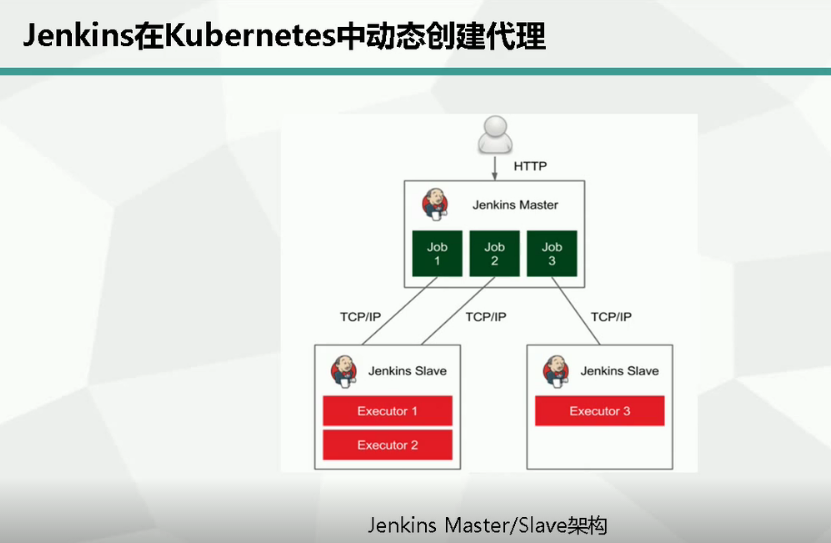
Master-slave架构如图：



需要解决Master单点瓶颈问题

Master的slave架构部署

我们只需要动态的去创建slave的pod···部署完成会自动删除这个pod··

Kubernetes插件：Jenkins在Kubernetes集群中运行动态代理

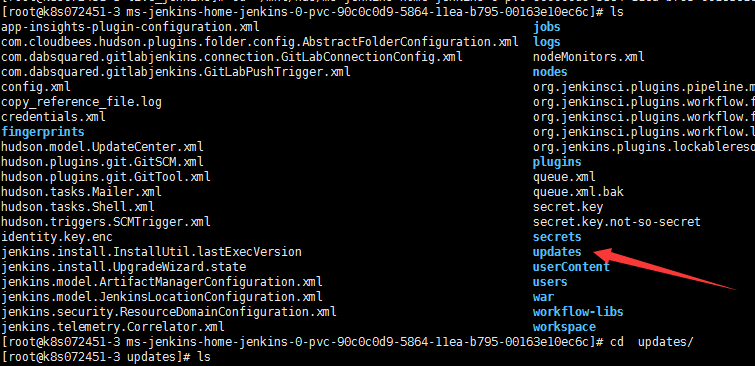
#部署slave需要用到的插件文档如下

插件介绍：<https://github.com/jenkinsci/kubernetes-plugin>

#jenkins安装插件默认是谷歌的地址 相当的慢···怎么解决这个问题呢

需要直接去到jenkins的配置文件 修改一个updates下面的文件

vim default.json

、

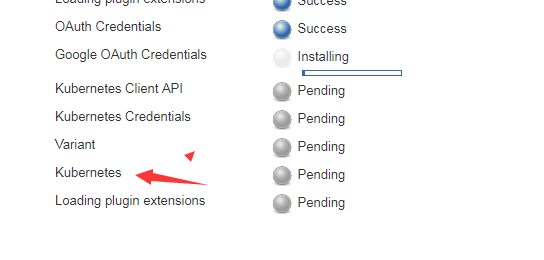
主要修改两个地方 一个是 谷歌修改成百度 还一个是 下载的插件地址修改成清华源



#这个下载jenkins的插件源我还没测试·····但是之前确实是下载插件非常慢的····

1.我们需要在jnekins上安装两个插件

1. ）git插件
2. ）kubernetes插件



Kubernetes Client API

Kubernetes Credentials

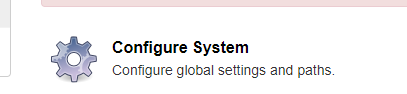
Kubernetes

#这几个插件需要下载多次，我把插件备份到了 百度云去了

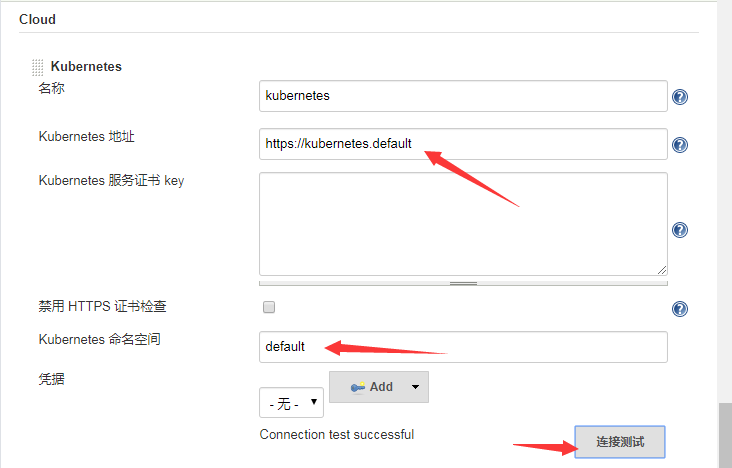
Git插件我已经安装过了··这里就没显示····kubernetes插件还很多个依赖

