

---

<https://github.com/johnhuang-cn>

---

微服务持续集成  
**Spring Cloud & GitLab & Docker & K8S**

**Version <1.0>**

# **Revision History**

<b>Date</b>	<b>Version</b>	<b>Description</b>	<b>Author</b>
7/15/2018	1.0	Initial version	John.Huang

# Catalog

1. 前言	5
1.1 目标	5
1.2 内容	5
1.3 预备环境	5
1.4 预备知识	5
1.5 预备文件	6
1.6 文档约定	6
2. 基本步骤及原理	6
3. 架构图	7
4. 网络设置	8
4.1.1 关闭防火墙	8
4.1.2 设置 ipv4_forward=1	8
5. 安装 Docker	8
5.1.1 官方文档	8
5.1.2 推荐步骤	9
6. 安装本地 Docker 库	10
7. GitLab 环境安装	11
7.1 安装 GitLab	11
7.1.1 使用 docker 方式安装 GitLab	11
7.1.2 创建 GitLab 存储目录	11
7.1.3 运行 GitLab	12
7.1.4 设置 GitLab 的 Root 密码	12
7.1.5 其它命令	14
7.2 安装 GitLab Runner	14
7.2.1 添加 GitLab 官方库	14
7.2.2 安装 Runner	15
7.2.3 获取 Runner Token	15

7.2.4 注册 Runner	15
7.2.5 检查结果	17
8. 安装 Minikube 环境	18
8.1 国内安装 Minikube 的三种方法	18
8.2 安装 Docker	18
8.3 安装 kubelet kubeadm kubectl	18
8.4 安装 minikube	19
8.5 启动	19
8.6 打开控制台	19
9. 镜像准备	20
9.1 创建 Aliyun Maven Docker 镜像	20
9.2 创建 kubectl 镜像	20
9.3 创建 Oracle JAVA 镜像	21
10. 创建 GitLab 项目	21
10.1 创建一个 k8s-demo Group	21
10.2 在 GitLab 创建项目	22
10.3 配置 Group 环境变量	23
10.4 提交项目到 GitLab	24
10.5 触发 CI	25
11. CI 过程相关配置文件说明	27
11.1 Trigger	27
11.2 .gitlab-ci.yml	28
11.3 Dockerfile 说明	29
11.4 Deployment 文件说明	29
11.5 Service 文件说明	31
11.6 Spring Cloud Eureka Client 端配置	31

# 1. 前言

## 1.1 目标

搭建 Spring Cloud + GitLab + Docker + K8S 的持续集成开发环境。

## 1.2 内容

- **Spring Cloud Hello World 工程**  
    基于 SpringCloud 2.0.3，含 Eureka, Feign/Hystrix, Gateway
- **GitLab 及持续集成配置**
- **Docker 及本地 Docker 仓库**
- **Kubernetes Minikube 搭建及部署**

## 1.3 预备环境

CentOS7 Linux 主机两台：

主机 A: Docker 本地仓库, GitLab 及 GitLab Runner 环境, 本文档中 IP 为 192.168.1.211

主机 B: Kubernetes Minikube 环境, 本文档中 IP 为 192.168.1.201

注：

1. 本文档基本上也适用于 Ubuntu 16.0.4
2. 请确认 SSH 服务已安装且开启，可用命令 `ps -e | grep ssh` 检查

```
[john@GitLab ~]# ps -e |grep ssh
 1066 ?        00:00:00 sshd
```

3. CentOS7 一般默认是安装了 SSH 的，Ubuntu 的可能没有，可以用下面命令安装

```
sudo apt-get update
```

```
sudo apt-get install openssh-server
```

## 1.4 预备知识

1. 掌握基本 Spring Boot 开发
2. 掌握 Linux, Git, Maven 基础操作
3. 了解 Kubernetes 基础知识
4. 了解 Docker 基本概念和命令  
<https://docs.docker.com/get-started/>
5. 了解持续集成基础知识
6. 了解 Spring Cloud 各基本组件：Eureka Server/Client, Feign, Gateway  
中文参考文档：<https://springcloud.cc/>

英文参考文档: [http://cloud.spring.io/spring-cloud-static/Finchley.RELEASE/multi/multi\\_spring-cloud.html](http://cloud.spring.io/spring-cloud-static/Finchley.RELEASE/multi/multi_spring-cloud.html)

## 1.5 预备文件

需 clone <https://github.com/johnhuang-cn/spring-cloud-k8s-ci-template> 项目到本地, 里面的文件和项目后面需要用。

## 1.6 文档约定

文档中标注黄色的地方表示需要根据自己实际情况更改。标注红色的地方表示不要漏了。

# 2. 基本步骤及原理

GitLab 内置有 CI 功能, 可代替 Jenkins 完成大部分的编译打包发布场景。常见的 CI 过程直接由 GitLab 完成要方便些。要让 GitLab 启动 CI, 需要在工程根目录添加`.GitLab-ci.yml`文件, 在该文件里编写 CI 脚本。

通常一个 spring boot 工程部署到 K8S 的过程是这样的:

1. 用 Maven 打包成 jar
2. 生成 docker 镜像
3. 将生成的镜像 push 到 docker 镜像库
4. 用 kubectl 将镜像发布到 K8S 集群

为什么使用 Docker:

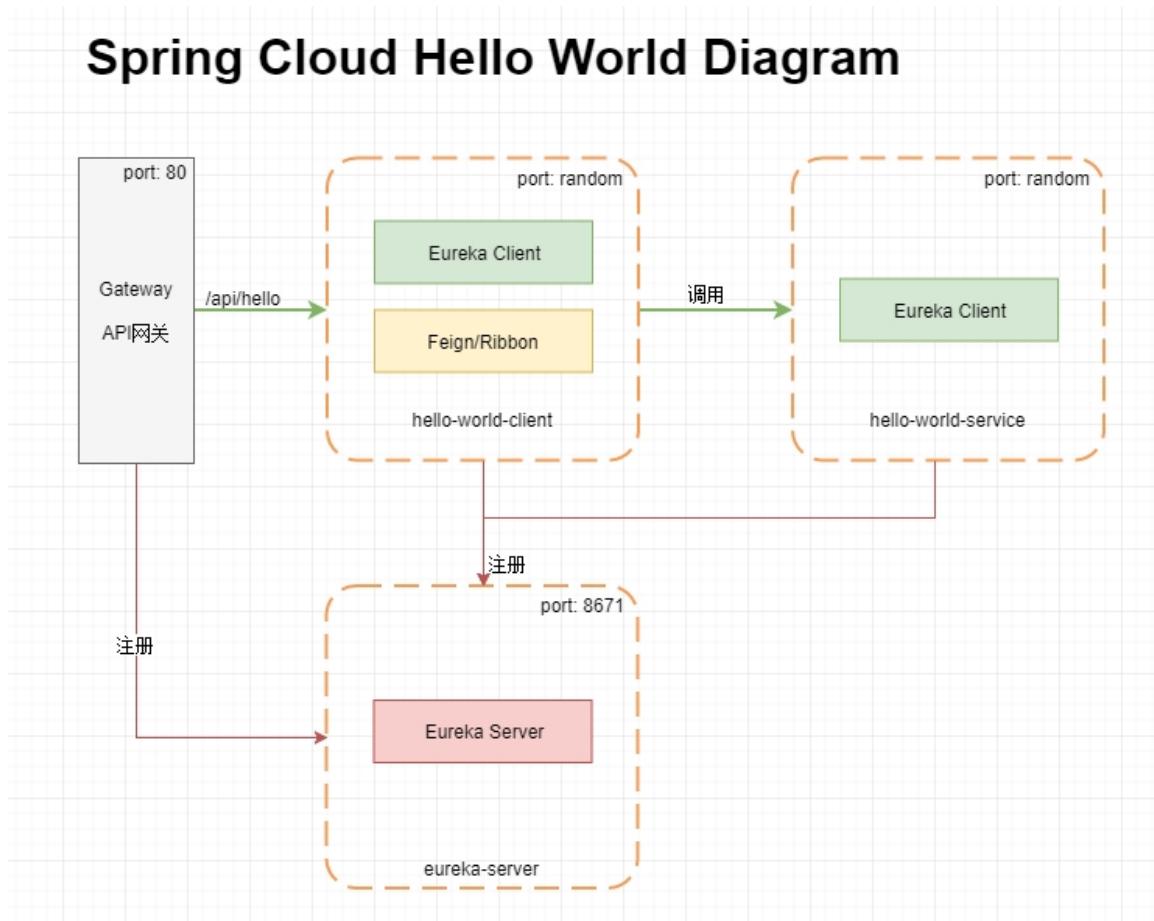
1. 打包编译和运行过程使用 Docker 不会污染主机环境, 由于 Docker 的无状态特性, 每次启动都是全新干净的环境, 确保每次打包编译和运行都是全新的
2. 将应用程序打成 Docker, 可以实现一次编译, 到处运行
3. 用 Docker 简化了部署, 直接操作容器服务就可以了, 基本可以忽略 Linux 操作系统间的差别, 不需要每套服务器分别安装配置, 不需要记复杂的 Linux 各项配置。

用了 Spring Cloud 为什么还要使用 K8S:

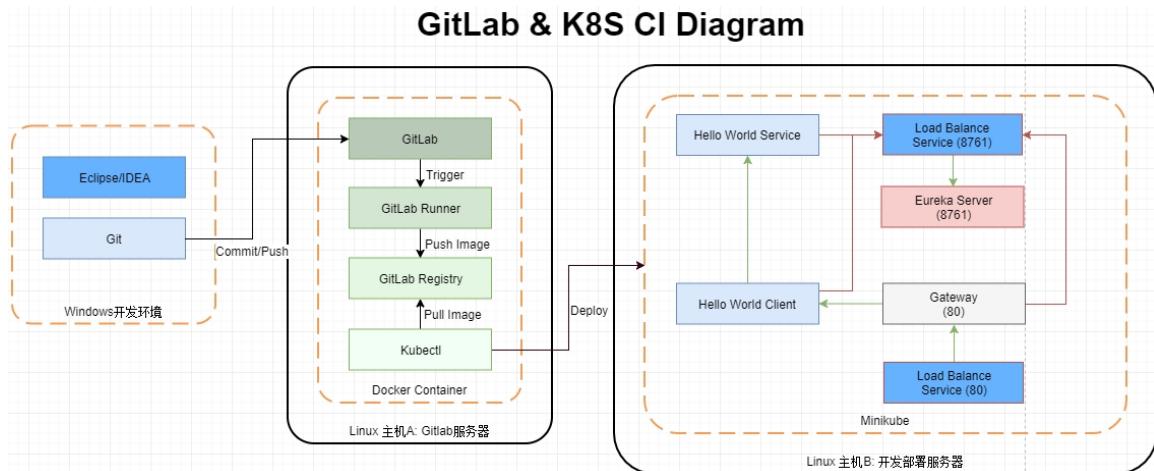
1. K8S 提供的滚动更新对运维来说十分方便, 可以实现平滑升级, 也可以在升级失败后, 快速回滚到上一版本
2. K8S 可以方便地进行弹性伸缩
3. 自动修复, 自动重启

下面开始设置完成这 4 个步骤所需的环境和脚本。

### 3. 架构图



组成：Gateway api 网关 > 消费端 > 服务端 和一个注册中心，共 4 个 Spring Boot 项目



## 4. 网络设置

以下设置需要在主机 A/B 上设置

### 4.1.1 关闭防火墙

在默认环境下，Docker 启动的容器互访会有网络访问问题，初次配置最好先将防火墙关掉。

CentOS7 版本防火墙默认使用 firewalld，因此使用以下命令关闭：

```
systemctl stop firewalld
```

```
systemctl disable firewalld
```

Ubuntu16.04，可以用

```
ufw disable
```

### 4.1.2 设置 *ipv4\_forward=1*

不设置的话 docker 容器内部访问网络会有问题。

运行：

```
sudo vi /etc/sysctl.conf
```

添加：

```
net.ipv4.ip_forward=1
```

重启 network：

```
systemctl restart network
```

检查：

```
sysctl net.ipv4.ip_forward
```

## 5. 安装 Docker

需要在主机 A/B 上均安装 Docker

### 5.1.1 官方文档

CentOS <https://docs.docker.com/install/linux/docker-ce/centos/>

Ubuntu <https://docs.docker.com/install/linux/docker-ce/ubuntu/>

### 5.1.2 推荐步骤

通过 Repository 方式安装，设置步骤稍多。个人喜欢 Install from a package 方式。

1. 通过 [https://download.docker.com/linux/centos/7/x86\\_64/stable/Packages/](https://download.docker.com/linux/centos/7/x86_64/stable/Packages/) 查看 CentOS7 下的 Docker-CE 最新稳定版本

## Index of /linux/centos/7/x86\_64/stable/Packages/

..		
<a href="#">docker-ce-17.03.0.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	19M
<a href="#">docker-ce-17.03.1.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	19M
<a href="#">docker-ce-17.03.2.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	19M
<a href="#">docker-ce-17.06.0.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	21M
<a href="#">docker-ce-17.06.1.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	21M
<a href="#">docker-ce-17.06.2.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	21M
<a href="#">docker-ce-17.09.0.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	22M
<a href="#">docker-ce-17.09.1.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	22M
<a href="#">docker-ce-17.12.0.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	31M
<a href="#">docker-ce-17.12.1.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	31M
<a href="#">docker-ce-18.03.0.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	35M
<a href="#">docker-ce-18.03.1.ce-1.el7.centos.x86_64.rpm</a>	2018-06-08 05:48	35M
<a href="#">docker-ce-selinux-17.03.0.ce-1.el7.centos.noarch.rpm</a>	2018-06-08 05:48	29K
<a href="#">docker-ce-selinux-17.03.1.ce-1.el7.centos.noarch.rpm</a>	2018-06-08 05:48	29K
<a href="#">docker-ce-selinux-17.03.2.ce-1.el7.centos.noarch.rpm</a>	2018-06-08 05:48	29K

