

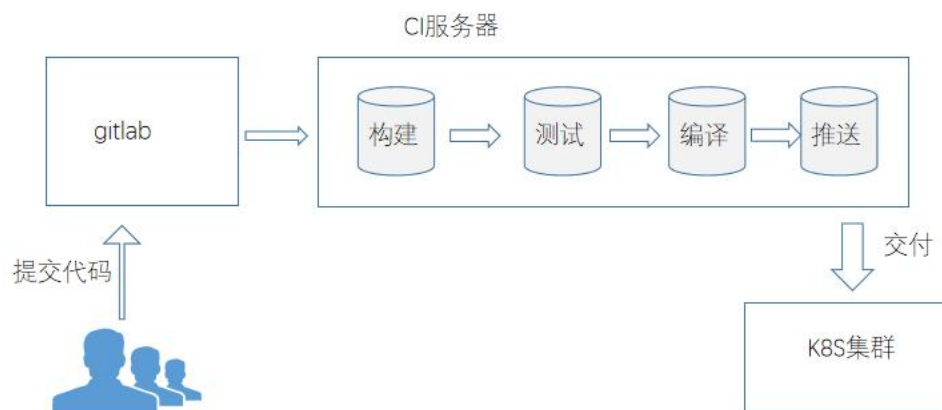
前面我们在 k8s 里部署应用，基本上都是从网络上 pull 下来的镜像。在生产环境里，公司有自己开发的一套应用程序，打包成镜像，然后在 k8s 环境里部署。

这整个过程包括几个步骤：

- 1.软件更新或者迭代
- 2.把新版的软件打包成镜像
- 3.把新的镜像在 k8s 集群里部署

这里有一个问题就是，如果如果软件迭代或者更新频繁，我们就需要不停的对软件进行打包，然后重新部署。

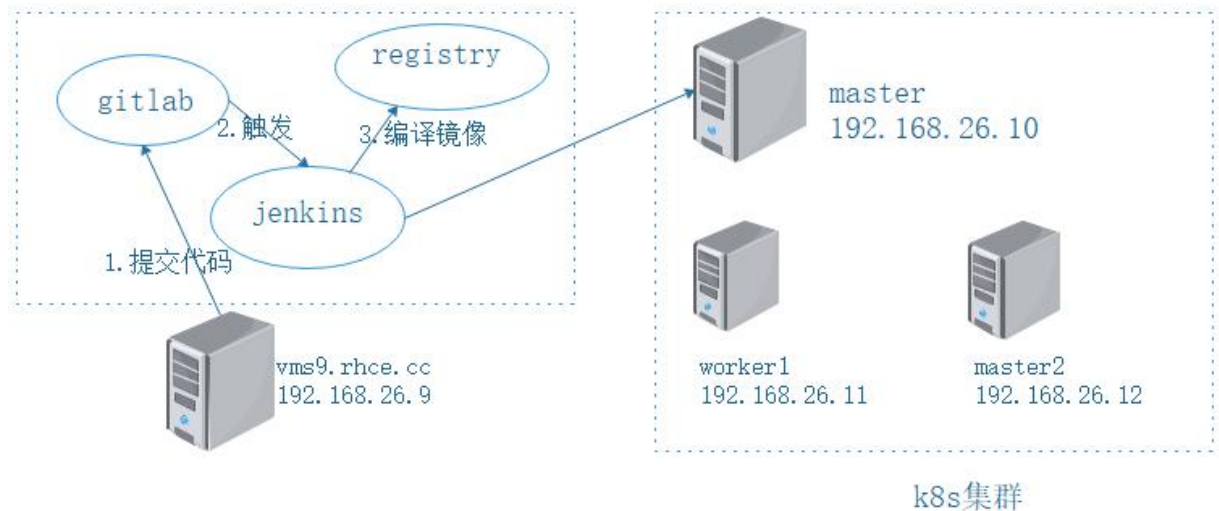
如果有这样的一台服务器，可以帮助我们自动的把软件打包成镜像，之后自动在 k8s 环境里部署新的镜像，这样即使代码频繁迭代，也可以快速的在 k8s 环境里部署，如下图：



当程序员把代码提交到 gitlab 时，马上会触发 CI（持续集成）服务器，开始对这段新的代码进行重新编译成镜像，然后自动在 k8s 里部署（CD 持续交付/部署），这样整个过程就变得简单了很多。

这里 CI 服务器我们可以用 jenkins 来做。

15.1.实验拓扑



此实验里，vms9 是作为客户端（程序员的写代码的地方），vms9 上跑了 3 个容器分别是

- gitlab---作为代码仓库
- jenkins--作为 CI 服务器
- registry--作为镜像仓库

当 vms9 上提交代码到 gitlab 之后，会立马触发 jenkins，会对新代码进行编译成镜像，然后在 vms10 上进行部署新的镜像。

15.2.准备 vms9，并搭建仓库

在 vms9 上安装 docker，并设置修改参数

```
root@vms9 ~]# yum install docker -y
```

已加载插件：fastestmirror

...输出..