家里网络慢的话安装这几个插件需要蛮久的（我在家安装了5个小时，需要不停的安装，因为可能一直超时）

2.配置kubernetes插件

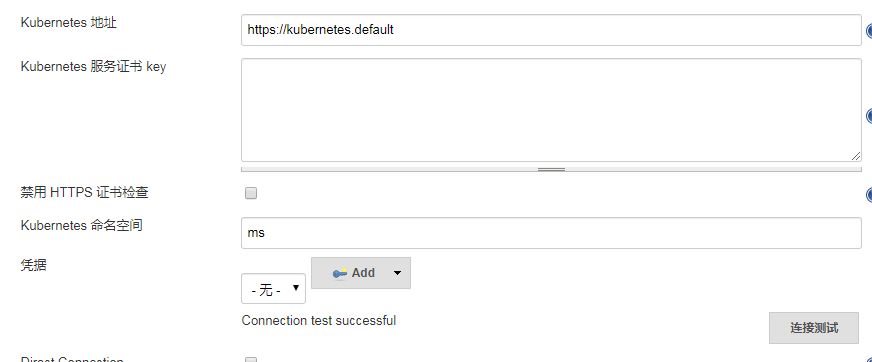


系统配置拉取到最下面cloud—kubernetes 如下图：



#上图我名称使用的默认的，kubernetes的地址需要配置下 注意是https，名称空间配置下default，如果在kubernetes外边 部署jenkins，kubernetes服务证书key需要填写ca的··

#如果你jenkins部署到了 其他名称空间 这边你也要填写其他的 这个名称空间 指得是你K8S部署jenkins的名称空间 我这边第二次部署到了ms空间



kubectl get svc 可以查看到kubernetes的默认地址



他的dns是提供的https的访问···所以地址应该是<https://kubernetes.default>

前面是dns地址后面default是默认名称空间··前提需要安装dns服务··我这边都默认安装的

然后就能连接测试了···能通过证明能连接了

然后还需要填写下jenkins地址···

可以kubectl get svc 查看到jenkins地址





因为jenkins有两个地址，所以我这边只填写的http

注意下：如果jenkins不是在集群外下面kubernetes的地址需要填写kube-apiserver对外提供服务的地址··一般是前面是负载均衡地址和6443端口



当然外部部署jenkins 这个jenkins地址也需要填写外部的 jenkins地址了··还需要确保jenkins50000端口是监听的···这个端口是连接slave的



还两块配置 也不需要了 一块是pod模板配置和容器配置·····我们将来都在jenkins的pipeline去配置

然后保存，jenkins的配置就OK了···

辅助资料：

Jenkins配置pod和容器的pipeline配置可以在下面地址找到相关资料

<https://github.com/jenkinsci/kubernetes-plugin>

3.测试pipeline脚本来动态的创建slave的pod

#我们现在来测试下 pipeline的配置

我们需要配置一个启动jenkins的slave的pod镜像

官方的提供的slave镜像不太好···我们需要自己弄一个镜像

镜像的作用需要完成以下的任务·····我们dockerfile也需要考虑下面的步骤



cat Dockerfile

FROM centos:7

LABEL maintainer lizhenliang

RUN yum install -y java-1.8.0-openjdk maven curl git libtool-ltdl-devel && \

yum clean all && \

rm -rf /var/cache/yum/\* && \

mkdir -p /usr/share/jenkins

COPY slave.jar /usr/share/jenkins/slave.jar

COPY jenkins-slave /usr/bin/jenkins-slave

COPY settings.xml /etc/maven/settings.xml

RUN chmod +x /usr/bin/jenkins-slave

ENTRYPOINT ["jenkins-slave"]

#注意 pod里面也需要docker 我们这边是挂载了宿主机的···

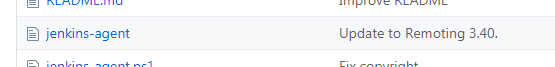
Master怎么和slave通信呢··需要官方的slave.jar包放到镜像中.

