K8s+jenkins+github+springboot 环境搭建

By Wuyong at 2021-01-12

目录

一、K8s 集群安装

1、所有机器上执行以下命令，准备安装环境：(注意是所有机器，主机 master，从机 node 都要安装)

1.1、安装 epel-release 源

```
yum -y install epel-release
```

1.2、所有机器关闭防火墙

```
systemctl stop firewalld
systemctl disable firewalldsetenforce 0

#查看防火墙状态
firewall-cmd --state
```

2、现在开始 master 主机上 192.168.26.227 安装 kubernetes Master

2.1、使用 yum 安装 etcd、kubernetes-master

```
yum -y install etcd kubernetes-master
```

2.2、编辑：vi /etc/etcd/etcd.conf 文件，修改结果如下：

```
#[Member]
#ETCD_CORS=""
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
#ETCD_WAL_DIR=""
#ETCD_LISTEN_PEER_URLS="http://localhost:2380"
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
ETCD_NAME="default"
#ETCD_SNAPSHOT_COUNT="100000"
#ETCD_HEARTBEAT_INTERVAL="100"
#ETCD_ELECTION_TIMEOUT="1000"
#ETCD_QUOTA_BACKEND_BYTES="0"
#ETCD_MAX_REQUEST_BYTES="1572864"
#ETCD_GRPC_KEEPALIVE_MIN_TIME="5s"
#ETCD_GRPC_KEEPALIVE_INTERVAL="2h0m0s"
#ETCD_GRPC_KEEPALIVE_TIMEOUT="20s"
#
#[Clustering]
#ETCD_INITIAL_ADVERTISE_PEER_URLS="http://localhost:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://localhost:2379"
#ETCD_DISCOVERY=""
#ETCD_DISCOVERY_FALLBACK="proxy"
#ETCD_DISCOVERY_PROXY=""
#ETCD_DISCOVERY_SRV=""
#ETCD_INITIAL_CLUSTER="default=http://localhost:2380"
#ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
#ETCD_INITIAL_CLUSTER_STATE="new"
#ETCD_STRICT_RECONFIG_CHECK="true"
#ETCD_ENABLE_V2="true"
#
#[Proxy]
"/etc/etcd/etcd.conf" 69L, 1684C
```

2.3、配置：vi /etc/kubernetes/apiserver 文件，配置结果如下：

```
1 192.168.26.227   2 192.168.26.228   3 192.168.26.233
###
# kubernetes system config
#
# The following values are used to configure the kube-apiserver
#

# The address on the local server to listen to.                    配置完之后截的图
KUBE_API_ADDRESS="--insecure-bind-address=0.0.0.0"

# The port on the local server to listen on.
KUBE_API_PORT="--port=8080"

# Port minions listen on
KUBELET_PORT="--kubelet-port=10250"

# Comma separated list of nodes in the etcd cluster
KUBE_ETCD_SERVERS="--etcd-servers=http://127.0.0.1:2379"

# Address range to use for services
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16"

# default admission control policies
KUBE_ADMISSION_CONTROL="--admission-control=NamespaceLifecycle,NamespaceExists,LimitRanger,SecurityConte

# Add your own!
KUBE_API_ARGS=""
~
~
~
~
~
~
-- INSERT --
ssh://192.168.26.227:22
```

2.4、启动 etcd、kube-apiserver、kube-controller-manager、kube-scheduler 等服务，并设置开机启动。

```
for SERVICES in etcd kube-apiserver kube-controller-manager
kube-scheduler;
do
systemctl restart $SERVICES;
systemctl enable $SERVICES;
systemctl status $SERVICES ;
done
```

2.5、在 etcd 中定义 flannel 网络

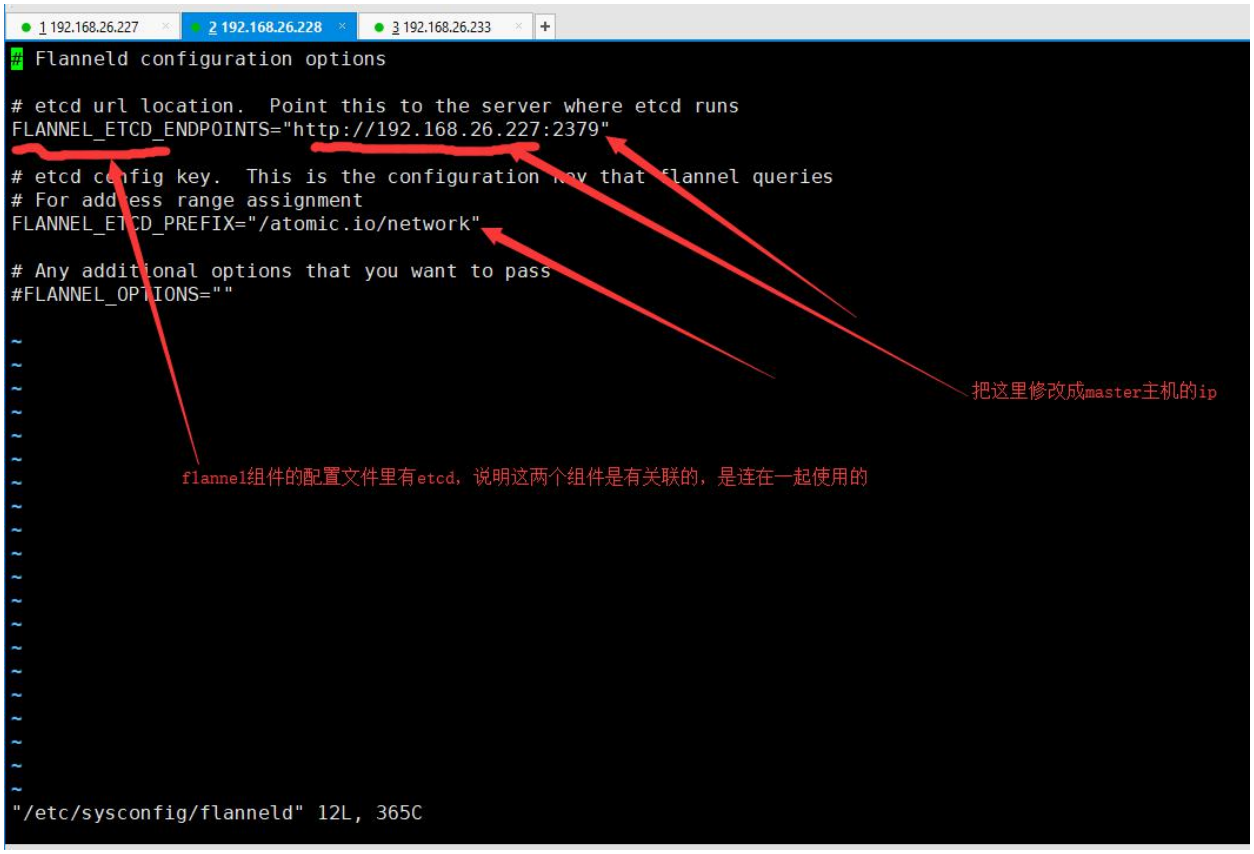
```
etcdctl mk /atomic.io/network/config '{"Network":"172.17.0.0/16"}'
```

