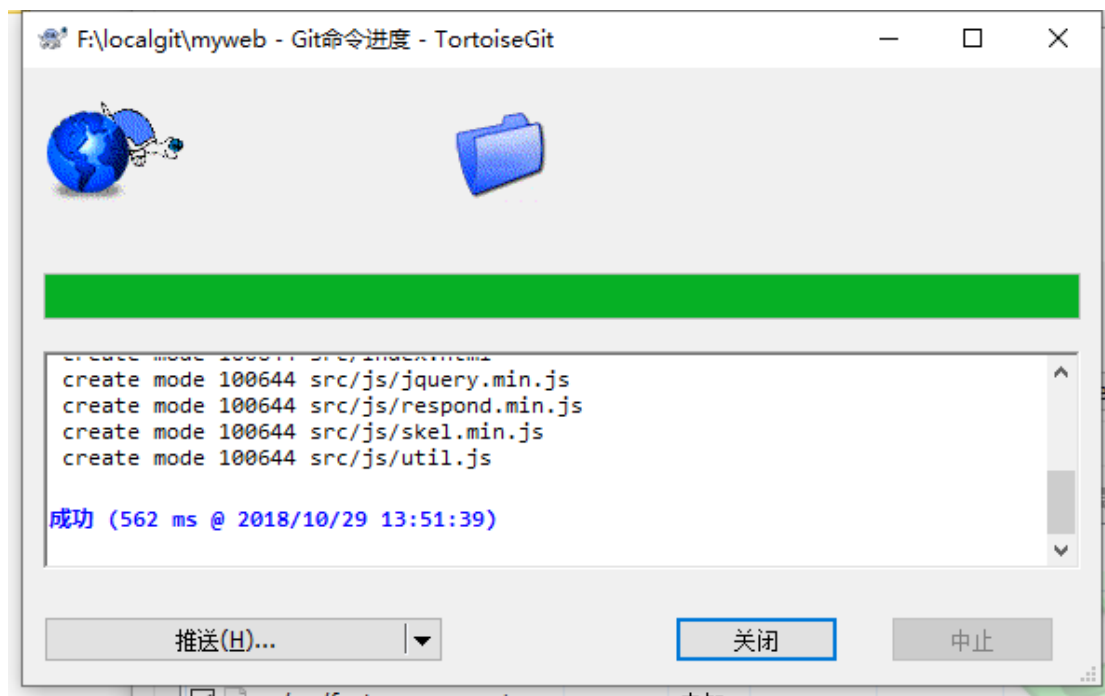




### 2.4.4.3 提交本地代码到本地的代码仓库

将本地开发的网站代码的源码文件放到本地的 git 库 myweb 目录中，然后在 myweb 目录中右键“git 提交”，之后将本地的修改提交到本地的 master 分支中。