我们这边有官方的slave.jar包了 但是也可以直接在你master地址那下载slave.jar:

<http://jenkins.gold-malls.com/jnlpJars/slave>.jar

#我们的dockerfile里面的jenkins-salve哪里来呢，也有个地址

<https://github.com/jenkinsci/docker-jnlp-slave>

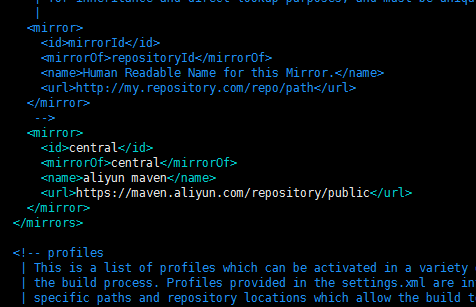


#官方的名字是jenkins-agent 我们这边的脚本是jenkins-slave 但是内容都一样

这个文件就是一个shell脚本

#还一个maven配置文件需要修改，替换默认的maven配置

settings.xml 主要增加了一个阿里云的maven仓库地址··如下图



我要把这个镜像推送到阿里云上去·····阿里云镜像地址：

docker pull registry.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:[镜像版本号]

我们需要把镜像推送到上面镜像地址

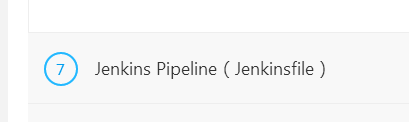
docker build -t registry.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_jdk1.8 .

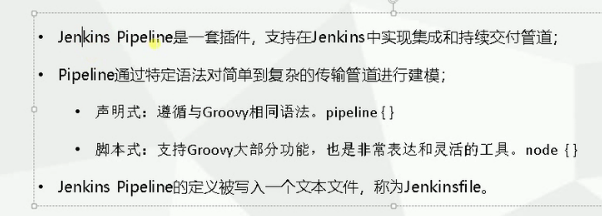
docker push registry.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_jdk1.8

#我这边是使用的阿里云的镜像仓库 我K8S又是用的阿里云的···所以我这里没有配置信任仓库，默认docker只会https的方式访问镜像仓库

第二次部署 我也是直接使用这个jenkins-slave就行了

始下一章节





怎么写pipeline呢 可以去jenkins官方找到 相关的语法实例

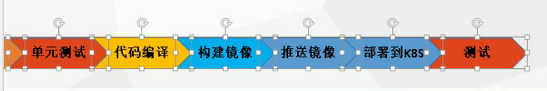
<https://jenkins.io/doc/book/pipeline/syntax>

#创建一个简单的demo感受下pipeline

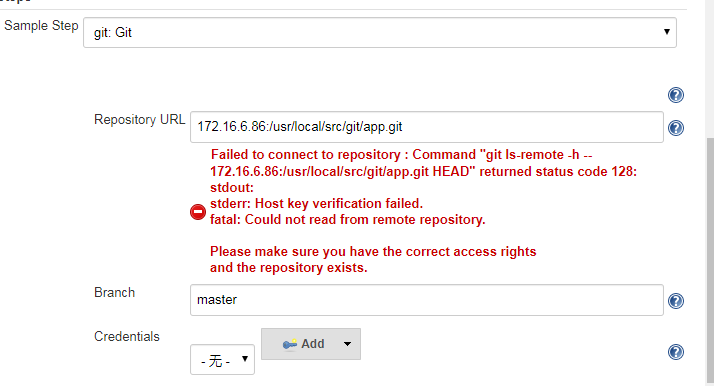


#简单的我之前有个pipeline的学习，我就跳过了

Pipeline主要是需要完成下面的步骤：我们需要琢磨怎么在脚本中完成··



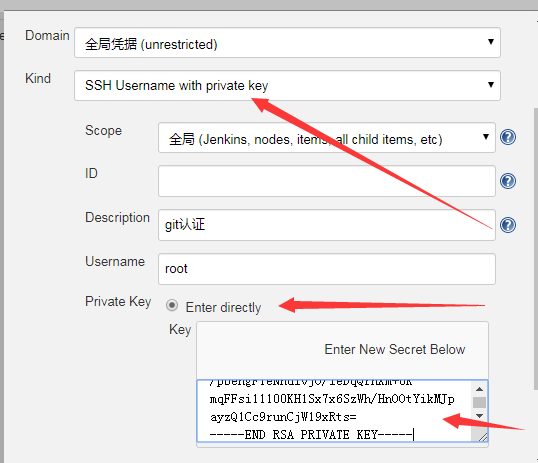
如果pipeline语法不会写 就去jenkins后台生成pipeline语法··



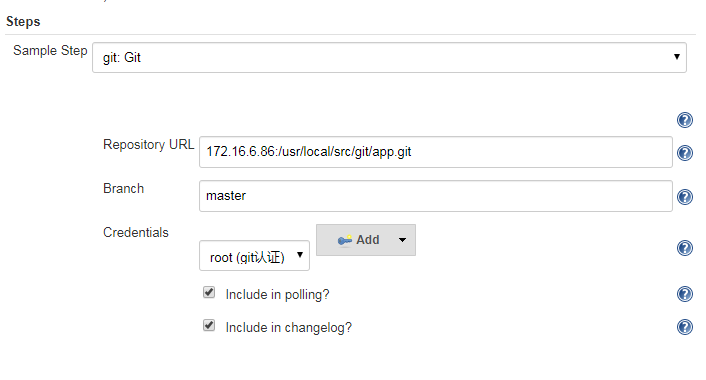
比如生成我的git地址···

172.16.6.86:/usr/local/src/git/app.git

因为jenkins部署在K8S里面···所以我们需要配置一个凭证 ，能动态的访问git，我们可以把git的私钥地址拷贝过来，我这边访问git是通过root用户和免秘钥访问