3、接下来弄 node 从机上的配置安装什么的

3.1、在 node 机上 192.168.26.228 安装 kubernetes Node 和 flannel 组件应用

```
yum -y install flannel kubernetes-node
```

3.2、为 flannel 网络指定 etcd 服务，修改/etc/sysconfig/flanneld 文件，配置结果如下图：



3.3、修改：vi /etc/kubernetes/config 文件，配置结果如下图：

```
1 192.168.26.227    2 192.168.26.228    3 192.168.26.233
###
# kubernetes system config
#
# The following values are used to configure various aspects of all
# kubernetes services, including
#
#   kube-apiserver.service
#   kube-controller-manager.service
#   kube-scheduler.service
#   kubelet.service
#   kube-proxy.service
# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"

# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"

# Should this cluster be allowed to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=false"

# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=http://192.168.26.227:8080"
~
~
~
~
~
~
~
~
~
```

这里配置成mastert主机的ip:端口,192.168.26.227:8080

3.4、修改 node 机的 kubelet 配置文件/etc/kubernetes/kubelet

```
1 192.168.26.233   2 192.168.26.227   3 192.168.26.228   4 192.168.26.108
###
# kubernetes kubelet (minion) config

# The address for the info server to serve on (set to 0.0.0.0 or "" for all interfaces)
KUBELET_ADDRESS="--address=0.0.0.0"                                   改成0.0.0.0

# The port for the info server to serve on
KUBELET_PORT="--port=10250"

# You may leave this blank to use the actual hostname
KUBELET_HOSTNAME="--hostname-override=192.168.26.228"                 改成当前此node机的ip

# location of the api-server
KUBELET_API_SERVER="--api-servers=http://192.168.26.227:8080"         改成主机master机的ip：端口

# pod infrastructure container
KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructur

# Add your own!
KUBELET_ARGS=""
~
~
~
~
~
~
~
~
~
~
~
```

3.5、node 节点机上启动 kube-proxy,kubelet,docker,flanneld 等服务，并设置开机启动。

```
for SERVICES in kube-proxy kubelet docker flanneld;
do
systemctl restart $SERVICES;
systemctl enable $SERVICES;
systemctl status $SERVICES;
done
```

在 master 主机上 192.168.26.227 执行如下命令，查看运行的 node 节点机器：

```
kubectl get nodes
```

成功了，结果图如下：

```
[root@localhost local]# kubectl get nodes
NAME              STATUS    AGE
192.168.26.228    Ready     19s
```

二、jenkins 安装

1、安装 JDK

```
yum install -y java
```

2、安装 jenkins

添加 Jenkins 库到 yum 库，Jenkins 将从这里下载安装。

```
wget -O /etc/yum.repos.d/jenkins.repo
http://pkg.jenkins-ci.org/redhat/jenkins.repo2 rpm --import
https://jenkins-ci.org/redhat/jenkins-ci.org.key3 yum install -y
jenkins
```

如果不能安装就到官网下载 jenkis 的 rmp 包，官网地址（http://pkg.jenkins-ci.org/redhat-stable/）

```
wget
http://pkg.jenkins-ci.org/redhat-stable/jenkins-2.7.3-1.1.noarch.rpm2
rpm -ivh jenkins-2.7.3-1.1.noarch.rpm
```

配置 jenkis 的端口

```
vi /etc/sysconfig/jenkins
```

找到修改端口号：

JENKINS_PORT="8080" #此端口不冲突可以不修改

3、启动 jenkins

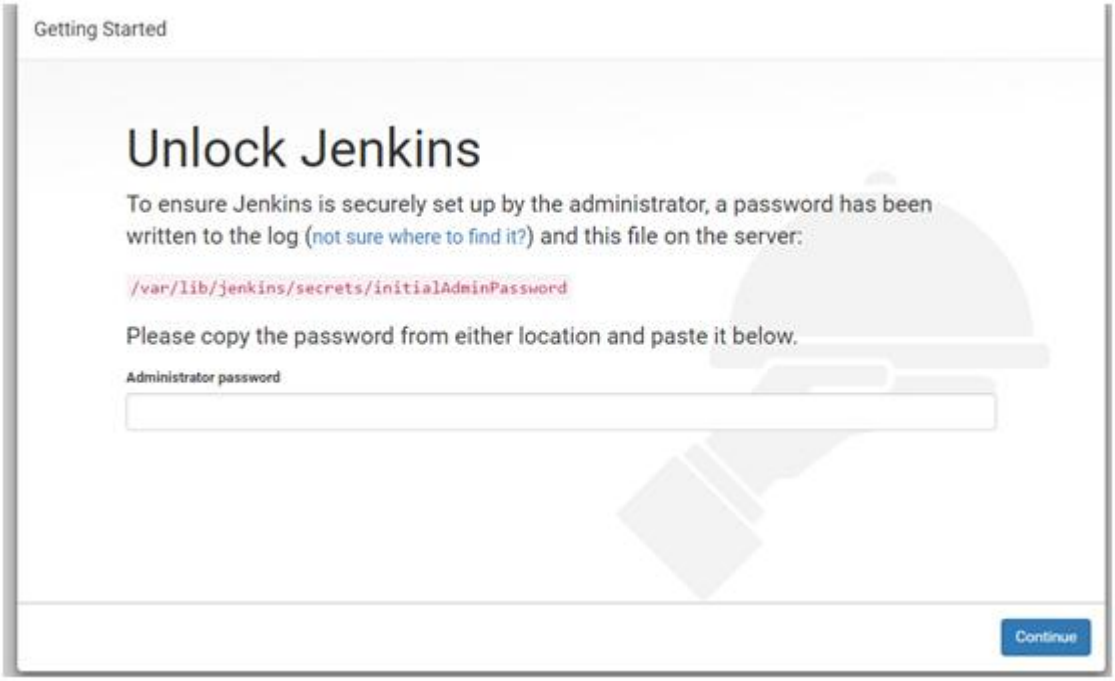
```
service jenkins start/stop/restart
```