2. 使用 curl 下载 rpm

```
curl -O https://download.docker.com/linux/centos/7/x86_64/stable/Packages/docker-ce-18.03.1.ce-1.el7.centos.x86_64.rpm
```

3. 安装 Docker-CE

```
sudo yum install docker-ce-18.03.1.ce-1.el7.centos.x86_64.rpm
```

对问题一路 Yes

4. 启动 Docker

```
systemctl start docker
```

5. 测试是否安装成功

```
sudo docker run hello-world
```

```
[john@GitLab ~]$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157elc3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/
```

看到以上信息即表示安装成功

## 6. 设置开机启动

```
systemctl enable docker
```

# 6. 安装本地 Docker 库

官方文档: <https://docs.docker.com/registry/>

在主机 A 上运行安装命令:

```
sudo docker run -d \
-p 192.168.1.211:5000:5000 \
--restart=always \
--name registry \
-v /mnt/registry:/var/lib/registry \
registry:latest
```

官方的 docker search 192.168.1.211:5000 的命令查看镜像不靠谱，下面是用于查看私有库中现有镜像的命令：

```
# curl -XGET http://192.168.1.211:5000/v2/_catalog  
# curl -XGET http://192.168.1.211:5000/v2/镜像名/tags/list
```

因为 docker client 默认是要求 https 的，所以在**主机 A/B** 上需要分别设置 daemon.json 才能正常访问。在两台主机上打开/etc/docker/daemon.json (如果没有则新建)，添加以下内容：

```
{  
  "insecure-registries" : ["192.168.1.211:5000"]  
}
```

添加后需重启 docker：

```
systemctl restart docker
```

具体参考：

<https://docs.docker.com/registry/insecure/#deploy-a-plain-http-registry>

## 7. GitLab 环境安装

以下部署在**主机 A** 进行。

### 7.1 安装 GitLab

#### 7.1.1 使用 docker 方式安装 GitLab

官方参考文档：<https://docs.GitLab.com/ee/install/docker.html>

用 docker 将 GitLab CE 版镜像拉到本地，在国内下载可能需要一点时间：

```
sudo docker pull gitlab/gitlab-ce
```

#### 7.1.2 创建 GitLab 存储目录

在用户主目录下创建 GitLab 存储目录：

```
mkdir -vp gitlab/{data,logs,config}
```

```
[john@GitLab ~]$ mkdir -vp gitlab/{data,logs,config}
mkdir: created directory 'gitlab'
mkdir: created directory 'gitlab/data'
mkdir: created directory 'gitlab/logs'
mkdir: created directory 'gitlab/config'
```

### 7.1.3 运行 *GitLab*

官方参考文档: <https://docs.GitLab.com/omnibus/docker>

根据实际用户路径替换下面的 /home/john/。

根据实际 IP 替换下面的 192.168.1.211

因为 22 端口被主机 SSH 所用，所以用 1022 映射容器的 22 端口。

```
sudo docker run --detach \
--hostname 192.168.1.211 \
--publish 192.168.1.211:443:443 --publish 192.168.1.211:80:80 --publish 192.168.1.211:1022:22 \
--name gitlab \
--restart always \
--volume /home/john/gitlab/config:/etc/gitlab \
--volume /home/john/gitlab/logs:/var/log/gitlab \
--volume /home/john/gitlab/data:/var/opt/gitlab \
gitlab/gitlab-ce:latest
```

注：用上述命令(--restart always)启动后，GitLab 将随 Linux 开机启动。

### 7.1.4 设置 *GitLab* 的 Root 密码

GitLab 默认管理员用户是 root。启动 GitLab 大约一两分钟后，打开浏览器，访问 http://192.168.1.211。第一次访问会提示你修改密码，输入一个 8 位数以上的密码 (password)，提交。

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](#)

然后就可以用 root/password 登录 GitLab。

Your password has been changed successfully.

## 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

root

Password

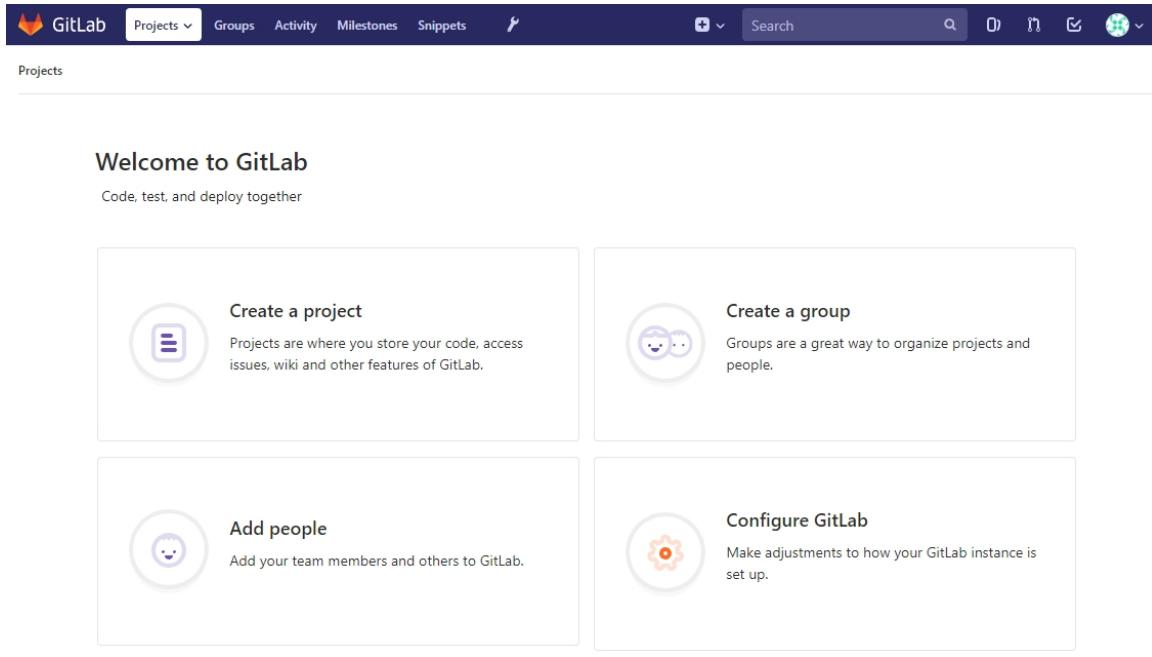
\*\*\*\*\*

Remember me

[Forgot your password?](#)

[Sign in](#)

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



### 7.1.5 其它命令

1. Stop the running GitLab container:

```
sudo docker stop GitLab
```

2. Remove existing GitLab container:

```
sudo docker rm GitLab
```

3. Pull the new GitLab image:

```
sudo docker pull GitLab/GitLab-ce:latest
```

- 4.