#这样就没有红色的字了，就能访问到git仓库了···

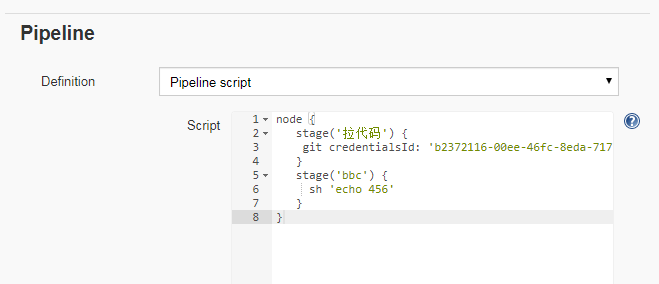


然后我就能生成这个pipeline脚本了

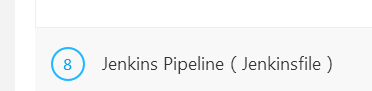
git credentialsId: 'b2372116-00ee-46fc-8eda-71753c97a082', url: '172.16.6.86:/usr/local/src/git/app.git '

#配置git凭据就OK了

#然后把这个拉取代码的pipeline脚本 放到 jenkins的pipeline脚本



#下一章节



本章节35

#根据上面插件的demo我们 自己写一个简单的 使用jenkins-slave部署项目

可以参考官方<https://github.com/jenkinsci/kubernetes-plugin>只是参考，我们下面的部署也是参考了文档 。实际部署中还是和文档有些不同，实际是文档拼凑

才能部署OK

#pipeline脚本式部署测试这个jenkins-slave拉取代码

pipeline变量格式如下

// 公共

def registry = "registry-vpc.cn-hangzhou.aliyuncs.com"

// 项目

def project = "goldplusgold"

def app\_name = "demo"

def image\_name = "${registry}/${project}/k8s:${app\_name}\_{BUILD\_NUMBER}"

def git\_address = "172.16.6.86:/usr/local/src/git/app.git"

// 认证

def secret\_name = "registry-pull-secret"

def docker\_registry\_auth = "275cb37e-70c8-4d87-9c66-60b482a87777"

def git\_auth = "6d663f61-dd33-4771-b33c-285caf09ac64"

def k8s\_auth = "854ae16f-dfa2-46ee-a214-a4c4aed0768e"

podTemplate(label: 'jenkins-slave', cloud: 'kubernetes', containers: [

containerTemplate(

name: 'jnlp',

image: "${registry}/${project}/k8s:jenkins\_slave\_jdk1.8"

),

],

volumes: [

hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/var/run/docker.sock'),

hostPathVolume(mountPath: '/usr/bin/docker', hostPath: '/usr/bin/docker')

],

)

{

node("jenkins-slave") {

stage('拉取代码'){

git credentialsId: '6d663f61-dd33-4771-b33c-285caf09ac64', url: '172.16.6.86:/usr/local/src/git/app.git'

}

}

}

#一定要注意这个位置的两个大括号····上面的变量一定要包裹这个node下面的内容 要不然 他jenkins-salve无法工作··

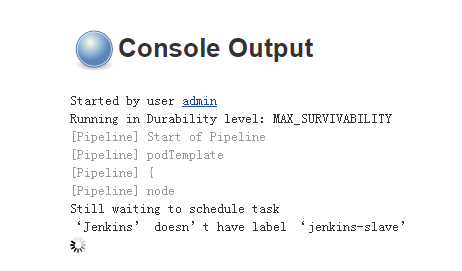
还一个问题，我这边jenkins-slave 没有添加 拉取docker仓库镜像的镜像 也能拉取到这个jenkins-slave镜像···我是百思不得其解

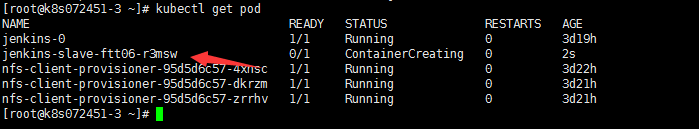
（image: "${registry}/${project}/k8s:jenkins\_slave\_jdk1.8"）就是这个镜像 为什么能拉取到····见鬼了