- 安装成功后 Jenkins 将作为一个守护进程随系统启动

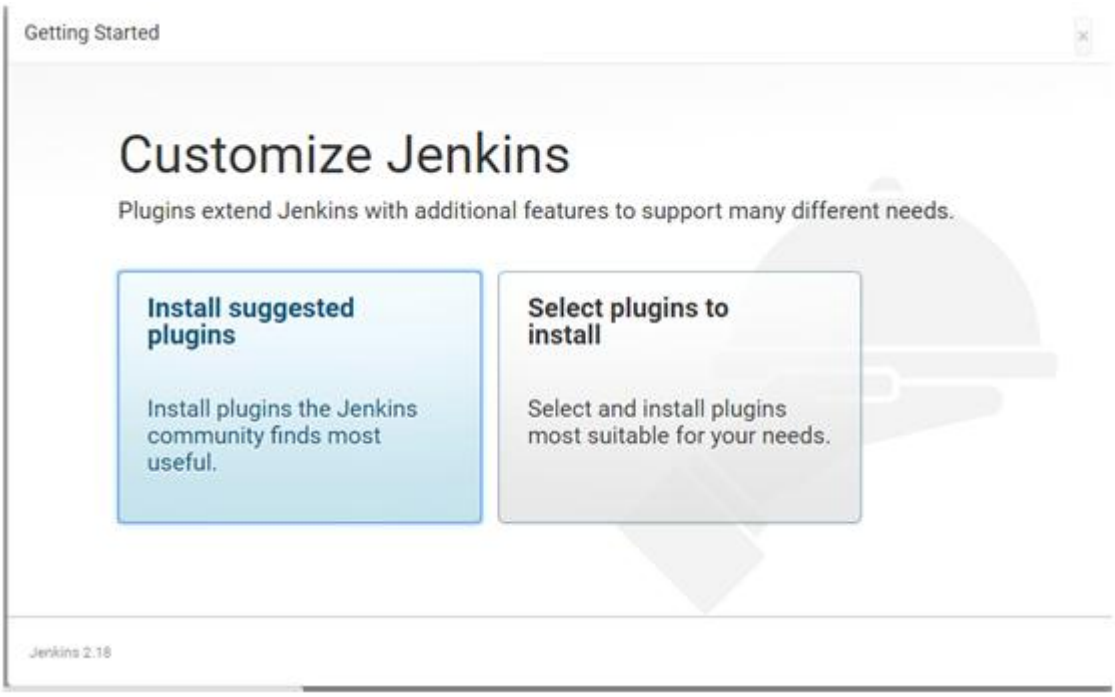
- 系统会创建一个“jenkins”用户来允许这个服务，如果改变服务所有者，同时需要修改/var/log/jenkins, /var/lib/jenkins, 和/var/cache/jenkins 的所有者
- 启动的时候将从/etc/sysconfig/jenkins 获取配置参数
- 默认情况下，Jenkins 运行在 8080 端口，在浏览器中直接访问该端进行服务配置
- Jenkins 的 RPM 仓库配置被加到/etc/yum.repos.d/jenkins.repo

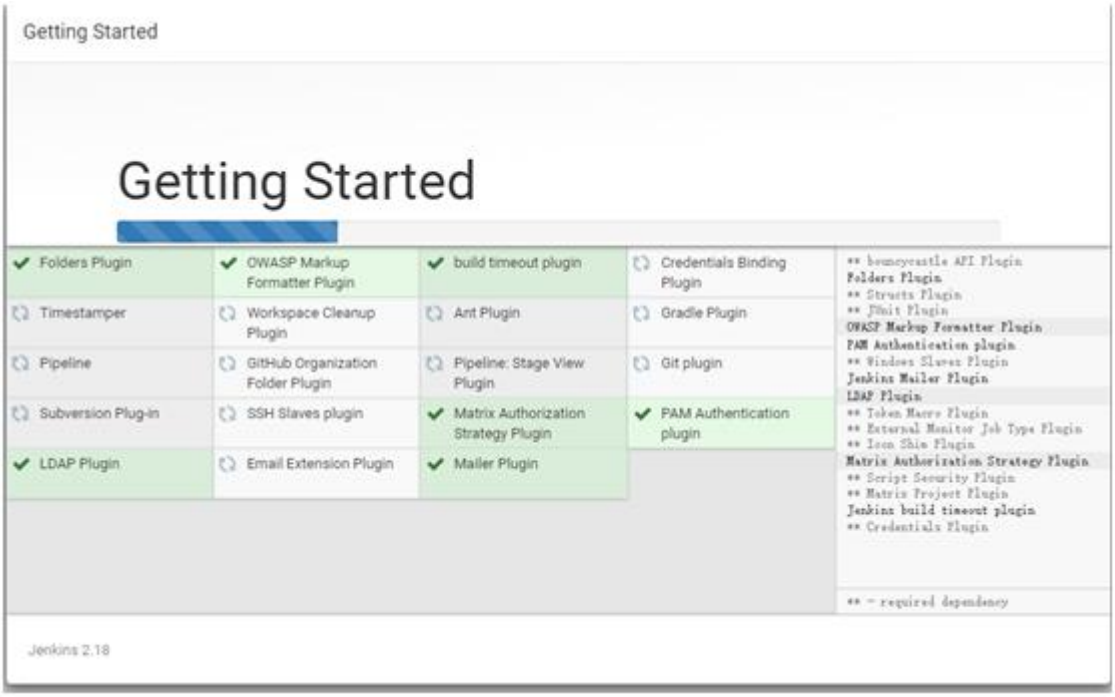
4、打开 jenkins

在浏览器中访问
首次进入会要求输入初始密码如下图，
初始密码在：/var/lib/jenkins/secrets/initialAdminPassword



选择“Install suggested plugins”安装默认的插件，下面 Jenkins 就会自己去下载相关的插件进行安装。


Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.18


Getting Started
Getting Started
Folders Plugin
OWASP Markup Formatter Plugin
build timeout plugin
Credentials Binding Plugin
Timestamper
Workspace Cleanup Plugin
Ant Plugin
Gradle Plugin
Pipeline
GitHub Organization Folder Plugin
Pipeline: Stage View Plugin
Git plugin
Subversion Plug-in
SSH Slaves plugin
Matrix Authorization Strategy Plugin
PAM Authentication plugin
LDAP Plugin
Email Extension Plugin
Mailer Plugin
Jenkins 2.18

创建超级管理员账号

Getting Started

Create First Admin User

用户名:

密码:

确认密码:

全名:

电子邮件地址:

Jenkins 2.18

Continue as admin

Save and Finish

Getting Started

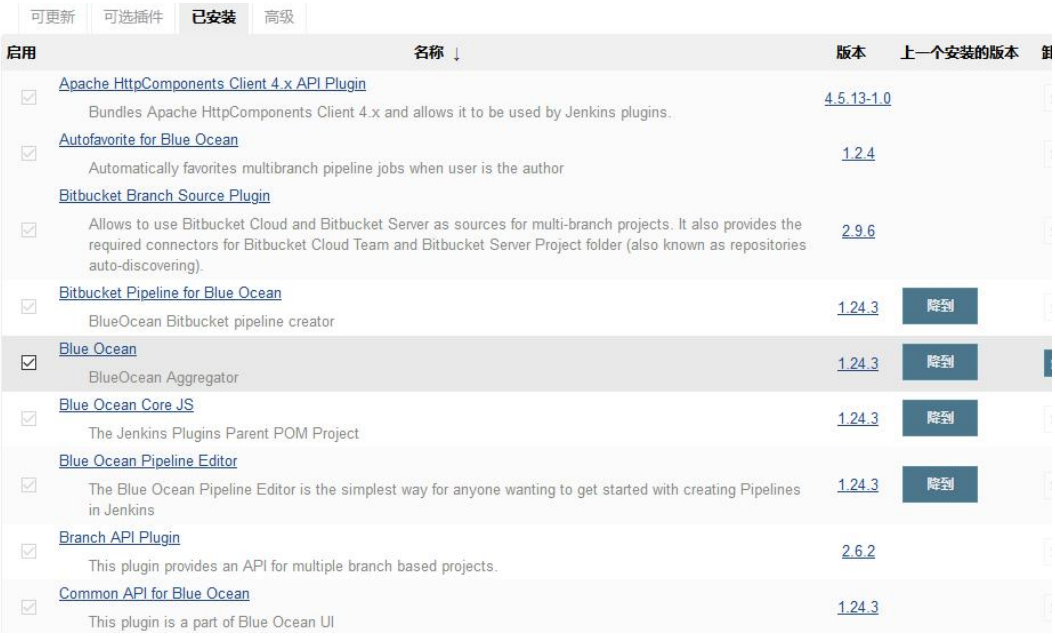
Jenkins is ready!

Your Jenkins setup is complete.

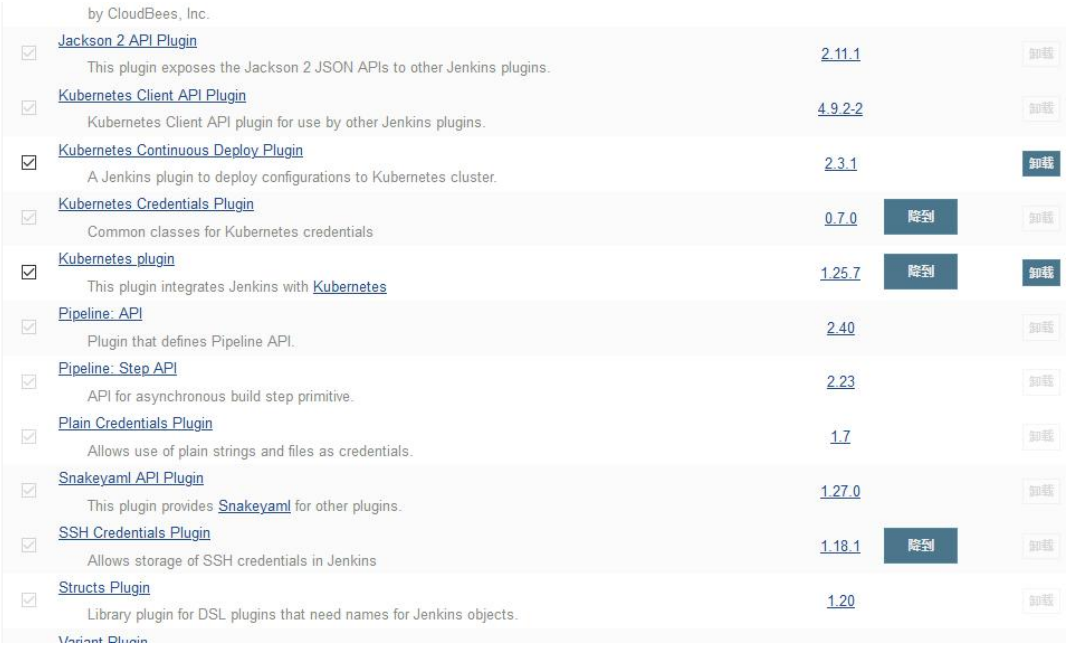
Start using Jenkins

Jenkins 2.18

5、 安装 Blue Ocean 插件



安装 kubernetes



三、docker 安装

OS 环境：CentOS 7

安装依赖
yum install -y yum-utils \
device-mapper-persistent-data \
lvm2

添加镜像地址
yum-config-manager --add-repo \
http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

安装 docker-ce
Yum clean all
Yum makecache
Yum -y install docker-ce

启动服务
Systemctl start docker

查看版本
Docker version

运行镜像 helloWorld

Sudo mkdir -p /etc/docker
Sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://yelruh97.mirror.aliyuncs.com"]
}
EOF

sudo systemctl daemon-reload
sudo systemctl restart docker

Docker pull hello-world
docker images
Docker run hello-world

四、docker 私有仓库搭建

1、安装指令

```
docker pull registry
```

默认拉取最新版：

```
[root@CentOS7 /]# docker pull registry
Using default tag: latest
Trying to pull repository docker.io/library/registry ...
latest: Pulling from docker.io/library/registry
486039affc0a: Pull complete
ba51a3b098e6: Pull complete
8bb4c43d6c8e: Pull complete
6f5f453e5f2d: Pull complete
42bc10b72f42: Pull complete
```

2、配置私有仓库地址

```
vim /etc/docker/daemon.json
```

输入如下参数，注意修改为自己的 ip 地址：

```
"insecure-registries": ["10.211.55.4:5000"]
```

```
{
 "insecure-registries": ["10.211.55.4:5000"]
}
~
```

`:wq` 保存退出，然后我们重启启动一下 docker

```
systemctl restart docker
```

3、创建容器

```
docker run -d -p 5000:5000 --name registry docker.io/registry
```

部分参数说明：

- -d：让容器在后台运行
- -p：指定容器内部使用的网络端口映射到我们使用的主机上
- --name：指定容器创建的名称

4、重新加载配置

```
sudo systemctl daemon-reload
```

然后浏览器访问：**http://10.211.55.4:5000/v2/_catalog**

如果访问不到，尝试关闭防火墙：

```
systemctl stop firewalld
```

如果还是访问不不到，可以重启一下 docker

```
sudo systemctl restart docker
```

然后重新运行一下容器。

5、验证上传镜像到私有仓库

我们使用 HelloWorld 镜像进行测试，首先先拉取一下：

```
docker pull hello-world
```

```
[root@CentOS7 etc]# docker pull hello-world
Using default tag: latest
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f
Status: Downloaded newer image for docker.io/hello-world:latest
[root@CentOS7 etc]#
```

拉取之后我们看一下镜像名称及版本：

```
[root@CentOS7 etc]# docker images
REPOSITORY                                  TAG                 IMAGE ID            CRE
docker.io/gitlab/gitlab-ce                  latest              100b8f44f2af        47
docker.io/jenkins/jenkins                   lts                 5d1103b93f92        8 d
docker.io/registry                          latest              708bc6af7e5e        4 m
docker.io/hello-world                       latest              bf756fb1ae65        5 m
docker.elastic.co/elasticsearch/elasticsearch  6.8.5              4f90d9a6692f        6 m
```