作为依赖被升级:

docker-client.x86_64 2:1.13.1-162.git64e9980.el7.centos

docker-common.x86_64 2:1.13.1-162.git64e9980.el7.centos

完毕!

```
[root@vms9 ~]#
```

因为本机器上运行的一个容器会作为镜像仓库，所以需要编辑/etc/sysconfig/docker，在 OPTIONS 里添加--insecure-registry=192.168.26.9:5000，又因为后续创建 jenkins 容器的时候，需要使用 vms9 上安装的 docker，所以需要添加-H tcp://0.0.0.0:2376 -H unix:///var/run/docker.sock，最终的效果变成了：

```
OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false  
--insecure-registry=192.168.26.9:5000 -H tcp://0.0.0.0:2376 -H
```

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

启动 docker 并设置为开机自动启动：

```
[root@vms9 ~]# systemctl enable docker --now
```

```
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
```

```
[root@vms9 ~]#
```

在 vms9 这台机器上下载镜像

```
[root@vms9 ~]# docker pull hub.c.163.com/library/registry:latest
```

```
Trying to pull repository hub.c.163.com/library/registry ...
```

```
latest: Pulling from hub.c.163.com/library/registry
```

```
25728a036091: Pull complete
```

```
...
```

```
[root@vms9 ~]#
```

另外请自行把 nginx 镜像下载下来。

创建容器作为 docker 镜像仓库

```
[root@vms32 ~]# docker run -d --name registry -p 5000:5000 --restart=always -v /myreg:/var/lib/registry hub.c.163.com/library/registry
```

```
aea0cc63e2fa0b6529c7419a9190c324587e94cd12ecb40178b129fddb6f1605
```

```
[root@vms32 ~]#
```

15.3.修改 k8s 集群中所有节点的设置

因为我们环境里即将使用的仓库地址为 192.168.26.9:5000，所以在所有 vms10~vms12 三台机器上修改/etc/sysconfig/docker，在 OPTIONS 里添加

--insecure-registry=192.168.26.9:5000，内容变为：

```
OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false  
--insecure-registry=192.168.26.9:5000'
```

然后在三台机器上重启 docker:

```
systemctl restart docker
```

15.4.安装 gitlab 并配置

gitlab

先下载 gitlab 中文版的镜像

```
[root@vms9 ~]# docker pull beginor/gitlab-ce
```

```
Using default tag: latest
```

```
Trying to pull repository docker.io/beginor/gitlab-ce ...
```

```
latest: Pulling from docker.io/beginor/gitlab-ce
```

```
...输出...
```

```
Status: Downloaded newer image for docker.io/beginor/gitlab-ce:latest
```

```
[root@vms9 ~]#
```

部署 gitlab 容器

```
[root@vms9 ~]# mkdir -p /data/gitlab/etc /data/gitlab/log /data/gitlab/data
[root@vms9 ~]# chmod 777 /data/gitlab/etc /data/gitlab/log /data/gitlab/data
[root@vms9 ~]#
[root@vms9 ~]# docker run -dit --name=gitlab --restart=always -p 8443:443 -p 80:80 -p
222:22 -v /data/gitlab/etc:/etc/gitlab -v /data/gitlab/log:/var/log/gitlab -v
/data/gitlab/data:/var/opt/gitlab --privileged=true beginor/gitlab-ce
4d6c98cffb6e9d5f0bce4f7e34070d74def333b2564cca7f74d18bd0c5e45862
[root@vms9 ~]#
```

在创建此容器时，因为使用了数据卷，所以 gitlab 容器的配置也都保存在服务器 vms9 的相关目录上了。因为我们需要修改 gitlab 的配置并让其生效，所以大概 1 分钟之后，先关闭此容器：

```
[root@vms9 ~]# docker stop gitlab
gitlab
[root@vms9 ~]#
```

修改一下两处位置：

1.用 vim 编辑器修改/data/gitlab/etc/gitlab.rb 以下几处内容：

```
external_url 'http://192.168.26.9'
gitlab_rails['gitlab_ssh_host'] = '192.168.26.9'
gitlab_rails['gitlab_shell_ssh_port'] = 222
```