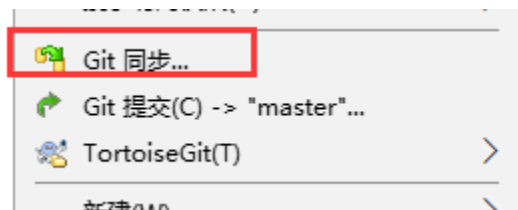


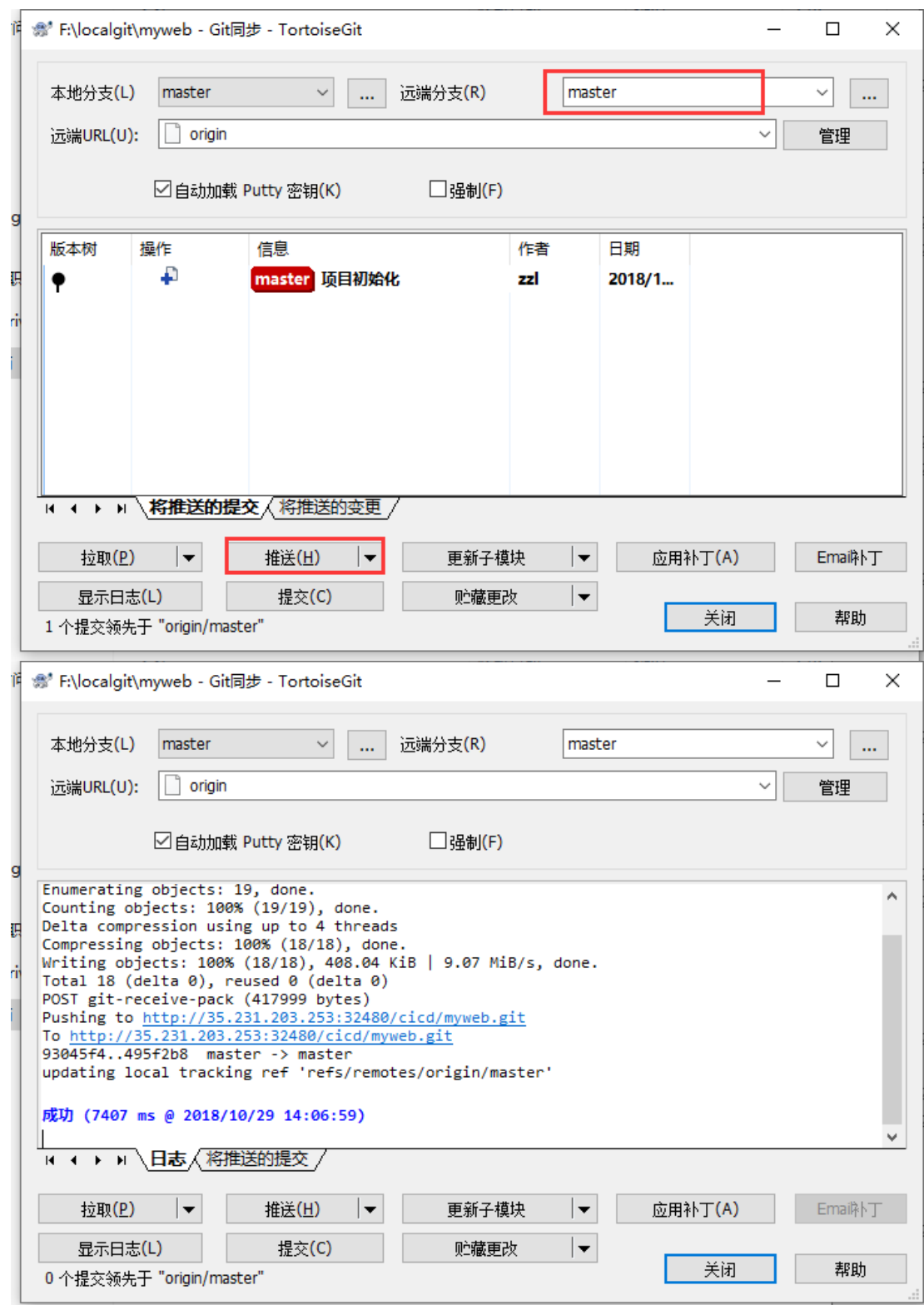


#### 2.4.4.4 同步本地代码仓库到远端代码仓库

点“git 同步”将已经提交的变更同步到远端代码仓库中。

选择远端的分支（现在远端只有 **master** 一个分支），之后点“推送”将本地的变更推送到远端。推送成功后到 **gitlab** 中可以看到本地的代码已经传上去了。







## 2.4.5 确认 gitlab 代码仓库中的内容

登录 gitlab，查看 myweb 项目中的代码内容，可以发现 src 目录已经上传了。

[cicd](#) > [myweb](#) > [Details](#)

 **myweb**  Internal [Add license](#)

Project ID: 4

0


☆ Star


0

🔗 Fork


HTTP

http://gitlab.example.c





+

 Global

[Readme](#) [Files \(92 KB\)](#) [Commit \(1\)](#) [Branch \(1\)](#) [Tags \(0\)](#)



Add Changelog


Add Contribution guide



Enable Auto DevOps


Add Kubernetes cluster



Set up CI/CD


master  myweb / 

[History](#) [Find file](#) [Web IDE](#) 

 **项目初始化**  
zhangzhilong authored 2 hours ago 

495f2b85 

Name	Last commit	Last update
 src	项目初始化	2 hours ago
 README.md	Initial commit	2 hours ago

 README.md

myweb

## 3 Jenkins 配置流水线

### 3.1 方案总览

首先是本地开发机将代码同步到 **gitlab**，然后触发 **jenkins** 将代码同步到 **k8s** 集群，并在 **nginx** 中运行起来。

### 3.2 全局配置

#### 3.2.1 配置到 **gitlab** 代码仓库的连接

“系统管理” → “系统设置” → “gitlab”。

去掉认证的勾选（否则 **gitlab** 中的 **webhook** 会报 403 错误），填写连接名称，**gitlab** 的连接地址可以填写为内网的地址 <http://10.142.0.2:32480>，证书新增一个，凭据类型选择“Gitlab API Token”，值填写 **gitlab** 中获取到的 token “**xoZbzrGBfxAHFjqZNDnH**”，ID 取名为“**gitlab-zzl-token**”，最后点测试，能成功连接即可应用/保存。

Gitlab

Enable authentication for '/project' end-point ☒

GitLab connections

Connection name

A name for the connection

Gitlab host URL

The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

Credentials

API Token for accessing Gitlab


Success

高级...

Test Connection

删除

新增

 添加凭据

Domain

全局凭据 (unrestricted)

类型

GitLab API token

范围

全局 (Jenkins, nodes, items, all child items, etc)

API token

.....

ID

gitlab-zzi-token

描述

添加

取消

### 3.2.2 配置到 k8s-master 操作系统的 ssh 连接

“系统管理” → “系统设置” → “publish over SSH” → “新增”。

填写 k8s-master 节点的操作系统信息，IP 地址填写为内网的地址，用户名为 root，勾选使用密码认证，然后填写 root 的密码。远程目录指的是 ssh 过去之后默认的目录。最后点测试，能够连通即可应用/保存。

Disable exec

SSH Servers

SSH Server

Name

k8s-master

Hostname

10.142.0.2

Username

root

Remote Directory

/nfs/jenkins-repo

☒ Use password authentication, or use a different key

Passphrase / Password

.....