见鬼见鬼见鬼了 不明白的地方说三次，这里我做了2次试验都是在这里出了点错误···

#注释name: 'jnlp', 这个名字是写死的 不能修改···

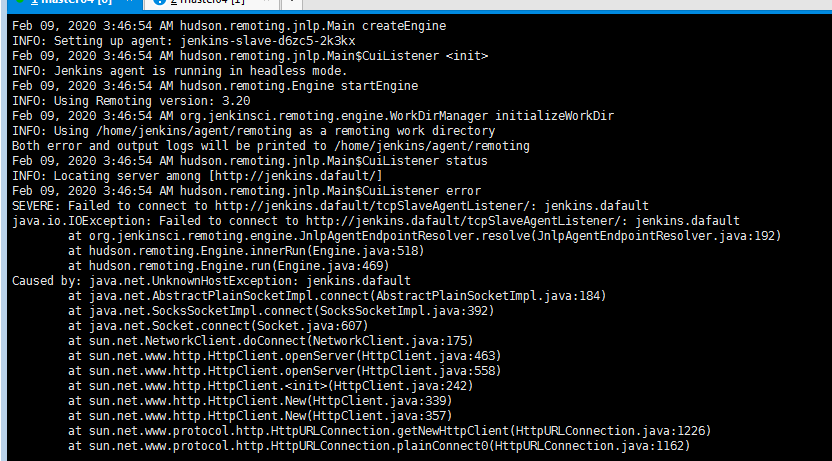
#下面是部署的时候截图 这里有的时候 一直卡主 就是上面的原因 我的变量 pod模板 没有包裹到这个node

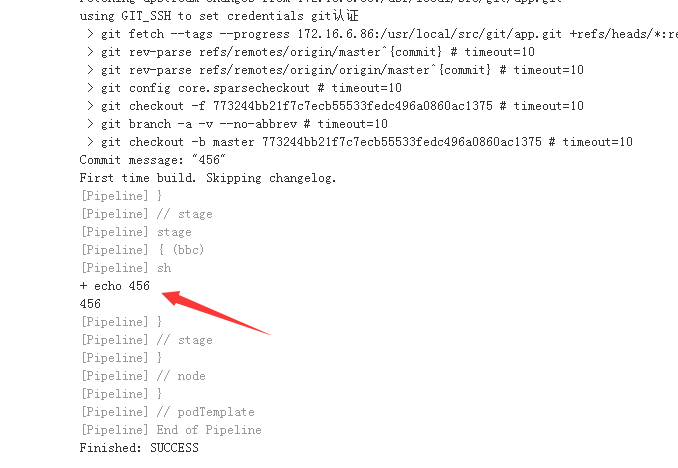


#上面一直在拉取镜像，这个页面是OK的 需要等等··并且还会在k8s集群创建一个slave的pod，就证明这个k8S部署jenkins-slave正在正常工作  


#使用jenkins-slave一直报错····，是因为我全局配置的 jenkins名称空间写错了··

[root@k8s072451-3 nfs-client]# kubectl logs jenkins-slave-d6zc5-2k3kx





#直接部署就已经 OK了··我项目确实就修改了一个456的文件···这是一个简单的jenkins-slave拉取项目·就已经成功

#第2次 就测试 jenkins-slave拉取 一个代码 成功····

Pipeline声明式脚本

// 公共

def registry = "registry-vpc.cn-hangzhou.aliyuncs.com"

// 项目

def project = "goldplusgold"

def app\_name = "demo"

// def image\_name = "${registry}/${project}/k8s:${app\_name}\_{BUILD\_NUMBER}"

def git\_address = "172.16.6.86:/usr/local/src/jenkins"

// 认证

def secret\_name = "registry-pull-secret"

def docker\_registry\_auth = "275cb37e-70c8-4d87-9c66-60b482a87777"

def git\_auth = "6d663f61-dd33-4771-b33c-285caf09ac64"

def k8s\_auth = "854ae16f-dfa2-46ee-a214-a4c4aed0768e"

pipeline {

agent {

kubernetes {

label "jenkins-slave"

yaml """

apiVersion: v1

kind: Pod

metadata:

name: jenkins-slave

namespace: ms

spec:

containers:

- name: jnlp

image: registry.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_jdk1.8

imagePullPolicy: Always

volumeMounts:

- name: docker-cmd

mountPath: /usr/bin/docker

- name: docker-sock

mountPath: /var/run/docker

volumes:

- name: docker-cmd

hoatPath:

path: /usr/bin/docker

- name: docker-sock

hostPath:

path: /var/run/docker

"""

}

}

parameters {

gitParameter branch: '', branchFilter: '.\*', defaultValue: 'master', description: '请选择你要发布的分支', name: 'Branch', quickFilterEnabled: false, selectedValue: 'NONE', sortMode: 'NONE', tagFilter: '\*', type: 'PT\_BRANCH\_TAG'

}

stages {

stage('拉取代码') {

steps{

checkout([$class: 'GitSCM',

branches: [[name: "${Branch}"]],

doGenerateSubmoduleConfiguratsons: false,

extensions: [], submoduleCfg: [],

userRemoteConfigs: [[credentialsId: '6d663f61-dd33-4771-b33c-285caf09ac64',

url: '172.16.6.86:/usr/local/src/jenkins']]])

}

}

stage('代码编译'){

steps {

sh """ mvn clean package -Dmaven.test.skip=true """