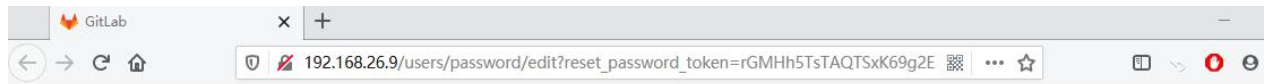
2.用 vim 编辑修改/data/gitlab/data/gitlab-rails/etc/gitlab.yml 以下内容：

```
11  gitlab:
12    ## Web server settings (note: host is the FQDN, do not include http://)
13    host: 192.168.26.9
14    port: 80
15    https: false
```

然后启动容器

```
[root@vms9 ~]# docker start gitlab
gitlab
[root@vms9 ~]#
```

在浏览器里输入 192.168.26.9，会让我们为 root 用户设置新的密码



请为您的新帐户创建密码。

GitLab 中文社区版

用于代码协作的开源软件

细粒度访问控制管理 git 仓库以保证代码安全。使用合并请求进行代码审查并加强团体合作。每个项目均有自己的问题跟踪和维基页面。

修改密码

新密码

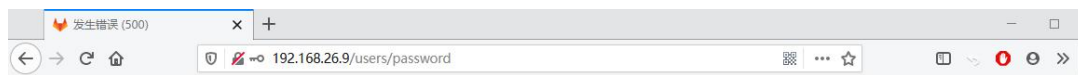
确认新密码

修改密码

没有收到确认邮件？ [重新发送](#)

已有账号和密码？ [登录](#)

如果密码设置的没有满足一定的复杂性，则会有如下报错



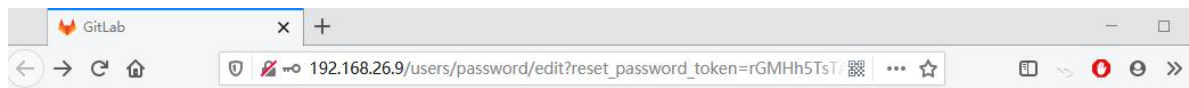
500

哇，因为我们的原因发生错误。

请尝试刷新页面，或者回退到上一页面再操作。

如果此问题继续存在请联系 GitLab 管理员。

重新设置新的密码，满足一定的复杂性，然后点击修改密码



请为您的新帐户创建密码。

GitLab 中文社区版

用于代码协作的开源软件

细粒度访问控制管理 git 仓库以保证代码安全。使用合并请求进行代码审查并加强团体合作。每个项目均有自己的问题跟踪和维基页面。

修改密码

新密码

.....

确认新密码

.....

修改密码

没有收到确认邮件？ [重新发送](#)

已有账号和密码？ [登录](#)

输入 root 和刚刚设置过的密码，点击登录：



您的密码已修改成功。

GitLab 中文社区版

用于代码协作的开源软件

细粒度访问控制管理 git 仓库以保证代码安全。使用合并请求进行代码审查并加强团体合作。每个项目均有自己的问题跟踪和维基页面。

登录

注册

用户名或邮箱

root

密码

.....

☒ 记住我

[忘记密码？](#)

登录

尚未收到确认邮件？ [重新发送确认邮件。](#)

登录之后，点击创建一个项目

欢迎来到 GitLab

集代码，测试和部署于一体。



然后在项目名称位置写入 p1，可见登记选中公开，然后点击创建项目

空白项目	从模板创建	导入项目
<div><div>项目路径</div><div>http://192.168.26.9/ root</div></div> <div><div>项目名称</div><div>p1</div></div> <p>希望将几个相关联的项目放置于同一个命名空间下？ 创建群组</p> <div><div>项目描述 (可选)</div><div>说明格式</div></div> <div><div>可见等级</div><div><div><input type="radio"/> 私有</div><div>项目访问权限必须明确授权给每个用户。</div><div><input type="radio"/> 内部</div><div>该项目允许已登录的用户访问。</div><div><input checked="" type="radio"/> 公开</div><div>该项目允许任何人访问。</div></div></div> <div><div>创建项目</div><div>取消</div></div>		

然后点击复制按钮获取 clone 的链接：



在客户端上测试

用命令 `yum install git -y` 安装 git 客户端软件, 然后用 `git clone` 把此项目的版本库克隆下来:

```
[root@vms9 ~]# git clone http://192.168.26.9/root/p1.git
```

正克隆到 'p1'...

warning: 您似乎克隆了一个空版本库。

```
[root@vms9 ~]#
```

设置一些变量

```
[root@vms9 ~]# cd p1/
```

```
[root@vms9 p1]# git config --global user.name "lduan"
```

```
[root@vms9 p1]# git config --global user.email lduan@example.com
```

```
[root@vms9 p1]# git config --global push.default simple
```

```
[root@vms9 p1]#
```

创建 index.html 并推送到代码仓库

```
[root@vms9 p1]# echo 1111 > index.html
```

```
[root@vms9 p1]# git add .
```

```
[root@vms9 p1]# git commit -m 111
```

```
[root@vms9 p1]# git push
```

Username for 'http://192.168.26.9': root

Password for 'http://root@192.168.26.9':

Counting objects: 3, done.