Path to key

Key

Jump host

Port

22

Timeout (ms)

300000

Disable exec

Proxy type

Proxy host

Proxy port

0

Proxy user

Proxy password

Success

Test Configuration

### 3.2.3 配置 webhook

“系统管理” → “系统设置” → “gitlab web hook”。

勾选自动创建项目，项目的分支默认为 master，项目的名称使用 master 分支的项目名称。

Gitlab Web Hook

Create new projects for merge requests

Trigger build also when pushing to merged branches

☒ Automatic project creation

Project master branch

master

Use master project name

☒

Description

Automatically created by Gitlab Web Hook plugin

高级...



### 3.2.4 配置到 kubernetes 集群的连接

“系统管理” → “系统设置” → “云” → “增加一个云” → “kubernetes”。

【A】 如果 jenkins 是以 pod 的形式运行在当前的 kubernetes 集群中，则可以将 kubernetes 的地址直接配置成 <https://kubernetes.default> 即可测试成功，jenkins 的地址配置成 <http://jenkins.default>

云

Kubernetes

名称

Kubernetes 地址

Kubernetes 服务证书 key

禁用 HTTPS 证书检查

Kubernetes 命名空间

凭据

Jenkins 地址

Jenkins 通道

Connection Timeout

Read Timeout

容器数量

Pod Retention

连接 Kubernetes API 的最大连接数

kubernetes

https://kubernetes.default

☐

- 无 -

http://jenkins.default

0

0

10

Never

32

?

?

?

?

?

?

?

?

?

?

?

?

?

连接测试

【B】如果jenkins是独立运行的，或者需要添加其他的kubernetes集群，那么参考如下的方法。

从对应的kubernetes集群的master节点的~/.kube/config文件中找到三个值certificate-authority-data、client-certificate-data、client-key-data，找到他们后边对应的字符串或者文件。

```
openssl pkcs12 -export -out cert.pfx -inkey client.key -in client.crt -certfile ca.crt
```

Enter Export Password: (12345678)

Verifying - Enter Export Password:

```
[root@icp-master pkcs12]# ll
```

total 20

```
-rw-r--r-- 1 root root 1025 Nov  2 16:04 ca.crt
-rw-r--r-- 1 root root 3645 Nov  2 16:09 cert.pfx
-rw----- 1 root root 6052 Nov  2 16:08 client.crt
-rw----- 1 root root 1704 Nov  2 16:06 client.key
```

范围

全局 (Jenkins, nodes, items, all child items, etc)

Certificate

From a PKCS#12 file on Jenkins master

File

/var/jenkins\_home/pkcs12/cert.pfx

⚠ Could retrieve key "1". You may need to provide a password

Upload PKCS#12 certificate

Password

.....

ID

icp-k8s-certificate

描述

icp-k8s-certificate

保存

Kubernetes

名称

Kubernetes 地址

Kubernetes 服务证书 key

禁用 HTTPS 证书检查

Kubernetes 命名空间

凭据

icp

https://192.168.50.126:8001

☒

default

CN=kubecfg, O=system:masters (icp-k8s-certificate) Add

Connection test successful

连接测试

## 3.3 配置代码同步流水线

### 3.3.1 新建任务

“新建任务”→填写任务名称 **myweb**→选择“构建自由风格的软件项目”。之后对任务的参数进行配置。

### 3.3.2 配置 gitlab 代码仓库

“源码管理”配置 **gitlab**，URL 为 **gitlab** 中创建的项目：<http://35.231.203.253:32480/cicd/myweb.git>，证书新增一个，类型为用户名和密码认证，填写 **gitlab** 的登录用户名 **zzl** 和密码 **12345678**。由于 **gitlab** 的该项目中只有 **master** 一个分支，所以分支默认即可【如有其它分支且要从该分支中构建，如 **dev**，则填写 **\*/dev**】。

源码管理

无

Git

Repositories

Repository URL

http://35.231.203.253:32480/cicd/myweb.git

Credentials

zzl/\*\*\*\*\*

Ad\*

Name

Refspec

Add Repository

Branches to build

Branch Specifier (blank for 'any')

\*/master

Add Branch

源码库浏览器

(自动)

Additional Behaviours

新增

Mercurial

Subversion

添加凭据

Domain

全局凭据 (unrestricted)

类型

Username with password

范围

全局 (Jenkins, nodes, items, all child items, etc)

Username

zzl

Password

\*\*\*\*\*

ID

gitlab-zzl-password

描述

添加

取消

### 3.3.3 构建触发器

构建触发器中选择“当 gitlab 中有改变时进行构建”，这样当 gitlab 中的代码有任何改变时，都会触发该任务。

同时需要记录下 gitlab webhook 的 URL `http://10.142.0.2:32080/project/myweb`，后面需要到 gitlab 中去配置 webhook。

**构建触发器**

☐ 触发远程构建 (例如,使用脚本) ?

☐ 其他工程构建后触发 ?

☐ 定时构建 ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: `http://35.231.203.253:32080/project/myweb` ?