至此我们就有了一个 hello-world 镜像，接下来我们使用 push 指令将镜像推送到刚刚搭建的 registry 中：

```
# 标记 hello-world 该镜像需要推送到私有仓库
docker tag hello-world:latest 127.0.0.1:5000/hello-world:latest
# 通过 push 指令推送到私有仓库
docker push 127.0.0.1:5000/hello-world:latest
```

```
[root@CentOS7 etc]#
[root@CentOS7 etc]# docker tag hello-world:latest 127.0.0.1:5000/hello-world:latest
[root@CentOS7 etc]#
[root@CentOS7 etc]# docker push 127.0.0.1:5000/hello-world:latest
The push refers to a repository [127.0.0.1:5000/hello-world]
9c27e219663c: Pushed
latest: digest: sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042 size: 52
[root@CentOS7 etc]#
```

再来访问：**http://10.211.55.4:5000/v2/_catalog**

我们可以看到私有仓库目录已经有刚刚推送上去的 `hello-world` 镜像了。

6、验证从私有仓库下载镜像

验证完了上传，我们再来测试一下下载镜像：

```
# 格式如下：
docker pull 127.0.0.1:5000/镜像名称:镜像版本号
# 以 hello-world 为例：
docker pull 127.0.0.1:5000/hello-world
```

```
[root@CentOS7 etc]# docker pull 127.0.0.1:5000/hello-world:latest
Trying to pull repository 127.0.0.1:5000/hello-world ...
latest: Pulling from 127.0.0.1:5000/hello-world
Digest: sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042
Status: Image is up to date for 127.0.0.1:5000/hello-world:latest
[root@CentOS7 etc]# docker images
REPOSITORY                                  TAG                 IMAGE ID            CREAT
docker.io/gitlab/gitlab-ce                  latest              100b8f44f2af        47 ho
docker.io/jenkins/jenkins                   lts                 5d1103b93f92        8 day
docker.io/registry                          latest              708bc6af7e5e        4 mon
127.0.0.1:5000/hello-world                  latest              bf756fb1ae65        5 mon
docker.io/hello-world                       latest              bf756fb1ae65        5 mon
```

到这可能有的小伙伴就有疑惑了，你这通过 **ip** 能拉取吗，我看你一直用的 `127.0.0.1`，毕竟私有仓库搭建完是要给别人用的嘛~

那么我们就测试一下：

```
docker  pull  10.211.55.4:5000/hello-world:latest
```

```
[root@CentOS7 etc]# docker pull 10.211.55.4:5000/hello-world:latest
Trying to pull repository 10.211.55.4:5000/hello-world ...
latest: Pulling from 10.211.55.4:5000/hello-world
Digest: sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042
Status: Downloaded newer image for 10.211.55.4:5000/hello-world:latest
[root@CentOS7 etc]# docker images
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
docker.io/gitlab/gitlab-ce                  latest              100b8f44f2af        2 days ago          1.81 GB
docker.io/jenkins/jenkins                   lts                 5d1103b93f92        8 days ago          656 MB
docker.io/registry                          latest              708bc6af7e5e        4 months ago        25.8 MB
10.211.55.4:5000/hello-world                latest              bf756fb1ae65        5 months ago        13.3 kB
127.0.0.1:5000/hello-world                  latest              bf756fb1ae65        5 months ago        13.3 kB
```

如上图所示，通过 **ip** 也是可以拉取成功的，但是在这再额外补充一下，有的小伙伴可能提示如下：

```
Trying to pull repository 10.211.55.4:5000/hello-world ...Get
https://10.211.55.4:5000/v1/_ping: http: server gave HTTP response to HTTPS client
```

这种显然是拉取失败了，提示大致就是，尝试从 https 上拉取，但是返回的是 http 响应，如何解决呢？

我们可以通过如下进行处理一下，xxx 修改为自己的 **ip** 地址：

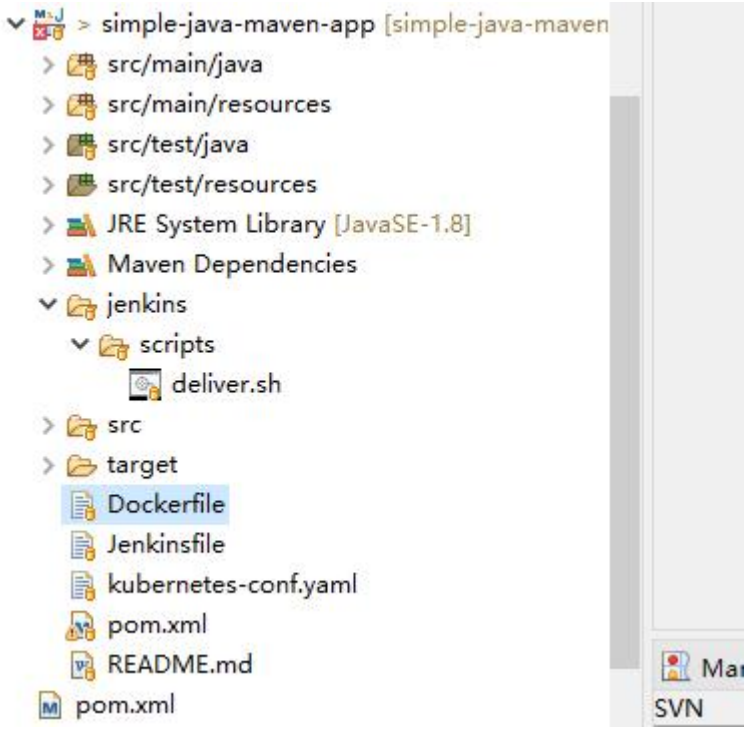
```
echo '{ "insecure-registries":["xxx.xxx.xxx.xxx:5000"] }' > /etc/docker/daemon.json
```

其实如果你是按上我上边这些步骤下来是不会有问题的，在**步骤 2** 中**配置私有仓库地址**时，其实我们已经配置了 `insecure-registries`，但是配置后我们需要重启启动一下 docker 才可以生效，毕竟是配置了系统文件嘛~