## 7.2 安装 GitLab Runner

Runner 可以 Shell 方式也可以以 Docker 容器方式运行。Shell 比较简单，问题会比较少。

用 Docker Container 方式一台机器可以装多个 Runner。本文中 GitLab Runner 和 GitLab 安装在同一台机器，装容器方式装多个会有性能问题，所以用简单的 Shell 方式。

官方参考文档：

<https://docs.GitLab.com/runner/install/linux-repository.html>

<https://docs.gitlab.com/runner/>

### 7.2.1 添加 GitLab 官方库

```
# For RHEL/CentOS/Fedora
```

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.rpm.sh | sudo bash
```

```
# For Debian/Ubuntu/Mint
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash
```

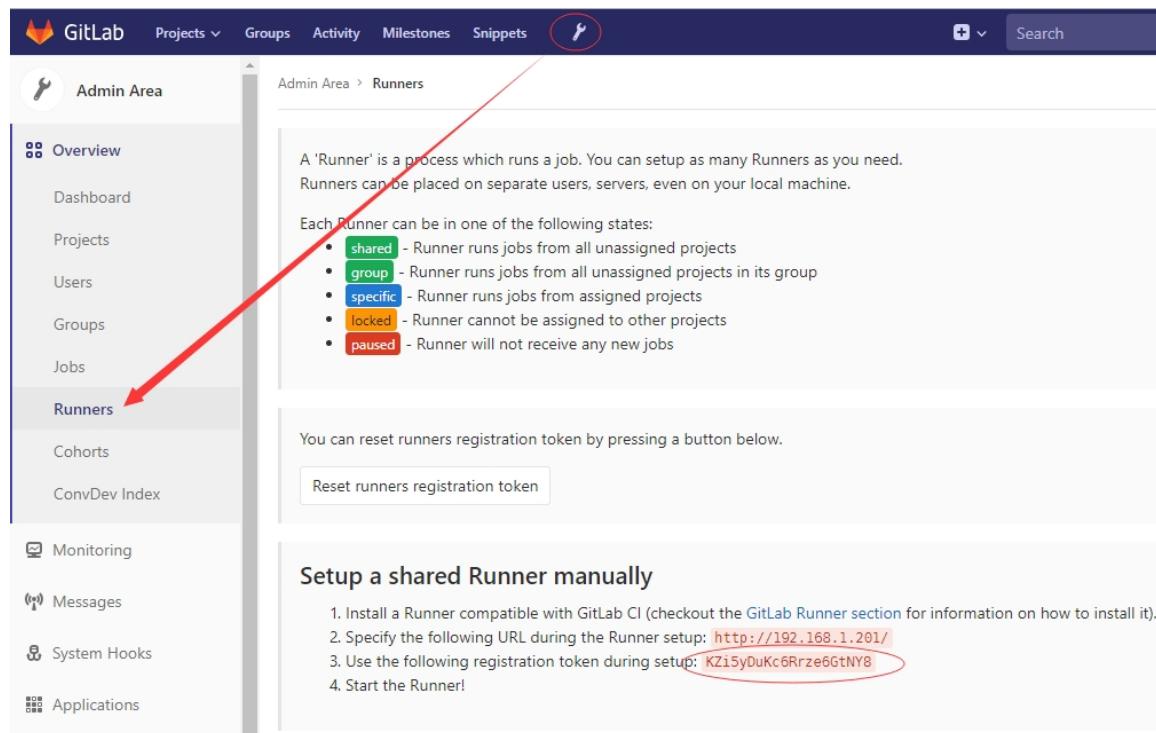
## 7.2.2 安装 Runner

```
# For RHEL/CentOS/Fedora
sudo yum install gitlab-runner

# For Debian/Ubuntu/Mint
sudo apt-get install gitlab-runner
```

## 7.2.3 获取 Runner Token

打开 GitLab 网页，按下图指示，找到注册 token。



## 7.2.4 注册 Runner

官方文档参考：<https://docs.GitLab.com/runner/register/index.html>

可以用问答形式，也可以用单行命令的形式。这里使用单行命令完成，需根据实际修改黄

色部分的值。

各行参数说明：

--non-interactive 免交互  
--executor runner 类型，这里只能是 docker  
--docker-image 默认 docker 镜像，这里用前面建的 ali-maven-docker:3.5.4-jdk-8-alpine 镜像  
--url GitLab 的地址  
--registration 上面找到的 GitLab token  
--description runner 名称(注册后可以在 GitLab ui 修改)  
--tag-list 定义标签列表，后面 GitLab Job 会用到(注册后可以在 GitLab ui 修改)  
--run-untagged 是否允许未指定 tag 的 job 用这个 runner，默认 true  
--locked 是否锁定，也就是一次只能一个 job 运行

```
sudo gitlab-runner register \
--non-interactive \
--executor "docker" \
--docker-image "192.168.1.211:5000/ali-maven-docker:3.5.4-jdk-8-alpine" \
--url "http://192.168.1.211/" \
--registration-token "KZi5yDuKc6Rrze6GtNY8" \
--description "Docker runner" \
--tag-list "shared-runner" \
--run-untagged \
--locked="false" \
--docker-privileged="false"
```

注册成功显示：