Enabled GitLab triggers	Push Events
	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input checked="" type="checkbox"/>
Accepted Merge Request Events	<input type="checkbox"/>
Closed Merge Request Events	<input type="checkbox"/>
Rebuild open Merge Requests	<div>Never ▾</div>
Approved Merge Requests (EE-only)	<input checked="" type="checkbox"/>
Comments	<input checked="" type="checkbox"/>
Comment (regex) for triggering a build	<div>Jenkins please retry a build <span>?</span></div>

高级...

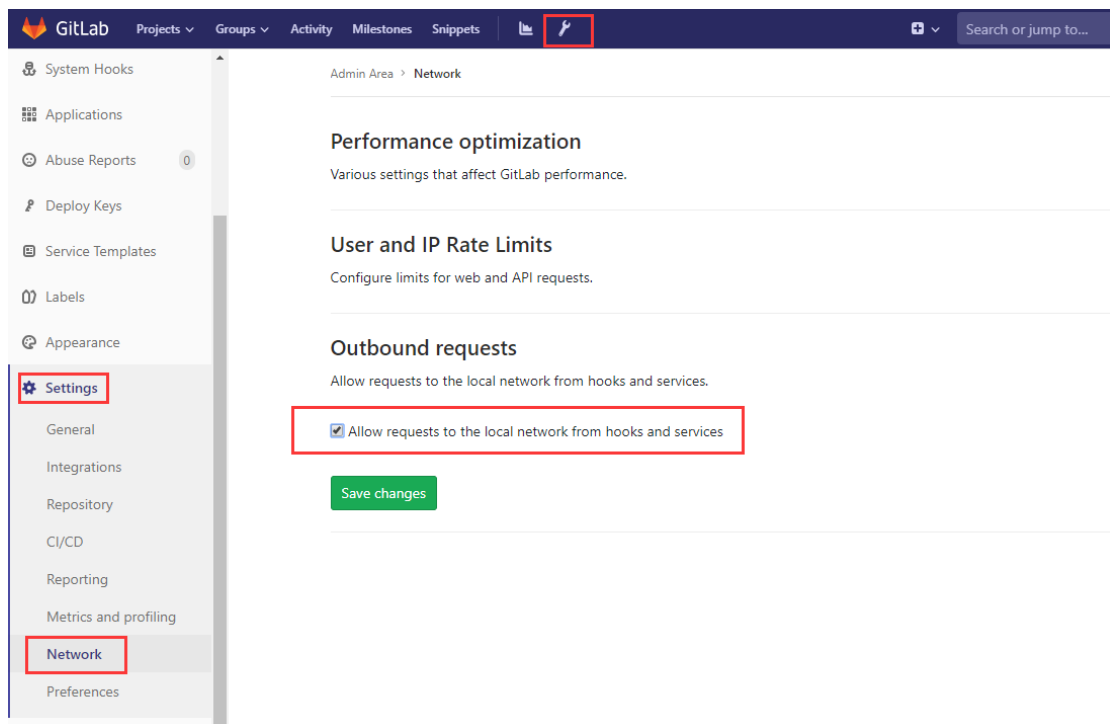
☐ GitHub hook trigger for GITScm polling ?

☐ 轮询 SCM ?