ok，至此整个搭建过程完毕~

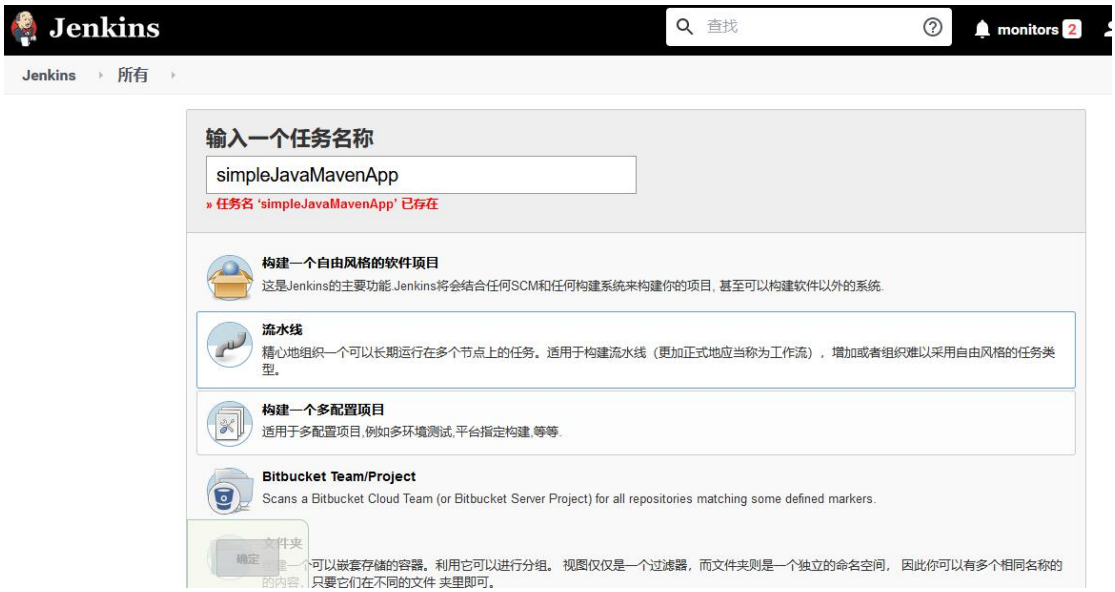
五、Springboot 应用自动构建发布示例

1、Springboot maven 项目文件目录结构

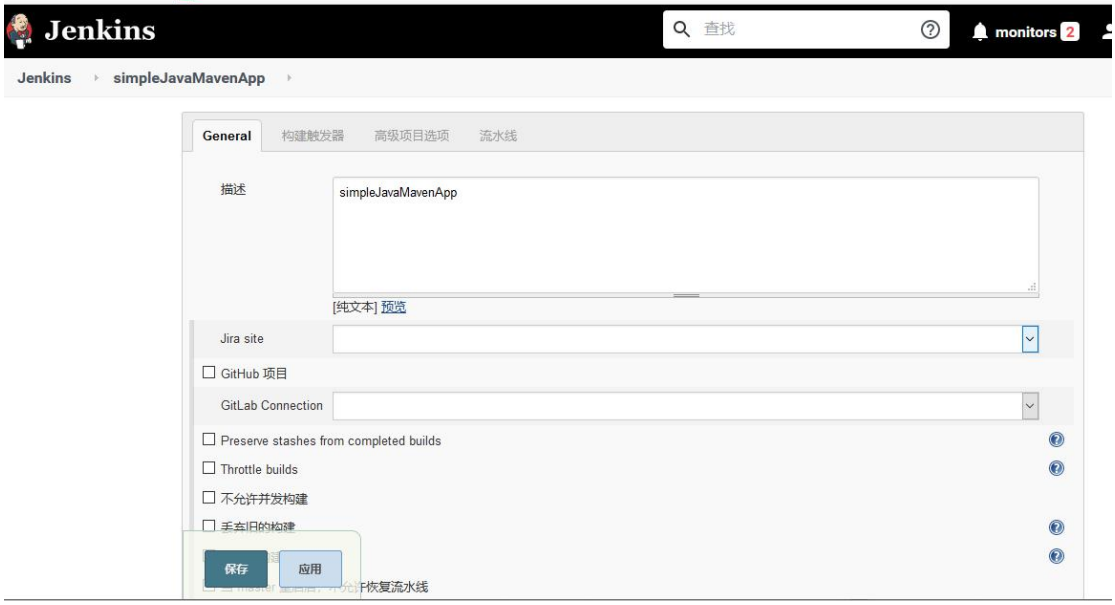


有几个重要文件需要说明：
Dockerfile:用于构建 docker 镜像的脚本
Jenkinsfile:用于 jenkins 流水线项目的自动构建脚本
Kubernetes-conf.yaml:用于 k8s 集群自动管理发布的脚本
Jenkins/scripts/deliver.sh: 完成 springboot jar 生成 docker 镜像，镜像上传到私有仓库，k8s 任务调度等