}

}

}

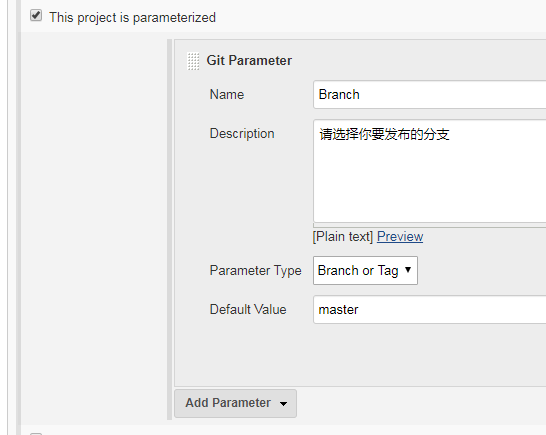
}

#注释：

记录声明式脚本容易出错的地方

1. 部署了很多次才成功，主要错误在变量名字 不需要写成什么${params.Branch} 直接写成$Branch就行了·可能和版本也有关系
2. 是这种方式部署jenkins-slave还可以写parameters这些参数 如果是node那种脚本式语法 我不知道parameters参数加到哪里

其实也可以用系统自带的图形格式的····但是这种写到脚本里的 看上去清晰···当然 默认会添加图形化格式的，下图是系统自动添加的



3）最上面的pipeline格式 是有双括号的 一个在上面 一个是最下面 包裹这些个内容，别少些了大括号

pipeline {

……………………

}

4）重要的事情说三遍 注意格式 格式 格式····还是这个jenkink-slave的格式 之前 我加了一个名称空间 就怎么都不行了 最后莫名其妙好了···

还有好几个格式···莫名其妙好了····碰到问题 最好重启下 jenkins····可能有好效果

Java项目 demo，构建一个镜像推送到镜像仓库

部署有两种方式 一种是脚本式一种是声明式，我们等下都尝试一下

1.Pipeline脚本式方式部署

#下面我们来一个java的demo项目。包裹拉取项目，打包，部署 几个步骤

#我先把这个java的demo项目推送到我的项目仓库先



#这个项目定义了一个jenkinsfile 就是pipeleine的脚本，以下是内容：

// 公共

def registry = "registry-vpc.cn-hangzhou.aliyuncs.com"

// 项目

def project = "goldplusgold"

def app\_name = "demo"

def image\_name = "${registry}/${project}/k8s:${app\_name}\_{BUILD\_NUMBER}"

def git\_address = "172.16.6.86:/usr/local/src/git/app.git"

// 认证

def secret\_name = "registry-pull-secret"

def docker\_registry\_auth = "881f76f7-ea94-4397-8006-23c300feae9a"

def git\_auth = "b2372116-00ee-46fc-8eda-71753c97a082"

def k8s\_auth = "854ae16f-dfa2-46ee-a214-a4c4aed0768e"

podTemplate(label: 'jenkins-slave', cloud: 'kubernetes', containers: [

containerTemplate(

name: 'jnlp',

image: "${registry}/${project}/k8s:jenkins\_slave\_jdk1.8"

),

],

volumes: [

hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/var/run/docker.sock'),

hostPathVolume(mountPath: '/usr/bin/docker', hostPath: '/usr/bin/docker')

],

)

{

}

node("jenkins-slave"){

// 第一步

stage('拉取代码'){

checkout([$class: 'GitSCM', branches: [[name: “${Branch}”]], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: “${git\_auth}”, url: “${git\_address}”]]])

}

// 第二步

stage('代码编译'){

sh "mvn clean package -Dmaven.test.skip=true"

}

// 第三步

stage('构建镜像'){

withCredentials([usernamePassword(credentialsId: "${docker\_registry\_auth}", passwordVariable: 'password', usernameVariable: 'username')]) {

sh """

echo '

FROM registry-vpc.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:tomcat.8.5

RUN rm -rf /usr/local/tomcat/webapps/\*

ADD target/\*.war /usr/local/tomcat/webapps/ROOT.war

' > Dockerfile

docker build -t ${image\_name} .

docker login -u ${username} -p '${password}' ${registry}

docker push ${image\_name}

"""

}

}

// 第四步

stage('部署到K8S平台'){

sh """

sed -i 's#\$IMAGE\_NAME#${image\_name}#' deploy.yml

sed -i 's#\$SECRET\_NAME#${secret\_name}#' deploy.yml

"""

kubernetesDeploy configs: 'deploy.yml', kubeconfigId: "${k8s\_auth}"