### 3.3.4 Gitlab 中配置 webhook

使用 `admin@example.com` 用户登录 gitlab，修改参数，允许针对本地网络进行 webhook。【只用配置一次即可】

“admin area” → “Settings” → “network”，勾选允许针对本地网络使用 webhook。

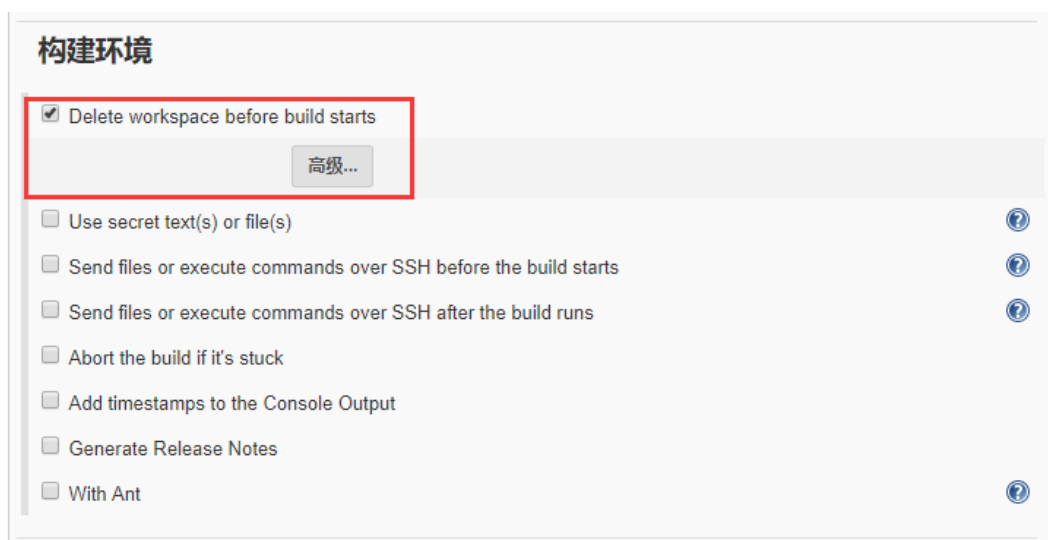


再使用 zzi 用户登录 gitlab，进入 myweb 项目中，然后“settings”→“Intergations”，填写 URL：http://10.142.0.2:32080/project/myweb，只勾选“Push Enents”，去掉“Enable SSL verification”，然后添加 webhook 选择“push events”进行测试，测试成果的返回是 http200。如果返回的是 http403，则需要在 jenkins 中把 gitlab 的认证取消掉才行（参考“[配置到 gitlab 代码仓库的连接](#)”）。

The screenshot shows the GitLab web interface for a project named 'myweb'. The 'Integrations' settings page is open, displaying various event types that can be subscribed to via webhooks. A dropdown menu is open, showing 'Push events' as the selected option. Below the settings, a green 'Add webhook' button is visible. Underneath, a list of webhooks shows one existing webhook with the URL 'http://10.142.0.2:32080/project/myweb' and the event type 'Push Events'. At the bottom of the page, a blue banner indicates 'Hook executed successfully: HTTP 200'.

### 3.3.5 配置构建环境

构建环境中，勾选“delete workspace before build starts”，要求在构建开始之前先删除本地的 `workspace` 目录内容，因为下一次构建时，可能 `gitlab` 仓库中的文件已经发生改变，尤其是删除了文件的时候，这样可以防止 `jenkins` 的 `workspace` 中还残留有 `gitlab` 中已经删除的文件。



### 3.3.6 【方式一】同步代码到 k8s 集群中

如果该代码不需要直接加入到镜像中，只是需要放到容器中挂载的共享磁盘目录上即可，那么可以将代码同步到 k8s 集群的指定目录中。

该部分总共分三步：

- A：传输代码并删除原有的 k8s 资源
- B：拷贝代码到即将建立 pv 的目录
- C：重新建立 k8s 资源

这三步中只有第一步传输了代码，其他两步都在在 k8s-master 节点上执行命令而已，按理说可以将所有需要执行的命令都放在第一步中即可，但是实验过程中发现这样做的话，在拷贝代码到将建立 pv 的目录中时没有将代码拷过去。所以实验过程将其分为三步进行。

#### 3.3.6.1 代码说明

本方式中使用到的 k8s 的 yaml 文件有两个，pv-myweb.yaml 和 k8s-myweb.yaml，文件内容如下：



---

其中 pv 使用 nfs 的方式，对应的 nfs 目录为：/nfs/jenkins-repo/nfs-myweb，该目录也是最终需要放置代码的位置。

最终网站的访问地址是：<http://35.231.203.253:30080/>

Pv-myweb.yaml 文件内容

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-myweb
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  persistentVolumeReclaimPolicy: Recycle
  accessModes:
    - ReadWriteMany
  nfs:
    path: /nfs/jenkins-repo/nfs-myweb
    server: 10.142.0.2
```

K8s-myweb.yaml 文件内容

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myweb-pvc
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 5Gi

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: myweb
```

```
name: myweb
spec:
  replicas: 1
  selector:
    matchLabels:
      run: myweb
  template:
    metadata:
      labels:
        run: myweb
    spec:
      containers:
        - image: nginx:1.7.9
          name: myweb
          volumeMounts:
            - name: html
              mountPath: /usr/share/nginx/html
      volumes:
        - name: html
          persistentVolumeClaim:
            claimName: myweb-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: myweb-svc
spec:
  selector:
    run: myweb
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30080
  type: NodePort
```

---

### 3.3.6.2 传输代码并删除原有的 k8s 资源

“构建”→“增加构建步骤”→“send files or execute commands over ssh”。

ssh server 选择 k8s-master;

**Source files:** 需要传输的文件，其实 gitlab 中的 myweb 项目的所有文件都会在 jenkins 服务器的\$WORKSPACE(/var/jenkins\_home/workspace/myweb) 目录中存一份，所以这里的文件其实下相对于 jenkins 服务器上\$WORKSPACE 目录来说的。填写 “\*\*” 说明是要传送该目录下所有的文件和文件夹（包含隐藏的文件，但不包含隐藏的文件夹）。