```
Registering runner... succeeded           runner=KZi5yDuK
Runner registered successfully. Feel free to start it, but if it's running already the config should
be automatically reloaded!
```

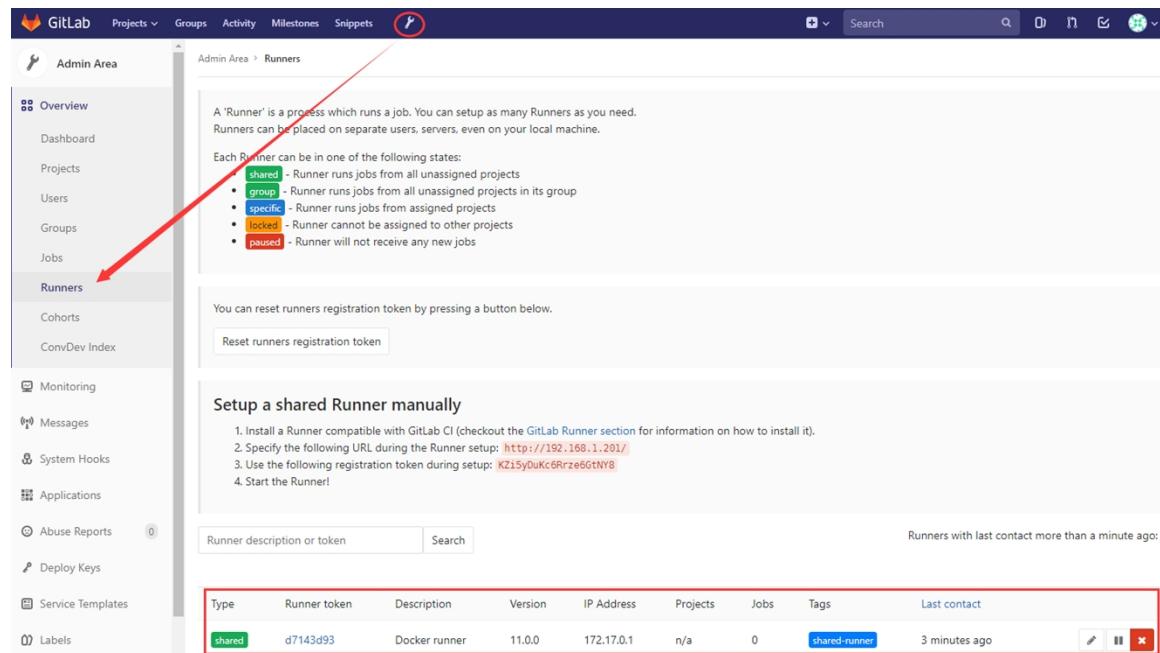
本文使用 socket binding 方式运行 docker，需修改/etc/gitlab-runner/config.toml 为 socket

binding 方式，并添加 maven 库目录的本地映射，这样每次打包时不需要重下依赖包。另外末尾加上 pull\_policy = "if-not-present"，这样不会每次都拉镜像。

```
[[runners]]  
name = "Docker runner"  
url = "http://192.168.1.211/"  
token = "d7143d934a8316d4bca29ae84851e9"  
executor = "docker"  
  
[runners.docker]  
tls_verify = false  
image = "fancybing/ali-maven-docker:3.5.4-jdk-8-alpine"  
privileged = false  
disable_cache = false  
volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache", "/home/john/.m2:/root/.m2"]  
shm_size = 0  
pull_policy = "if-not-present"  
  
[runners.cache]
```

## 7.2.5 检查结果

打开 GitLab > Admin > Runners，如下图看到 Runner 列表即为成功。



Type	Runner token	Description	Version	IP Address	Projects	Jobs	Tags	Last contact
shared	d7143d93	Docker runner	11.0.0	172.17.0.1	n/a	0	shared-runner	3 minutes ago

## 8. 安装 Minikube 环境

以下操作在机器 B (192.168.1.201) 上进行

### 8.1 国内安装 Minikube 的三种方法

因为 Kubernetes 安装过程中所需的镜像在 gcr.io 上，国内无法访问，所以无法正常安装，解决方案有：

1. 科学上网
2. 提前将所需的镜像拉下来，比较费时间
3. 用阿里云镜像地址重编译的 Minikube，本文使用这种

### 8.2 安装 Docker

Minikube 一般装在虚拟机上，比如 VirtualBox，本文所用的服务器已经是虚拟机，如果再装在虚拟机上的话，性能会很差。所以接下来的 Minikube 安装是使用--vm-driver=none 的形式。这种方式需要安装 docker，所以在上一节中要求在 B 主机也安装 Docker。这种方式也有缺点，因为它是使用 privilege=true 方式运行 api-server，直接将主机暴露在容器中，有不安全因素。

### 8.3 安装 kubelet kubeadm kubectl

参考：<https://blog.csdn.net/nklinsirui/article/details/80581286>

通过阿里云镜像安装 kubelet kubeadm kubectl：

CentOS

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

```
EOF
```

```
yum install -y kubelet kubeadm kubectl  
systemctl enable kubelet  
systemctl start kubelet
```

## Ubuntu

```
apt-get update && apt-get install -y apt-transport-https  
curl https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | apt-key add -  
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list  
deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-xenial main  
EOF  
apt-get update  
apt-get install -y kubelet kubeadm kubectl
```

## 8.4 安装 minikube

参考资料：

<https://yq.aliyun.com/articles/221687?spm=5176.10695662.1996646101.searchclickresult.3d253b3eMzCjL5>

```
curl -Lo minikube http://kubernetes.oss-cn-hangzhou.aliyuncs.com/minikube/releases/v0.28.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

## 8.5 启动

```
minikube start --registry-mirror=https://registry.docker-cn.com --vm-driver=none
```

## 8.6 打开控制台

```
minikube dashboard  
minikube dashboard --url  
http://192.168.1.203:30000  
--url 获取地址，用浏览器打开访问
```

## 9. 镜像准备

以下操作在主机 A 上进行

### 9.1 创建 Aliyun Maven Docker 镜像

需要创建一个 Maven 镜像用于编译 Spring Boot 工程并打包成 Docker 镜像上传到本地 Docker 库。官方的 Maven 是基于 Apache 中央库的，在国内用阿里要快点，另外官方镜像不支持 Docker，所以也需要集成 Docker。最后还需要修改该镜像的 /etc/docker/daemon.json，这样才能访问私库，所以需要手工创建。

先将从 <https://github.com/johnhuang-cn/spring-cloud-ci-template> 下载下来的/docke-images/ali-maven-docker/daemon.json 打开，修改里面的地址为 docker 本地库的实际地址：

```
{  
  "insecure-registries" : ["192.168.1.211:5000"]  
}
```

然后将 ali-maven-docker 整个目录拷贝到主机 A 的用户目录下，然后进入该目录，使用下面命令创建镜像并提交到私库：

```
cd ali-maven-docker  
  
sudo docker build -t 192.168.1.211:5000/ali-maven-docker:3.5.4-jdk-8-alpine .  
  
sudo docker push 192.168.1.211:5000/ali-maven-docker:3.5.4-jdk-8-alpine
```

### 9.2 创建 kubectl 镜像

需要一个 kubectl 镜像在 Gitlab 服务器上运行，并能远程连接 K8S 服务器的镜像。在 Docker Hub 上有最新的 kubectl 镜像：lachlanevenson/k8s-kubectl:v1.11.0，但除此外，为了能连接 K8S 服务器，还需要将 K8S 服务器上/etc/kubernetes/admin.conf 拷到镜像的 /root/.kube/config，所以还需要创建一个专用的镜像。

Dockerfile：

```
FROM lachlanevenson/k8s-kubectl:v1.11.0
LABEL maintainer="John Huang <john.h.cn@gmail.com>"
ENV KUBE_LATEST_VERSION="v1.11.0"
ADD admin.conf /root/.kube/config
WORKDIR /root
```

然后将 K8S(主机 B)上的`/etc/kubernetes/admin.conf`拷到 Dockerfile 所在目录，然后进入该目录，使用下面命令创建镜像并提交到私库：

```
cd kubectl
sudo docker build -t 192.168.1.211:5000/kubectl:1.11.0 .
sudo docker push 192.168.1.211:5000/kubectl:1.11.0
```