2、在 jenkins 中新建一个流水线项目，点击确定

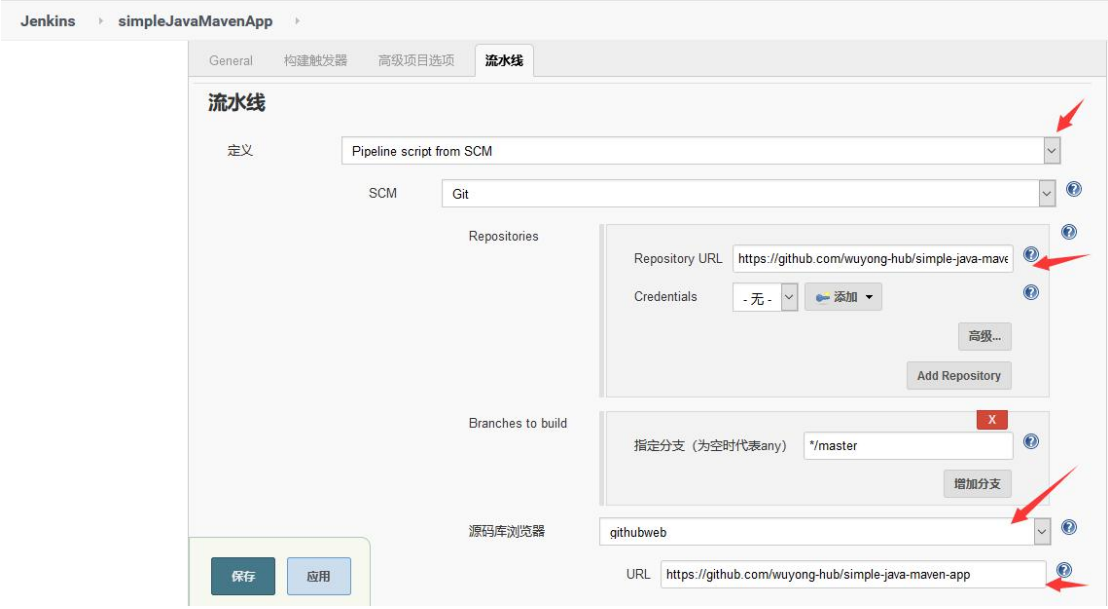


配置项目：

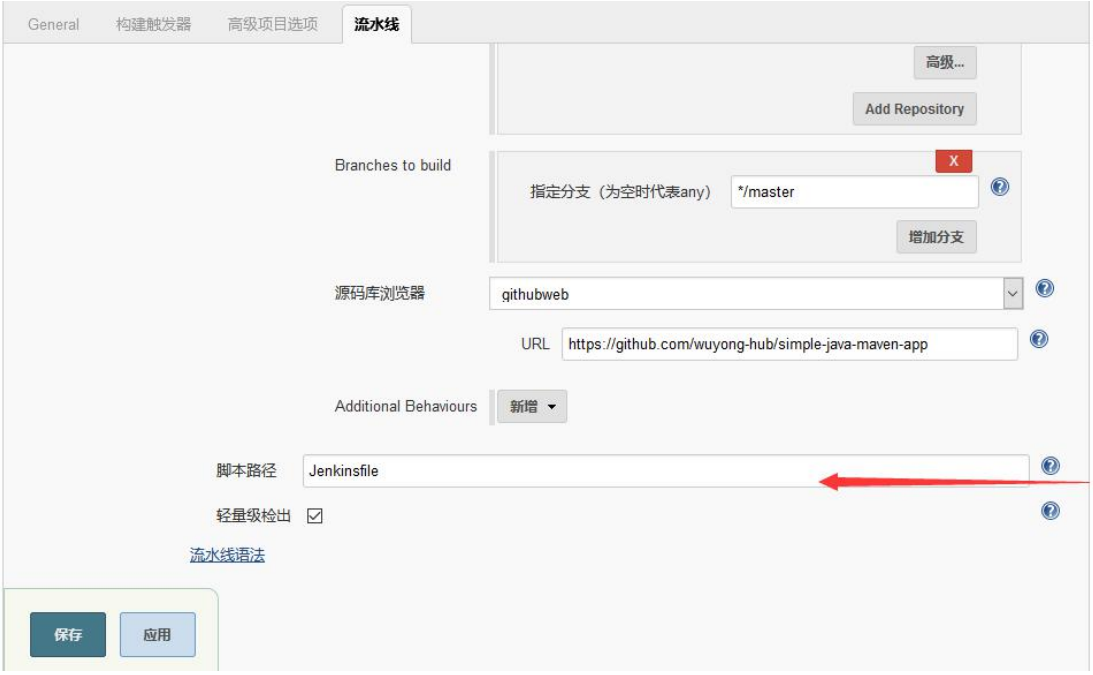


选择 github hook 选项，github 有新代码提交时触发自动构建

配置 github 仓库地址

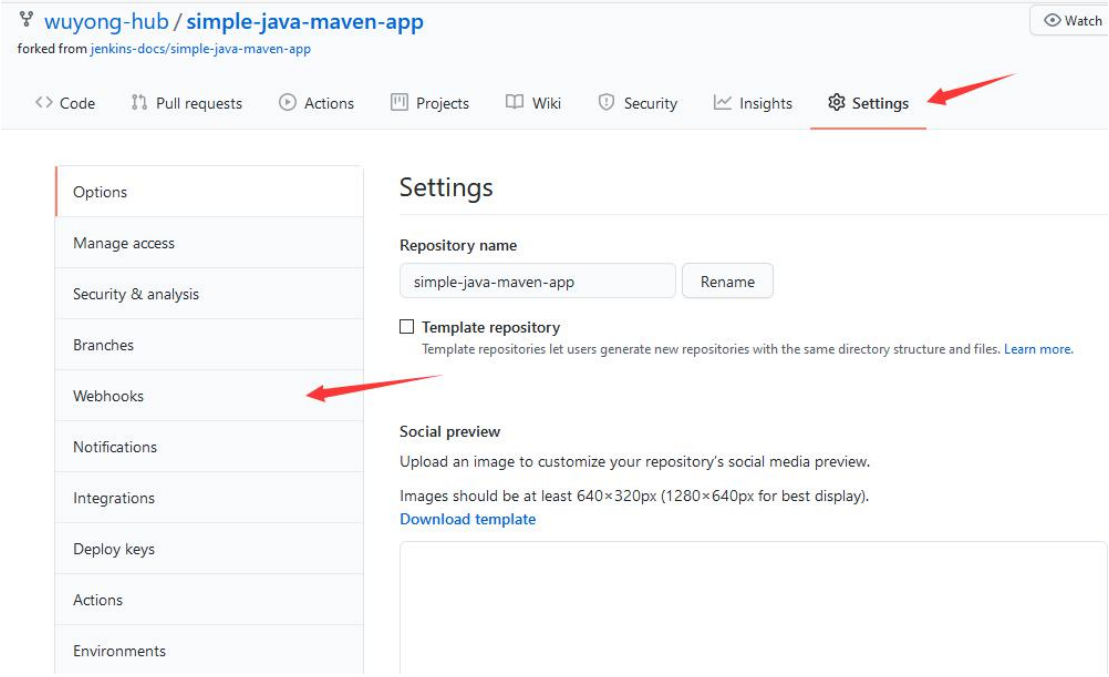


指定流水线脚本

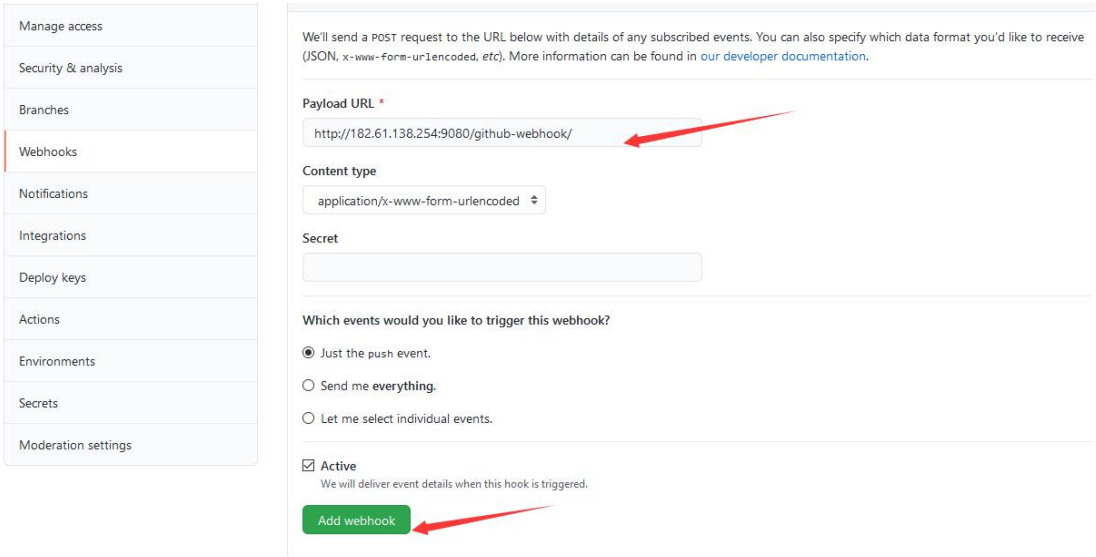


点击【保存】完成配置。

3、在 github 中配置 webhook



输入 jenkins 的 webhook 地址，点击新增完成配置



4、主要脚本解释

Jenkinsfile 内容：

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh '/usr/local/apache-maven-3.6.3/bin/mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh '/usr/local/apache-maven-3.6.3/bin/mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
         stage('Deploy') {
            steps {
                echo "Deploy stage: ..."
            }
        }
        stage('Deliver') {
            steps {
                sh './jenkins/scripts/deliver.sh'
            }
        }
    }
}
```

将应用打包成jar

通过脚本执行更新发布

Deliver.sh 内容：