忽略的前缀留空，因为本次传输不需要自动删除前缀目录；

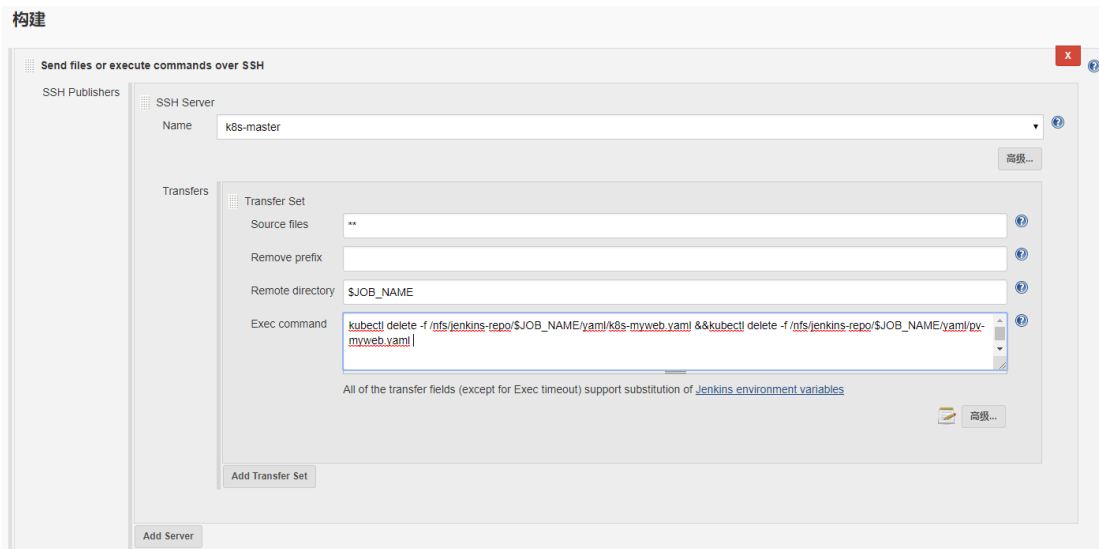
远程目录填写为“\$JOB\_NAME/”，这个目录是传输过去的文件放置的目录，也是一个相对的路径，是相对于 k8s-master 这个 ssh server 上配置的默认目录 “/nfs/Jenkins-repo” (参见 “[配置到 k8s-master 操作系统的 ssh 连接](#)”)。

执行的命令填写如下，即执行删除 pv 和 k8s 等资源的命令。执行之后将删除/nfs/jenkins-repo/nfs-myweb 目录下的所有内容。

```
kubectl delete -f /nfs/jenkins-repo/$JOB_NAME/yaml/k8s-myweb.yaml &&kubectl delete -f /nfs/jenkins-repo/$JOB_NAME/yaml/pv-myweb.yaml
```

**Exclude Files:** 需要排除的文件，本次实验中，不需要讲 git 的配置信息传输过去，即要排除.git 目录，填写 “\*\*/.git/\*\*” 会排除该目录。

**Clean remote:** 勾选的话，在同步之前将会清空远程目录的内容。



### 3.3.6.3 拷贝代码到即将建立 pv 的目录

“构建”→“增加构建步骤”→“send files or execute commands over ssh”。

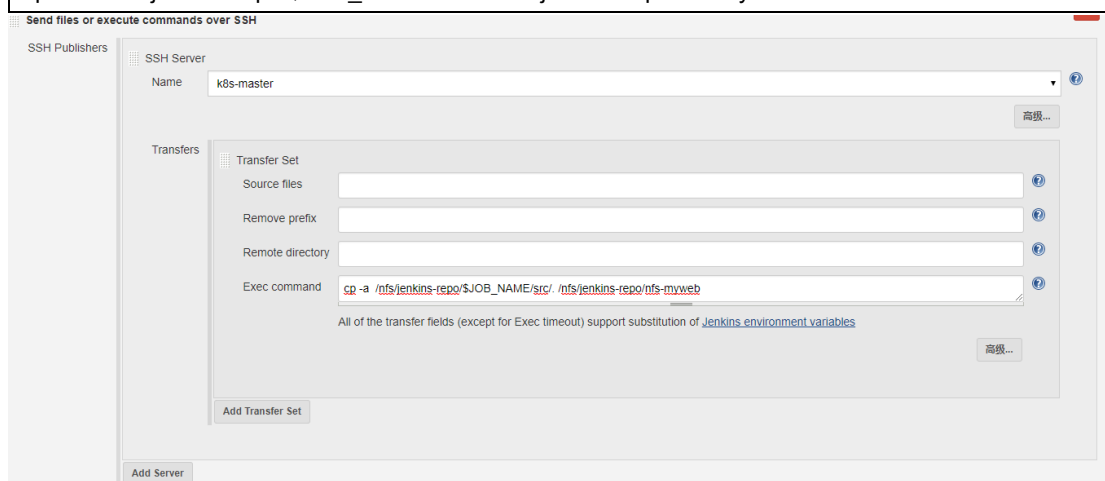
ssh server 选择 k8s-master;

Source files: 需要传输的文件, 因为文件在第一步已经传输过了, 所以留空。

RemoteDirectory: 远程目录, 留空。

执行的命令填写如下, 拷贝/nfs/jenkins-repo/\$JOB\_NAME/src/目录中的所有文件和文件夹到/nfs/jenkins-repo/nfs-myweb 目录。(nfs/jenkins-repo/nfs-myweb 目录是建立 pv 的 nfs 目录)。

```
cp -a /nfs/jenkins-repo/$JOB_NAME/src/ /nfs/jenkins-repo/nfs-myweb
```