Writing objects: 100% (3/3), 207 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To http://192.168.26.9/root/p1.git

* [new branch] master -> master

```
[root@vms9 p1]#
```

至此, gitlab 配置完毕。

15.5.jenkins 安装

把 jenkins 的镜像下载下来

```
[root@vms9 ~]# docker pull jenkins/jenkins:2.249.1-lts-centos7
```

Trying to pull repository docker.io/jenkins/jenkins ...

2.249.1-lts-centos7: Pulling from docker.io/jenkins/jenkins

...大量输出...

Status: Downloaded newer image for docker.io/jenkins/jenkins:2.249.1-lts-centos7

```
[root@vms9 ~]#
```

创建数据卷所需要的目录，并把所有者和所属组改为 1000

```
[root@vms9 ~]# mkdir /jenkins ; chown 1000.1000 /jenkins
```

```
[root@vms9 ~]#
```

这里为什么要改成 1000，是因为容器里是以 jenkins 用户的身份去读写数据，而在容器里 jenkins 的 uid 是 1000，可以看下此镜像的 Dockerfile 内容：

1	ADD file ... in /	72.35 MB
2	LABEL org.label-schema.schema-version=1.0 org.label-sche...	0 B
3	CMD ["/bin/bash"]	0 B
4	/bin/sh -c yum update -y	107.27 MB
5	ENV JAVA_HOME=/etc/alternatives/jre_openjdk	0 B
6	ARG user=jenkins	0 B
7	ARG group=jenkins	0 B
8	ARG uid=1000	0 B
9	ARG gid=1000	0 B

图片来自

<https://hub.docker.com/layers/jenkins/jenkins/2.249.2-lts-centos7/images/sha256-629136f7d2c2b479cda1198b3a50407cf07d969e8b136028b5bdb2ba717d0ac?context=explore>

创建 jenkins 容器

```
[root@vms9 ~]# docker run -dit -p 8080:8080 -p 50000:50000 --name jenkins
```

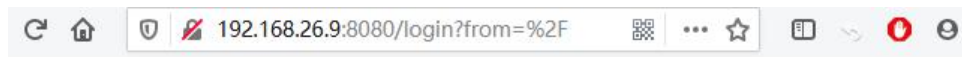
```
--privileged=true --restart=always -v /jenkins:/var/jenkins_home
```

```
jenkins/jenkins:2.249.1-lts-centos7
```

```
44a32750c94c400c9e1ddfcacd692f514850ca84bef7f36fe897ce0593a4cb5f
```

```
[root@vms9 ~]#
```

打开浏览器，输入 192.168.26.9:8080



Please wait while Jenkins is getting ready to work ...

Your browser will reload automatically when Jenkins is ready.

记住，此时一定要先打开浏览器打开这个页面，让其初始化一下，直到看到界面



因为要修改 jenkins 的配置，所以此时关闭 jenkins 容器

```
[root@vms9 ~]# docker stop jenkins
```

```
jenkins
```

```
[root@vms9 ~]#
```

然后用 vim 编辑打开 `/jenkins/hudson.model.UpdateCenter.xml`，按如下修改

```
<?xml version='1.1' encoding='UTF-8'?>
<sites>
  <site>
    <id>default</id>
    <url>https://updates.jenkins.io/update-center.json</url>
  </site>
</sites>
```

改为

```
<?xml version='1.1' encoding='UTF-8'?>
<sites>
  <site>
    <id>default</id>
    <url>http://mirrors.tuna.tsinghua.edu.cn/jenkins</url>
  </site>
</sites>
```

用 vim 编辑器打开 /jenkins/updates/default.json，按如下修改

```
{"connectionCheckUrl":"http://www.google.com/"
```

改成

```
{"connectionCheckUrl":"http://www.baidu.com/"
```

再次启动 jenkins

```
[root@vms9 ~]# docker start jenkins
```

```
jenkins
```

```
[root@vms9 ~]#
```