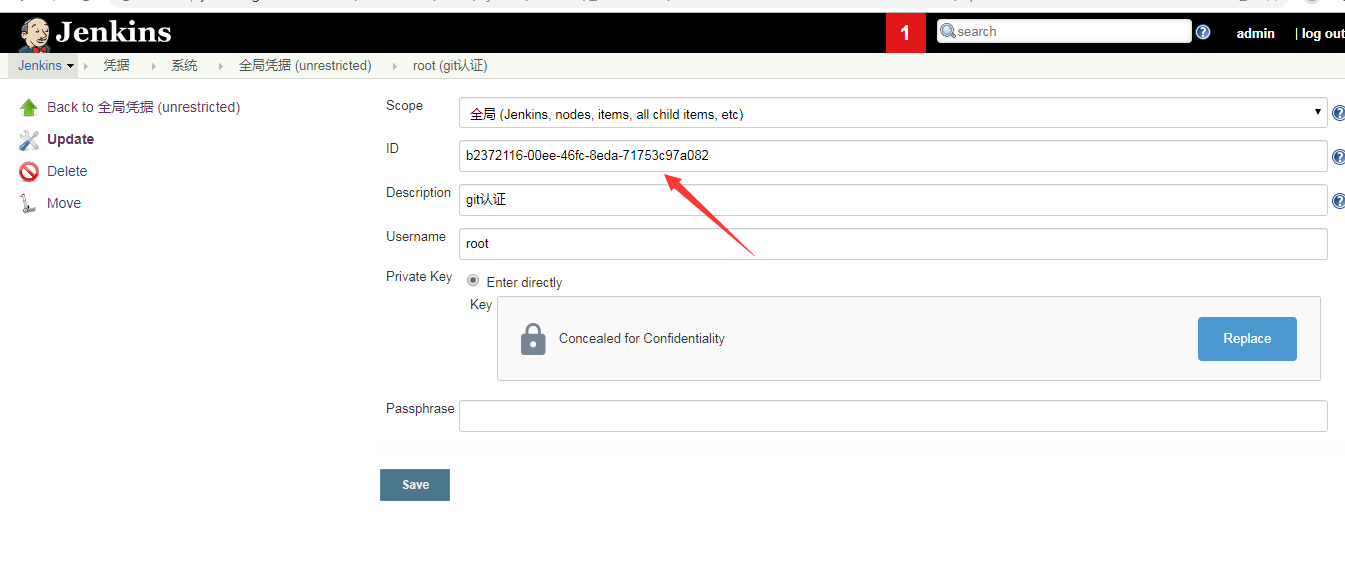
}

}

}

注意：上面脚本式部署定义了很多变量，这是为了 之后换了项目 只需要修改变量内容，脚本基本上都不需要怎么改变

1.下面我主要说明下凭据的id，其实就是我配置的用户名和密码他会自动生成一个id··这个就是凭据的id，下图是我拉取远程git项目的id·

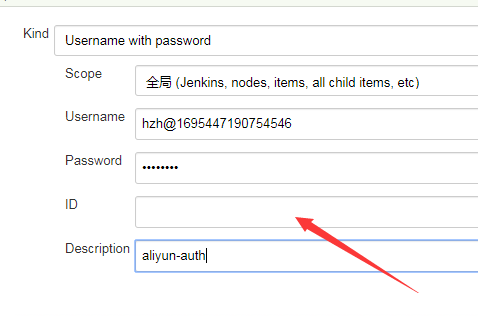


注意，我们之前拉取代码用的git这个pipeline格式 ，但是大部分人都会使用checkout的pipeline语法，所以我们这边拉取镜像也用到checkout的方式

2.认证docker怎么设置凭据

点击凭据---系统---添加凭据

我们这边设置一个拉取阿里云镜像的 用户名和密码 点击确定之后 他会自动生成一个ID我这边生成的是275cb37e-70c8-4d87-9c66-60b482a87777



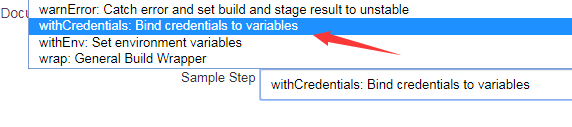
#然后我们需要去pipeline脚本修改这个ID为 275cb37e-70c8-4d87-9c66-60b482a87777

脚本中生成一下拉取docker镜像仓库凭据的用户名和密码的隐藏模式

这样我们可以直接引用比如$password $username

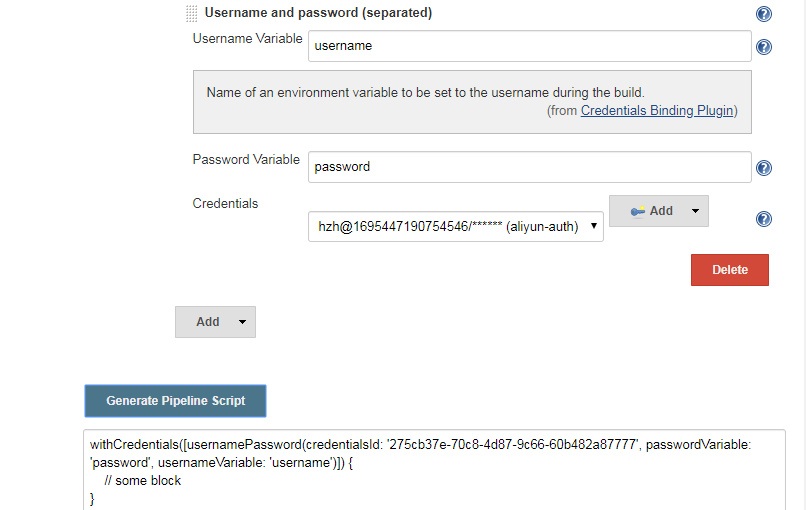
withCredentials([usernamePassword(credentialsId: "${docker\_registry\_auth}", passwordVariable: 'password', usernameVariable: 'username')])