其余默认。

### 3.3.6.4 重新建立 k8s 资源

“构建”→“增加构建步骤”→“send files or execute commands over ssh”。

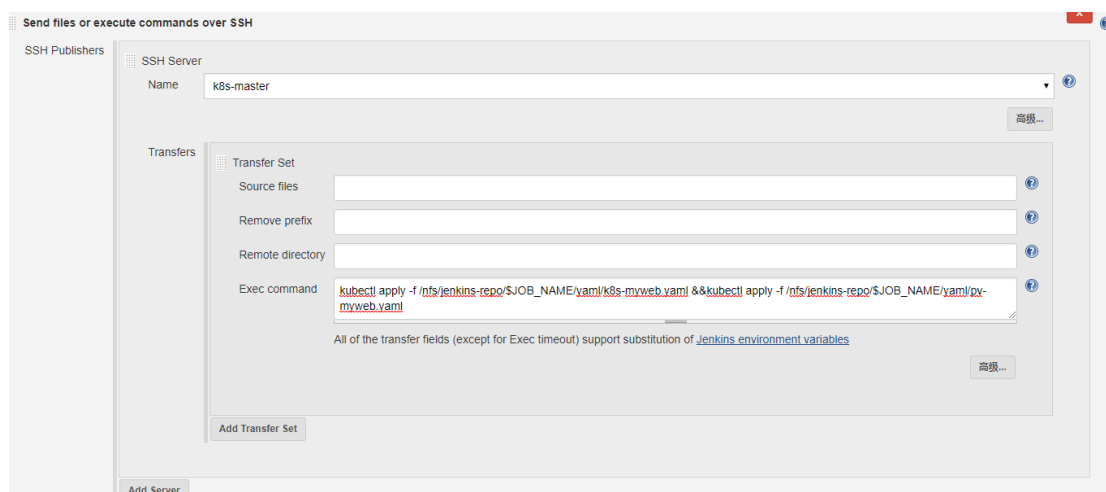
ssh server 选择 k8s-master;

Source files: 需要传输的文件, 因为文件在第一步已经传输过了, 所以留空。

RemoteDirectory: 远程目录, 留空。

执行的命令填写如下, 执行 `kubectl` 重新创建 pv 和 k8s 资源。

```
kubectl apply -f /nfs/jenkins-repo/$JOB_NAME/yaml/k8s-myweb.yaml &&kubectl apply -f /nfs/jenkins-repo/$JOB_NAME/yaml/pv-myweb.yaml
```



其余默认。

### 3.3.7 【方式二】将代码打包到镜像中

如果该代码需要直接打包进入镜像中，那么直接打包到镜像里，之后将镜像传输到私有镜像仓库中，再在 k8s 集群中启动相应的资源。

#### 3.3.7.1 代码说明

本方式中使用到文件有两个，Dockerfile 和 k8s-myweb1.yaml，文件内容如下：

其中 Dockerfile 是将程序代码打包到镜像中配置文件，用于建立镜像。

K8s-myweb1.yaml 是运行该镜像建立 k8s 资源，其中的变量 `$KUBERNETES_SECRET_NAME` 在运行时会自动被最初创建私有镜像仓库时建立的 secret，该 secret 在 jenkins 中进行配置。

最终运行起来的网站的访问地址是：<http://35.231.203.253:30081/>

Dockerfile 的内容

```
FROM nginx:1.7.9
ADD src /usr/share/nginx/html
```

K8s-myweb1.yaml 文件内容

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: myweb1
  name: myweb1
spec:
  replicas: 1
  selector:
    matchLabels:
      run: myweb1
  template:
    metadata:
      labels:
        run: myweb1
```

```
spec:
  containers:
    - image: local-registry:5000/myweb1:latest
      name: myweb1
  imagePullSecrets:
    - name: $KUBERNETES_SECRET_NAME