再次在浏览器里输入 192.168.26.9:8080



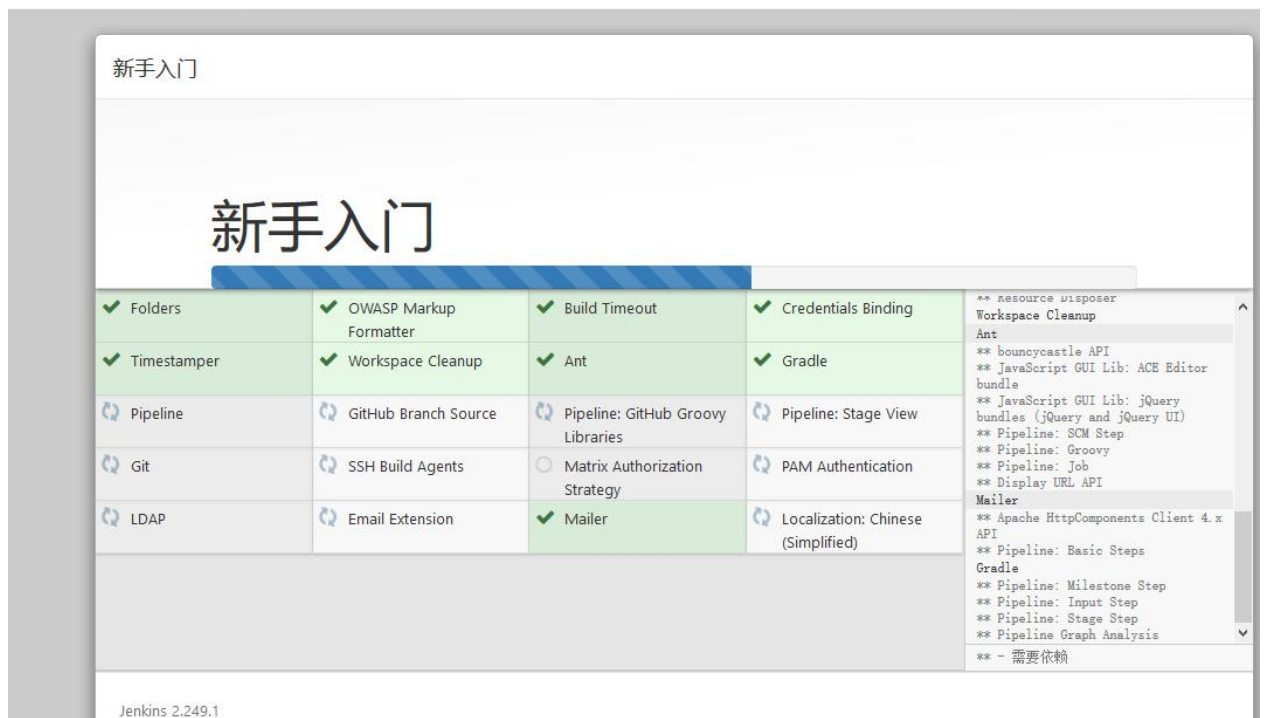
```
[root@vms9 ~]# cat /jenkins/secrets/initialAdminPassword
```

```
728ebaec3d154fb6afb04fc0e232842
```

```
[root@vms9 ~]#
```



直到所有插件全部安装完成



填写必要的网络信息，点击保存并完成

创建第一个管理员用户

用户名:	<input type="text" value="admin"/>
密码:	<input type="password" value="•••••"/>
确认密码:	<input type="password" value="•••••"/>
全名:	<input type="text" value="admin"/>
电子邮件地址:	<input type="text" value="aa@rhce.cc"/>

Jenkins 2.249.1

[使用admin账户继续](#)

[保存并完成](#)

再一次点击保存并完成，然后点击开始使用 jenkins

Jenkins已就绪！

Jenkins安装已完成。

[开始使用Jenkins](#)

15.6.安装 docker 插件

在 jenkins 主页面依次点击左侧的系统管理-插件管理-可选插件，在搜索栏搜索 docker，选中 docker 和 docker-build-step，然后点击下面的直接安装

<input checked="" type="checkbox"/>	Docker
	云提供商 集群管理和分布式构建 docker
	This plugin integrates Jenkins with Docker
<input checked="" type="checkbox"/>	docker-build-step
	构建工具 docker
	This plugin allows to add various docker commands to your job as build steps.

点击下面的直接安装

安装/更新 插件中

准备

- Checking internet connectivity
- Checking update center connectivity
- Success

Authentication Tokens API

● 完成

Docker Commons

● 完成

Docker API

● 完成

Docker

● 完成

docker-build-step

● 完成

Loading plugin extensions

● Success


返回首页

(返回首页使用已经安装好的插件)

☐

安装完成后重启Jenkins(空闲时)

点击返回首页，再依次点击系统管理-节点管理-Configure Clouds

配置集群


Add a new cloud ^

Docker

Save

Apply

在 add a new cloud 里选择 docker 之后页面跳转到如下页面

配置集群

Docker

Name

docker

Docker Cloud details...