### 9.3 创建 Oracle JAVA 镜像

运行 spring-boot 应用，需要一个 java 容器。公共 hub 上没有找到 oracle 的官方 java 镜像，大都是基于 openjdk 的，貌似是因为版权问题。我这里用的是 Oracle 官方 Git 的 Dockerfile (<https://github.com/oracle/docker-images/tree/master/OracleJava/java-8>) 创建的 ServerJRE 镜像(<https://hub.docker.com/r/fancybing/java/>)，可以通过下面命令获取：

```
sudo docker pull fancybing/java:serverjre-8
```

## 10. 创建 GitLab 项目

### 10.1 创建一个 k8s-demo Group

GitLab 首页 > Create a group > 按下图创建一个 k8s-demo Group：

## New Group

Group path	<input type="text" value="http://192.168.1.211/"/> k8s-demo
Group name	<input type="text" value="k8s-demo"/>
Description	<input type="text"/>
Group avatar	<input type="button" value="Choose File ..."/> No file chosen The maximum file size allowed is 200KB.
Visibility Level 	<input checked="" type="radio"/>  Private The group and its projects can only be viewed by members. <input checked="" type="radio"/>  Internal The group and any internal projects can be viewed by any logged in user. <input checked="" type="radio"/>  Public The group and any public projects can be viewed without any authentication.

- A group is a collection of several projects
- Members of a group may only view projects they have permission to access
- Group project URLs are prefixed with the group namespace
- Existing projects may be moved into a group

## 10.2 在 GitLab 创建项目

GitLab 首页 > Create a projects 在 Demo Group 下创建 4 个工程 eureka-server, hello-world-client, hello-world-service, gateway。

Blank project      Create from template      Import project

**Project path**      **Project name**

http://192.168.1.211/ k8s-demo      eu

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  
The project can be accessed without any authentication.

**Create project**      **Cancel**

k8s-demo

Global ▼

Filter by name...      Last created ▼      New project ▼

eureka-server	0 1 day ago
hello-world-client	0 21 hours ago
zuul	0 21 hours ago
hello-world-service	0 21 hours ago
gateway	0 20 hours ago

### 10.3 配置 Group 环境变量

到 GitLab > Groups > k8s-demo > Settings > CI/CD > Variable 里设置.gitlab-ci.yml 里要用到的环境变量：

**DOCKER\_HUB\_REPO:** 保存 Spring Cloud 应用的 docker 镜像库，本文使用 k8s-ci 做为持续集成的库标签： 192.168.1.211:5000/k8s-ci

这个标签会被 spring boot 工程里的.gitlab-ci.yml 用到。

The screenshot shows the 'Variables' section of the CI / CD Settings for the 'k8s-demo' project. It displays a single variable entry:

DOCKER_HUB_REPO	192.168.1.211:5000/k8s-ci
Input variable key	Input variable value

Below the variables, there are two buttons: 'Save variables' (green) and 'Hide value' (blue). The 'Runners settings' section is also visible, with a note to register runners for this group.

## 10.4 提交项目到 GitLab

将 <https://github.com/johnhuang-cn/spring-cloud-k8s-ci-template> 上的 template 项目 pull 到本地，并将 eureka-server, hello-world-client, hello-world-server, gateway 项目分别提交到 gitlab 对应的项目下。

注：访问各自项目，首页会给出参考的git 命令：

```

git config --global user.name "Administrator"
git config --global user.email "admin@example.com"

```

```

git clone http://192.168.1.211/demo/eureka-server.git
cd eureka-server
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master

```

```

cd existing_folder
git init
git remote add origin http://192.168.1.211/demo/eureka-server.git
git add .
git commit -m "Initial commit"
git push -u origin master

```

## 10.5 触发 CI

提交项目文件后，`.gitlab-ci.yml` 会触发 GitLab CI 运行，可以到 GitLab > Admin Area > Jobs 查看打包部署结果。

ID	Branch	Author	Commit Hash	Status	Job Type	Duration	Time Ago	
#178	master	#86 by	a4e0f875	passed	Docker runner#7	deploy	01:20	14 hours ago
#177	master	#80 by	9d100bc8	passed	Docker runner#7	deploy	00:13	19 hours ago
#176	master	#86 by	a4e0f875	passed	Docker runner#7	deploy	00:18	19 hours ago
#175	master	#86 by	a4e0f875	passed	Docker runner#7	build	00:53	19 hours ago
#174	master	#85 by	e624d340	passed	Docker runner#7	deploy	00:18	19 hours ago
#173	master	#85 by	e624d340	passed	Docker runner#7	build	01:36	19 hours ago

如果有运行失败，可以点击进去查看失败原因。全部 job 运行成功后，打开 Minikube Dashboard，可以看到成功的部署列表：

名称	标签	容器组	已创建	镜像
hello-world-server	app: hello-w.. version: 0.0.2	1 / 1	1 天	192.168.1.211
gateway	app: gateway version: 0.0.2	1 / 1	1 天	192.168.1.211
hello-world-client	app: hello-w.. version: 0.0.2	1 / 1	1 天	192.168.1.211
eureka-server	app: eureka-.. version: 0.0.2	1 / 1	1 天	192.168.1.211

打开服务可以查看 gateway 的 NodePort:

名称	标签	集群 IP	内部端点	外部端点
gateway-lb	app: gateway-lb version: 0.0.2	10.105.153.138	-	gateway-lb:80 TCP gateway-lb:31516 TCP
zuul-lb	app: zuul-lb version: 0.0.2	10.107.121.196	-	zuul-lb:80 TCP zuul-lb:32597 TCP
eureka	app: eureka version: 0.0.2	10.107.58.248	-	eureka:8761 TCP eureka:30161 TCP
kubernetes	component: apiserve provider: kubernetes	10.96.0.1	-	kubernetes:443 TCP kubernetes:0 TCP

上面的端口是 31516，所以可以访问: <http://192.168.1.201:31516/api/hello?name=john>，如果一切顺利，应该看到下面的结果:

```

← → ⌂ ⓘ 192.168.1.201:31516/api/hello?name=john
grid 应用 ⌂ Docker Command ⌂ Docker Doc ⌂ dubbo-start

```

Hello john version: v0.0.2

另根据上图，eureka 的 Node Port 是 30161，可以通过 <http://192.168.1.201:30161> 访问

eureka-server 的界面。里面的自保护状态的警告在本地环境下经常会遇到，可以暂时忽略。将来可以通过 health check，多个注册中心增加可用性等配置优化处理掉。

The screenshot shows the 'System Status' section of the Eureka Server interface. It displays various system parameters:

Environment	test
Data center	default
Current time	2018-07-15T06:05:55 +0000
Uptime	13:27
Lease expiration enabled	false
Renews threshold	8
Renews (last min)	8

A prominent red warning message at the bottom states: "EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE."