---
apiVersion: v1
kind: Service
metadata:
  name: myweb1-svc
spec:
  selector:
    run: myweb1
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30081
  type: NodePort
```

### 3.3.7.2 制作镜像并上传到私有镜像仓库

“构建” → “增加构建步骤” → “Docker build and publish”。

**Repository name:** 镜像的名称, 这里是以 JOB\_NAME 命名, 即本次 jenkins 任务的名称。

**TAG:** 镜像的标签, 这里是以构建的次数 BUILD\_NUMBER 来指定的, 上传时, 会上传两个镜像, 一个标签是 latest, 一个标签是 BUILD\_NUMBER。

**Docker Host URI :** 执行打镜像的 docker 进程信息, 因为本次启动 jenkins 使用的是宿主机上的 docker 进程, 因此可以填写为 `unix:///var/run/docker.sock` 。

**Docker registry URL :** 填写私有镜像仓库的连接 <http://local-registry:5000> 。

**Registry Credentials :** 登录私有镜像仓库的认证, 类型选择用户名和密码, 填写登录私有镜像仓库的用户名 `chilone`, 密码 `12345678`。

点开高级，Dockfile Path：填写 Dockerfile 的相对路径，本次实验是 dockerfile/Dockerfile (和本地的代码中的位置一致)。

注意：必须在 jenkins 上成功登陆一次镜像仓库，即保证 jenkins 容器中的 ~/.docker/config.json 文件中有该镜像仓库的登录认证信息。该步骤在执行时才会正确，否则在上传镜像时会报 “unauthorized: authentication required” 的错误。

这里配置的 registry credentials 似乎没有生效。

**构建**

Docker Build and Publish

X

?

Repository Name

\$JOB\_NAME

?

Tag

\$BUILD\_NUMBER

Docker Host URI

unix:///var/run/docker.sock

?

Server credentials

- 无 -

Ad

Docker registry URL

http://local-registry:5000

?

Registry credentials

chilone/\*\*\*\*\*

Ad

https\_proxy= http://some.proxy:port

Dockerfile Path

dockerfile/Dockerfile

Path to the Dockerfile for this build. File must be relative to build context. Can be used to specify a Dockerfile with a non-standard filename. Uses Docker client default if not specified.

### 3.3.7.3 建立或更新 k8s 资源


(1) 新增加一个连接 k8s-master 的凭据，“主页” → “凭据” → “系统” → “全局凭据” → “添加凭据”。


填写可以登录到 k8s-master 节点的操作系统用户名和密码。



---

Jenkins > 凭据 > 系统 > 全局凭据 (unrestricted) >

 返回到凭据域列表

 添加凭据

类型

Username with password

范围

全局 (Jenkins, nodes, items, all child items, etc)

Username

root

Password

.....

ID

k8s-master

描述

确定

## (2) 添加部署到 k8s 集群的构建步骤

“构建” → “增加构建步骤” → “deploy to kubernetes”。

Kubeconfig：新增凭据，类型选择 kubeconfig，ID 填写 k8s-master-kubeconfig，kubeconfig 选择 “from a file on the kubernetes master node”，server 填写 k8s-master 的 IP 地址，ssh Credentials 选择上面创建的登录 k8s-master 节点的凭据。

注意：kubeconfig 对应的文件 config 中，不能出现引用其他的文件，且 current-context 后必须有正确的值，否则会因为找不到对应的文件而连接不到 k8s 集群。如果 config 中出现了引用其他的文件，最好将 config 和对应的文件一并放到 jenkins 服务器的一个目录中，然后从该目录获取 kuconfig。

 **Jenkins 凭据提供者: Jenkins**

 添加凭据

Domain

全局凭据 (unrestricted)

类型

Kubernetes configuration (kubeconfig)

范围

全局 (Jenkins, nodes, items, all child items, etc)

ID

k8s-master-kubeconfig

描述

Kubeconfig

☐ Enter directly

☐ From a file on the Jenkins master

☒ From a file on the Kubernetes master node

Server

10.142.0.2

SSH Credentials

root/\*\*\*\*\*

Add

File

.kube/config

添加

取消

**Config files:** 填写 k8s 资源的 yaml 文件的名称 “yaml/k8s-web1.yaml” 即可。

点击 **Docker Container registry Credentials/kubernetes** 打开更多配置。

**Namespace** 默认是 **default**，根据实际需要更改，表示下面要创建的这个 **secret** 是在什么命名空间中的，。

**Secret** : 填写允许 k8s 集群从私有仓库下载镜像的 **secret** 的名称，如果集群中在对应的 **namespace** 中没有这个 **secret** 的话，则会自动创建。如果不指定这个 **secret**，将会出现镜像下载失败的错误。这里配置的 **secret** 值将自动替换 k8s-myweb1.yaml 文件中的 **\$KUBERNETES\_SECRET\_NAME** 变量。

**Docker Container registry Credentials** : 填写私有镜像仓库的 **URL**

<http://www.yuandingit.com>

第 47 页 / 共 49 页

7x24 小时服务热线: 400-007-0628

<http://local-registry:5000> ; 认证选择之前创建的登录私有仓库的凭据。

之后点 **verify configuration** 测试一下配置。

Deploy to Kubernetes

Kubeconfig: k8s-master-kubeconfig (k8s-master-kubeconf) Add

Config Files: yaml/k8s-myweb1.yaml

Enable Variable Substitution in Config: ☒

Deprecated Kubeconfig Settings...

Docker Container Registry Credentials / Kubernetes Secrets

Kubernetes Namespace for Secret: default

Secret Name: regcred

Docker Container Registry Credentials

Docker registry URL: http://local-registry:5000

Registry credentials: chilone/\*\*\*\*\* (local-...) Add

新增

Successfully validated configuration

Verify Configuration

### 3.4 触发自动构建任务

当本地的代码修改提交到 **gitlab** 时, **jenkins** 将会自动触发构建任务, 将代码同步到 **k8s** 集群, 或者讲代码打包成新的镜像, 并在 **k8s** 集群中启动起来。

通过两种方式的链接均可进行访问:

方式一: <http://35.231.203.253:30080/>

方式二: <http://35.231.203.253:30081/>

---

## 4 其他说明

方式一 过程中先删除了 **k8s** 资源，再拷贝代码，再创建 **k8s** 资源，中途会造成服务中断。

方式二 直接将代码打包到镜像中，然后执行 **kubectl apply** 时，只更新了镜像，其他配置没有变化，中途服务中断的时间相对较短。