Docker Agent templates...

Add a new cloud ▾

Save

Apply

Delete cloud

点击 docker cloud details, 输入 tcp://192.168.26.9:2376, 然后点击 test connection

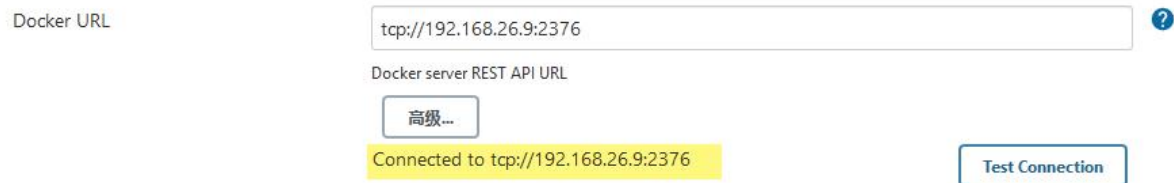
配置集群



可以看到当前 docker 的信息, 点击最下面的 save。

在 jenkins 首页, 依次点击系统管理-系统配置, 找到 docker build, 在 docker build 里输入 tcp://192.168.26.9:2376, 点击 test connection:

Docker Builder



点击最下面的保存, 这样 jenkins 就和 docker 关联起来了。

15.7.jenkins 安全设置

后面 gitlab 要和 jenkins 进行联动, 所以必须要需要对 jenkins 的安全做一些设置, 依次点击系统管理-全局安全配置-授权策略, 勾选"匿名用户具有可读权限"

授权策略



注意下面的的跨站点伪造保护(CSFR)请求必须要关闭, 但是在 Jenkins 版本自 2.2xx 版本之后, 在 web 界面里已经没法关闭了:

跨站请求伪造保护

Crumb Issuer

默认碎片生成器

默认碎片生成器

所以在当前 web 界面里暂且不要管它，点击下面的保存。

gitlab 要触发 jenkins 的话，就必须关闭跨站点伪造请求，web 界面里已经没法关闭了，所以要需要做如下设置

```
[root@vms9 ~]# docker exec -u root -it jenkins bash
[root@44a32750c94c /]#
```

```
[root@44a32750c94c /]# vi /usr/local/bin/jenkins.sh
```

找到 exec java 那行(大概是在第 37 行)，添加

```
-Dhudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION=true
```

```
exec java -Duser.home="$JENKINS_HOME"
```

```
-Dhudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION=true
```

```
"${java_opts_array[@]}" -jar ${JENKINS_WAR} "${jenkins_opts_array[@]}" "$@"
```

```
[root@44a32750c94c /]# exit
```

exit

```
[root@vms9 ~]#
```

然后重启 jenkins 容器

```
[root@vms9 ~]# docker restart jenkins
```

jenkins

```
[root@vms9 ~]#
```

再次登录到 web 界面查看 跨站点伪造请求的设置：

跨站请求伪造保护

This configuration is unavailable because the System property

hudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION is set to true.

That option should be considered unsupported and its use should be limited to working around compatibility problems until they are resolved.

这里已经是关闭了。

15.8.拷贝 kubeconfig 文件

在 vms9 上下载和当前 k8s 匹配的 kubectl，这里是 v1.19.2 版本

wget

<https://storage.googleapis.com/kubernetes-release/release/v1.19.2/bin/linux/amd64/kubectl>

并设置为可执行权限：

```
[root@vms9 ~]# chmod +x kubectl
```

```
[root@vms9 ~]#
```


把前面安全管理里创建过的 kubeconfig 文件 kc1 拷贝到 vms9 上

```
[root@vms9 ~]# scp 192.168.26.10:~/role/kc1 .
root@192.168.26.10's password:
kc1          100% 5506      3.9MB/s   00:00
[root@vms9 ~]# ls
anaconda-ks.cfg  kc1  kubectl  p1  set.sh
[root@vms9 ~]#
```

然后把这两个文件拷贝到 jenkins 容器里:

```
[root@vms9 ~]# docker cp kubectl jenkins:/
[root@vms9 ~]# docker cp kc1 jenkins:/
[root@vms9 ~]#
```

到 jenkins 容器里进行测试:

```
[root@vms9 ~]# docker exec -it jenkins bash
bash-4.2$
bash-4.2$ /kubectl --kubeconfig=/kc1 get nodes
NAME                STATUS    ROLES    AGE   VERSION
vms10.rhce.cc       Ready    master   12d   v1.19.2
vms11.rhce.cc       Ready    <none>   12d   v1.19.2
vms12.rhce.cc       Ready    <none>   12d   v1.19.2
bash-4.2$ exit
exit
[root@vms9 ~]#
```