The 'DS Replicas' section shows instances registered with Eureka:

Application	AMIs	Availability Zones	Status
HELLO-WORLD-CLIENT	n/a (1)	(1)	UP (1) - hello-world-client:-448057865
HELLO-WORLD-SERVICE	n/a (1)	(1)	UP (1) - hello-world-service:8080
SERVICE-GATEWAY	n/a (1)	(1)	UP (1) - gateway-5b888cf6f-jpf2b:service-gateway:80
SERVICE-ZUUL	n/a (1)	(1)	UP (1) - service-zuul:80

## 11.CI 过程相关配置文件说明

下面主要以 eureka-server 项目为例，说明整个 CI 过程中涉及的配置文件，其它组件大同小异。

### 11.1 Trigger

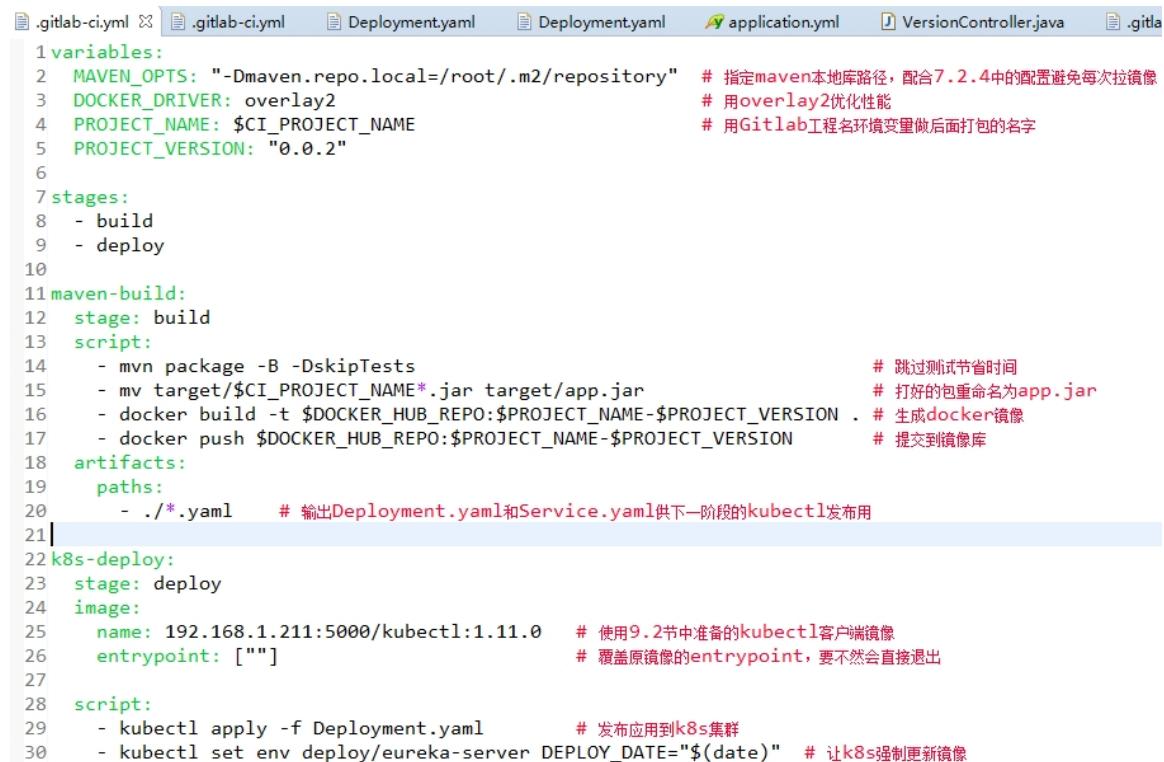
CI 的流程需要一个触发点，在 Gitlab 中，默认的触发点是提交动作。如果该 project 的根目录下存在.gitlab-ci.yml 文件的话，每个提交都会触发 Gitlab 启动 runner 按.gitlab-ci.yml 里的 Job 脚本顺序运行。GitLab 也可以添加自定义 Trigger，可以到各项目 Settings > CI > Pipeline Trigger 里进行设置。

## 11.2.gitlab-ci.yml

关于.gitLab-ci.yml 的详细说明参见：<https://docs.GitLab.com/ee/ci/yaml/README.html>

这里有各种项目的模板：<https://GitLab.com/GitLab-org/GitLab-ci-yml>

下面是 eureka-server 项目的.gitlab-ci.yml 说明（放大看图）：



```
1 variables:
2   MAVEN_OPTS: "-Dmaven.repo.local=/root/.m2/repository"      # 指定maven本地库路径，配合7.2.4中的配置避免每次拉镜像
3   DOCKER_DRIVER: overlay2                                     # 用overlay2优化性能
4   PROJECT_NAME: $CI_PROJECT_NAME                            # 用Gitlab工程名环境变量做后面打包的名字
5   PROJECT_VERSION: "0.0.2"
6
7 stages:
8   - build
9   - deploy
10
11 maven-build:
12   stage: build
13   script:
14     - mvn package -B -DskipTests                           # 跳过测试节省时间
15     - mv target/$CI_PROJECT_NAME*.jar target/app.jar      # 打好的包重命名为app.jar
16     - docker build -t $DOCKER_HUB_REPO:$PROJECT_NAME-$PROJECT_VERSION . # 生成docker镜像
17     - docker push $DOCKER_HUB_REPO:$PROJECT_NAME-$PROJECT_VERSION # 提交到镜像库
18   artifacts:
19     paths:
20       - ./*.yaml    # 输出Deployment.yaml和Service.yaml供下一阶段的kubectl发布用
21
22 k8s-deploy:
23   stage: deploy
24   image:
25     name: 192.168.1.211:5000/kubectl:1.11.0      # 使用9.2节中准备的kubectl客户端镜像
26     entrypoint: [""]                                # 覆盖原镜像的entrypoint，要不然会直接退出
27
28   script:
29     - kubectl apply -f Deployment.yaml             # 发布应用到k8s集群
30     - kubectl set env deploy/eureka-server DEPLOY_DATE="$(date)" # 让k8s强制更新镜像
```

以上配置有几个需要特别说明的：

### 第 2 行 MAVEN\_OPTS:

这个环境变量是让后面的 MAVEN 容器使用指定的目录做本地库路径。这个路径是配合 7.2.4 中的 Volume 配置起作用，两个路径必须一致。这是因为容器是无状态的，每次启动后，原来下载的库都消失了，每次就要重新下载，很耗时间。所以必须将库路径映射到主机的目录上达到持久化的目的，这样就不会每次都拉一遍库。

### 第 3 行 DOCKER\_DRIVER:

GitLab runner 的默认 storage driver 是 vfs，官方建议用 overlay2 可以提升性能。相关说明请参见：[https://docs.gitlab.com/ee/ci/docker/using\\_docker\\_build.html](https://docs.gitlab.com/ee/ci/docker/using_docker_build.html)