```
#!/usr/bin/env bash
echo 'The following complex command extracts the value of the <name/> element'
echo 'within <project/> of your Java/Maven project''s "pom.xml" file.'
set -x
NAME=`/usr/local/apache-maven-3.6.3/bin/mvn help:evaluate -Dexpression=project.name | grep "^[^\[]"`
set +x
```

```
echo 'The following complex command behaves similarly to the previous one but'
echo 'extracts the value of the <version/> element within <project/> instead.'
set -x
VERSION=`/usr/local/apache-maven-3.6.3/bin/mvn  help:evaluate  -Dexpression=project.version  |
grep "^[^\[]"`
set +x

#self-repo addr
DOCKER_REPO=182.61.138.254:5000

echo 'remove old image.'
set -x
CONTAINERID=`docker ps -a | grep ${NAME} | awk '{print $1}'`
if test -n "$CONTAINERID"
then
    docker stop $CONTAINERID
    docker container rm $CONTAINERID
    IMAGEID=`docker images | grep ${NAME} | awk '{print $3}'`
    docker image rm $IMAGEID
fi
set +x

echo 'build docker image and push to repository.'
rm -rf docker-build/
mkdir docker-build
cp target/${NAME}-${VERSION}.jar docker-build/app.jar
cp Dockerfile docker-build/
cd docker-build
set -x
DOCKER_NAME="$DOCKER_REPO/${NAME}:latest"
docker build -t $DOCKER_NAME .
docker push $DOCKER_NAME
set +x

#Kubernetes run
echo 'run k8s.'
set -x
cd ../
kubectl apply -f kubernetes-conf.yaml
set +x

echo 'END.'
```

以上脚本完成几个任务：

1、查找应用名称

2、查找版本号

3、根据应用名查找已经存在的 docker 镜像，若存在，则删除旧的镜像

4、构建新的 docker 镜像，并推送到私有镜像仓库

5、运行 k8s 自动安装脚本

Dockerfile 内容：

```
Dockerfile
FROM openjdk:8-jdk-alpine
MAINTAINER Wuyong
VOLUME /tmp
ADD app.jar app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

将可执行jar加入容器

docker运行入口

kubernetes-conf.yaml 内容：

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  labels:
    app: simple-java-maven-app
  name: simple-java-maven-app
  namespace: kube-system   #命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      app: simple-java-maven-app
  template:
    metadata:
```

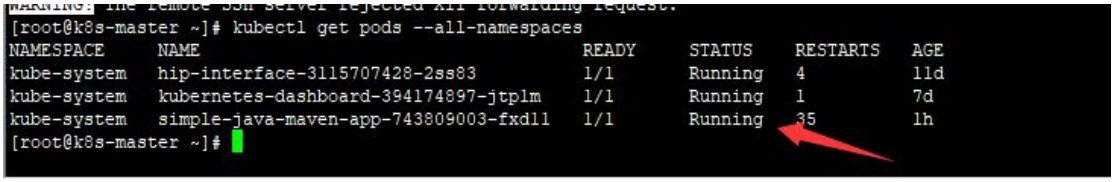
```
      labels:
        app: simple-java-maven-app
      # Comment the following annotation if
Dashboard must not be deployed on master
      annotations:

scheduler.alpha.kubernetes.io/tolerations: |
          [
            {
              "key": "dedicated",
              "operator": "Equal",
              "value": "master",
              "effect": "NoSchedule"
            }
          ]
    spec:
      containers:
      - name: simple-java-maven-app
        image:
182.61.138.254:5000/simple-java-maven-app:lates
t      #默认的镜像是使用 google 的，这里私有仓库镜像
imagePullPolicy: IfNotPresent
```

```
        ports:
        - containerPort: 9900
          protocol: TCP
        args:
          # Uncomment the following line to
manually specify Kubernetes API server Host
          # If not specified, Dashboard will
attempt to auto discover the API server and connect
          # to it. Uncomment only if the default
does not work.
          -
--apiserver-host=http://182.61.138.254:8080
#注意这里是 api 的地址
        livenessProbe:
          httpGet:
            path: /
            port: 9900
          initialDelaySeconds: 30
          timeoutSeconds: 30
---
kind: Service
apiVersion: v1
```

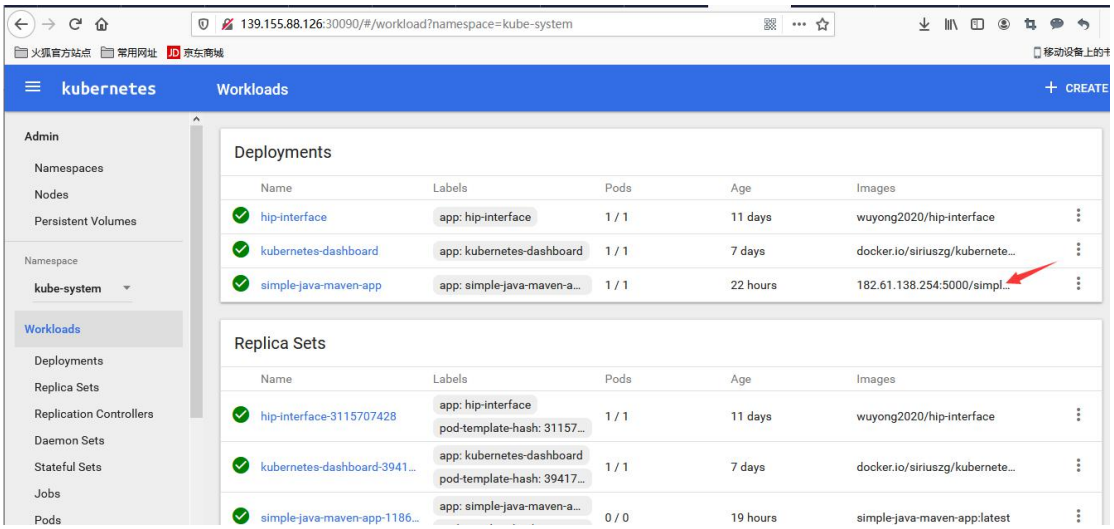
```
metadata:
  labels:
    app: simple-java-maven-app
  name: simple-java-maven-app
  namespace: kube-system
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 9900
    nodePort: 30099  #docker 容器对外接口
  selector:
    app: simple-java-maven-app
```

5、更新程序代码，通过 git 提交后，打开 jenkins blue Ocean，自动构建过程如下：

构建完成后,通过 k8s 主机查看服务，状态为 running 表示服务已经部署运行



通过 kubernetes-dashboard 查看服务：



浏览器访问：

至此，我们只需要在电脑上编写代码，提交到 github 后，以下流程都自动完成：

github 检测到代码更新==>通过 webhook 通知 jenkins 更新事件==>jenkins 下载最新代码==>找到流水线脚本并执行==>编译打包==>测试==>构建 docker 镜像==>上传到仓库==>执行 k8s 脚本自动运行 docker 镜像。