在 vms10 上为 web1 创建一个类型为 NodePort 的 svc:

```
[root@vms10 ~]# kubectl get deploy -n net -o wide
NAME    READY    UP-TO-DATE    AVAILABLE    AGE    CONTAINERS    IMAGES
web1    2/2      2              2            94s    nginx         nginx
[root@vms10 ~]#
[root@vms10 ~]# kubectl expose deployment web1 --port=80 --type=NodePort
service/web1 exposed
[root@vms10 ~]#
[root@vms10 ~]# kubectl get svc web1
NAME    TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
web1    NodePort    10.109.15.46  <none>         80:31407/TCP   10s
[root@vms10 ~]#
```

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

15.9.创建项目

点击新建任务，任务名称可以自定义，这里设置为 devops001，并点击一下 构建一个自由风格的软件项目，点击确定

输入一个任务名称

devops001

» 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

流水线
精心地组织一个可以长期运行在多个节点上的任务。适用于构建的任务类型。

构建一个多配置项目
适用于多配置项目,例如多环境测试,平台指定构建,等等。

确定

构建触发器里，选中触发远程构建(例如,使用脚本)，在身份验证令牌里输入 123123

☒ 触发远程构建 (例如,使用脚本)

身份验证令牌 123123

Use the following URL to trigger build remotely: JENKINS_URL/job/devops001/build?token=123123

Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

特别注意下面这段链接 JENKINS_URL/job/devops001/build?token=123123，这个连用于 gitlab 在触发 jenkins 时能用到的链接，我们这里，TOKEN_NAME 的值是 123123，JENKINS_URL 是 192.168.26.9:8080，所以整个链接为：
<http://192.168.26.9:8080/job/devops001/build?token=123123>

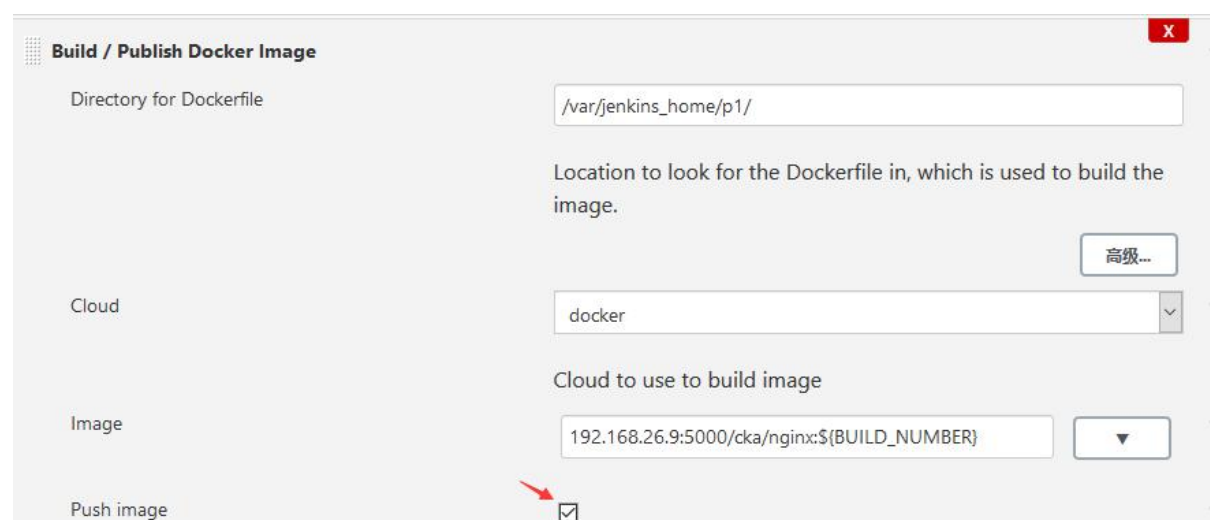
在构建-增加构建步骤里选择执行 shell，在里面输入如下内容

```
cd ~  
rm -rf p1  
git clone http://192.168.26.9/root/p1.git
```

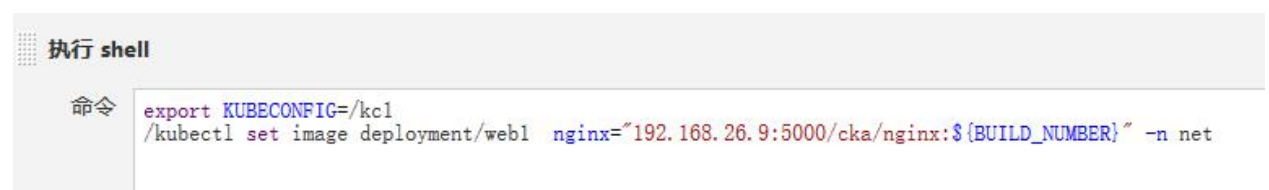
如下面这样：



再次增加构建步骤，在构建-增加构建步骤里选择 build/publish docker image，根据下图填写：
directory for dockerfile : /var/jenkins_home/p1/ 这里写的是容器里的目录
image: 192.168.26.9:5000/cka/nginx:\${BUILD_NUMBER}
之后如下图