### 第 30 行创建服务：

在 K8S 集群里，每次发布应用的 IP 和端口都是随机的，而 eureka 是 Spring Cloud 的注册

中心，所以地址的漂移会使整个 Spring Cloud 无法运作。为了让 eureka 获得一个固定的访问地址，需要借助 kubedns 和 service。Kubedns 会为每个 service 分配一个唯一的名字：`<service name>.<namespace>.svc.cluster.local`。而服务会通过 Deployment.yaml 的 selector 设置找到 eureka-server 应用，这样 Spring Cloud 的其它节点就可以通过 `eureka.default.svc.cluster.local` 这样的域名访问到 eureka-server 注册中心。

### 第 31 行强制更新镜像

请参看后面 11.4 章节的说明。

## 11.3 Dockerfile 说明



```
1 FROM fancybing/java:serverjre-8
2 ADD /target/app.jar app.jar
3 ENTRYPOINT ["java","-jar","/app.jar"]
```

对于 Spring Cloud 的每个应用需要根据 Dockerfile 打包成 Docker 镜像，下面是各行说明：

1. 使用 9.3 中准备的 Oracle Server JRE 镜像，Docker Hub 上的 JDK 镜像大多都是基于 OpenJDK 的，需要注意
2. 将编译的 spring boot 应用 jar 包加入镜像
3. Entrypoint 不能省略，让镜像启动时就自动启动应用。如果没有的话，当 K8S 启动容器时，应用并不能自动启动。

## 11.4 Deployment 文件说明

关于 K8S 的 Deployment.yaml 的更详细说明请参照：

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

下面是 eureka-server 的 Deployment.yaml：

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: eureka-server
5   labels:
6     app: eureka-server
7     version: 0.0.2
8 spec:
9   selector:
10    matchLabels:
11      app: eureka-server
12   replicas: 1
13   template:
14     metadata:
15       labels:
16         app: eureka-server
17     spec:
18       containers:
19         - name: eureka-server
20           image: 192.168.1.211:5000/k8s-ci:eureka-server-0.0.2
21           imagePullPolicy: Always|
```

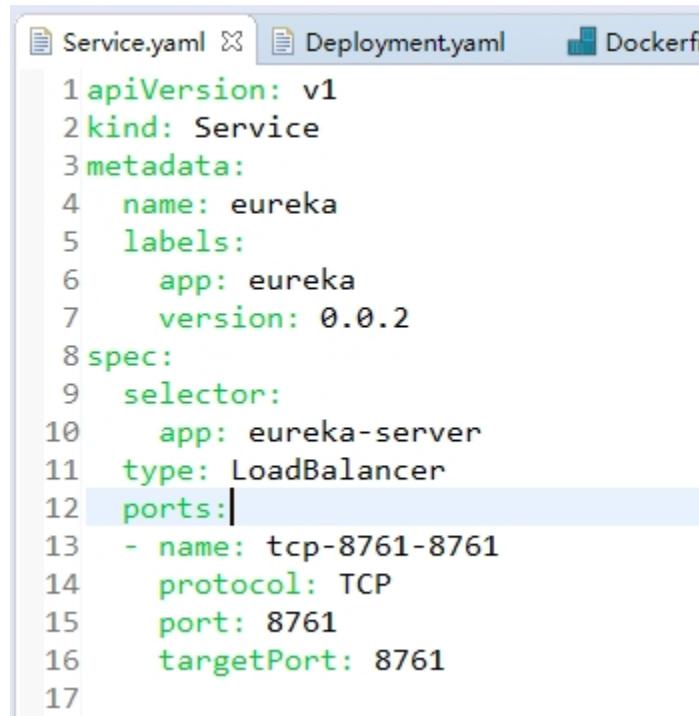
需要注意的是，第 21 行的 `imagePullPolicy` 需要设为 `Always`。目的是让每次更新 Deployment，都会强制更新镜像。这是因为使用 `kubectl` 命令更新 Deployment 的时候，只会更新有变更的部分，而在本文档中，每次 build 的时候，都是使用同一个版本号，也就是说版本号是相对固定的。但这样的话，每次更新 Deployment，镜像都是一样的，`kubectl` 实际上不会真正更新应用。将 `imagePullPolicy` 设为 `Always`，那么只要 Deployment 有任何一项更新，镜像都会重新下载。

那么接着就是有另一个问题，本文档中的 `Deployment.yaml` 也是相对固定的，如果直接用 `kubectl apply` 的话，会认为没有任何更新，也就是说不会有任何运作。所以这就需要另外一个 trick，也就是 11.2 节中的第 31 行，用当前时间更新一下 `deploy date` 属性，这样每次都会触发镜像更新。

不过，它的缺点是，第一次部署的时候，应用会启动两次。上一行 `apply` 命令会启动一次，紧接着一次更新又启动一次。第二次以后的部署，上一行的 `apply` 命令无效，所以只会启动一次。

另一个方案是，每次更新时都生成不同的版本号，但这样会使镜像库里的垃圾太多，毕竟代码提交是很频繁的。

## 11.5 Service 文件说明



```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: eureka
5   labels:
6     app: eureka
7     version: 0.0.2
8 spec:
9   selector:
10    app: eureka-server
11   type: LoadBalancer
12   ports:|
13     - name: tcp-8761-8761
14       protocol: TCP
15       port: 8761
16       targetPort: 8761
17
```

需要对外暴露的服务，比如本文中的 gateway 需要创建一个 LoadBalancer 服务。另外需要 DNS 名字访问的应用，比如本文的 eureka-server 也需要一个服务，这样 Spring Cloud 的其它组件才能按 DNS 名字访问注册。

*注：Deployment.yaml 和 Service.yaml 可以合并成一个，用---隔开。*

## 11.6 Spring Cloud Eureka Client 端配置

Eureka Client 端需要在 application.yml 中使用环境变量来配置注册中心的地址：

```
1 # hello-world-client
2
3 eureka:
4   client:
5     serviceUrl:
6       defaultZone: http://${eureka_host:localhost}:${eureka_port:8761}/eureka/
7     instance:
8       instance-id: ${spring.application.name}:${random.int}
9       prefer-ip-address: true
10
11 server:
12   port: 0
```

同时 eureka client 端的 Deployment.yaml 中需要配置环境变量：

```
17   spec:
18     containers:
19       - name: hello-world-service
20         image: 192.168.1.211:5000/k8s-ci:hello-world-service-0.0.2
21         imagePullPolicy: Always
22       env:
23         - name: eureka_host
24           value: "eureka.default.svc.cluster.local"
25         - name: eureka_port
26           value: "8761"
```

这样在 K8S 里启动的时候，各 Client 端会通过 eureka.default.svc.cluster.local 寻找注册中心，而在本地调试环境启动的时候，会使用 localhost:8761 的注册中心。