再次增加构建步骤，在构建-增加构建步骤里选择执行 shell，里面输入
export KUBECONFIG=/kc1
/kubectl set image deployment/web1
nginx="192.168.26.9:5000/cka/nginx:\${BUILD_NUMBER}" -n net
效果如下图：



点击保存

15.10.配置 gitlab 和 jenkins 的联动

在 gitlab 配置页面，点击最上层的扳手图标



然后点击左侧最下方的设置，然后展开 Outbound requests，选中 允许钩子和服务访问本地网络，如下图，然后点击保存修改

Outbound requests

Allow requests to the local network from hooks and services.

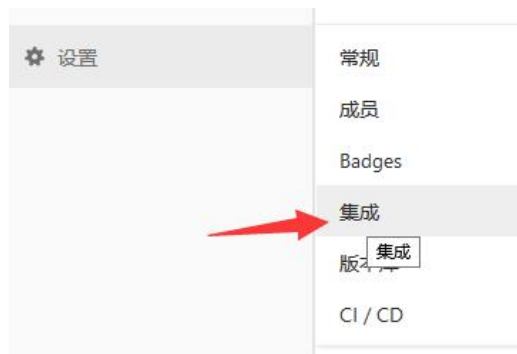
☒ 允许钩子和服务访问本地网络

保存修改

再依次点击项目-您的项目



进入到 p1 项目，点击左侧的设置-集成：



在集成-连接里输入 `http://192.168.26.9:8080/job/devops001/build?token=123123`，这个地址是在 jenkins 里创建项目时构建触发器时得到的连接

集成

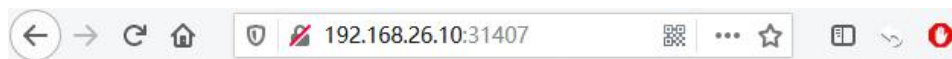
Web 钩子 可以绑定项目发生的事件。

链接(URL)

http://192.168.26.9:8080/job/devops001/build?token=123123

增加 web 钩子

15.11.实战练习



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

在 vms9 上下载 dockerfile 所需要的基镜像

```
[root@vms9 p1]# docker pull nginx
```

Using default tag: latest

Trying to pull repository docker.io/library/nginx ...

...输出...

Status: Downloaded newer image for docker.io/nginx:latest

```
[root@vms9 p1]#
```

在 vms9 上 git clone 下来的目录 p1 里，创建 Dockerfile 文件内容如下

```
[root@vms9 p1]# cat Dockerfile
```

```
FROM docker.io/nginx
```

```
MAINTAINER lduan
```

```
ADD index.html /usr/share/nginx/html/
```

```
EXPOSE 80
```

```
CMD ["nginx", "-g","daemon off;"]
```

```
[root@vms9 p1]#
```

开始提交代码

```
[root@vms9 p1]# git add .
```

```
[root@vms9 p1]# git commit -m '22'
```

```
[master 127d00c] 22
```

```
1 file changed, 5 insertions(+)
```

```
create mode 100644 Dockerfile
```

```
[root@vms9 p1]# git push
```

```
Username for 'http://192.168.26.9': root
```

```
Password for 'http://root@192.168.26.9':
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 375 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To http://192.168.26.9/root/p1.git
    968de38..127d00c  master -> master
[root@vms9 p1]#
```

切换至 jenkins，可以看到已经编译成功：



点击控制台输出，可以看到完成编译过程：

控制台输出

```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/devops001
[devops001] $ /bin/sh -xe /tmp/jenkins8687722728468676620.sh
+ cd /var/jenkins_home
+ rm -rf p1
+ git clone http://192.168.26.9/root/p1.git
Cloning into 'p1'...
Docker Build
Docker Build: building image at path /var/jenkins_home/p1
Step 1/5 : FROM docker.io/nginx

...输出...
[devops001] $ /bin/sh -xe /tmp/jenkins6078903812628512641.sh
+ export KUBECONFIG=/kc1
+ KUBECONFIG=/kc1
+ /kubectl set image deployment/web1 nginx=192.168.26.9:5000/cka/nginx:3 -n net
deployment.apps/web1 image updated
Finished: SUCCESS
```

切换会浏览器



1111