1）选择这个把凭证绑定到变量



2）然后选择

设置一个用户名和密码，设置之后他会默认把你的阿里云镜像的用户名和密码依次保存到这两个变量中，到时候可以引用这两个变量，之后会docker login -u ${username} -p '${password}' 引用变量



#截图生成的pipeline脚本格式

withCredentials([usernamePassword(credentialsId: '0bd9a0a6-2270-474f-a432-ef18cb214b5f', passwordVariable: 'username', usernameVariable: 'password')]) {

// some block

}

1. 上面脚本的pod模板这一块解析一下
2. name一定要是jnlp，这是jenkins官方需要设置的
3. jenkins-slave镜像我们是没配置docker的，不要紧，直接用宿主机的挂载

#正式开始部署java-demo项目

// 公共

def registry = "registry-vpc.cn-hangzhou.aliyuncs.com"

// 项目

def project = "goldplusgold"

def app\_name = "demo"

def image\_name = "${registry}/${project}/k8s:${app\_name}\_${BUILD\_NUMBER}"

def git\_address = "172.16.6.86:/usr/local/src/git/app.git"

// 认证

def secret\_name = "registry-pull-secret"

def docker\_registry\_auth = "275cb37e-70c8-4d87-9c66-60b482a87777"

def git\_auth = "b2372116-00ee-46fc-8eda-71753c97a082"

def k8s\_auth = "854ae16f-dfa2-46ee-a214-a4c4aed0768e"

podTemplate(label: 'jenkins-slave', cloud: 'kubernetes', containers: [

containerTemplate(

name: 'jnlp',

image: "registry-vpc.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_jdk1.8"

),

],

volumes: [

hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/var/run/docker.sock'),

hostPathVolume(mountPath: '/usr/bin/docker', hostPath: '/usr/bin/docker')

],

)

{

node('jenkins-slave'){

// 第一步

stage('拉取代码'){

checkout([$class: 'GitSCM', branches: [[name: "${Branch}"]], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: "${git\_auth}", url: "${git\_address}"]]])

}

// 第二步

stage('代码编译'){

sh "mvn clean package -Dmaven.test.skip=true"

}

// 第三步

stage('构建镜像'){

withCredentials([usernamePassword(credentialsId: "${docker\_registry\_auth}", passwordVariable: 'password', usernameVariable: 'username')]) {

sh """

echo '

FROM registry-vpc.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:tomcat.8.5

RUN rm -rf /usr/local/tomcat/webapps/\*

ADD target/\*.war /usr/local/tomcat/webapps/ROOT.war

' > Dockerfile

docker build -t ${image\_name} .

docker login -u ${username} -p '${password}' ${registry}

docker push ${image\_name}

"""

}

// 第四步

stage('部署到K8S平台'){

sh """

sed -i 's#\$IMAGE\_NAME#${image\_name}#' deploy.yml

sed -i 's#\$SECRET\_NAME#${secret\_name}#' deploy.yml

"""

kubernetesDeploy configs: 'deploy.yml', kubeconfigId: "${k8s\_auth}"

}

}

}

}

#如果少大括号 注意补一下，一般格式很容易出错

#注释一下上面的代码意思 和我出现的问题

1）上面是拉取代码，打包，推送镜像的步骤 ，还没有把项目部署到K8S里面 下一章节开始部署项目到 K8S 如果还需要单元测试的话 那还需要配置 更多的参数，直接写到jenkins-slave的镜像里面去配置扫描代码的服务·····image: "registry-vpc.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_jdk1.8"是公共镜像 所以我这边不能用户名和密码



2）#官方参考文档

#https://github.com/jenkinsci/kubernetes-plugin

3）#我先把最后一步部署到K8S里面拿掉，脚本只执行拉取代码，打包，推送镜像到仓库，下面是我拿掉的步骤

// 第四步

stage('部署到K8S平台'){

sh """

sed -i 's#\$IMAGE\_NAME#${image\_name}#' deploy.yml

sed -i 's#\$SECRET\_NAME#${secret\_name}#' deploy.yml

"""

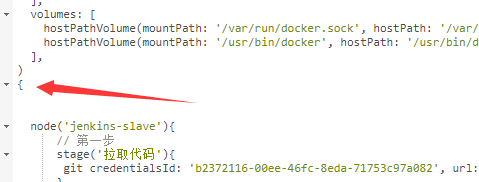
kubernetesDeploy configs: 'deploy.yml', kubeconfigId: "${k8s\_auth}"

}

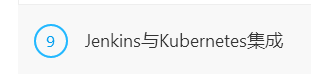
4） #部署pipeline项目容易出错的几个地方

#注意 我自己生成的拉取项目的pipeline部署脚本··竟然不成功,变量都不生效 ，变量一定弄双引号

#注意pipeline脚本式部署的这个大括号的位置···要包裹node 最后还一个大括号 组成一队 我这老错



下一章节，部署项目到K8S



#先说明下瑞松镜像之后 可以 测试下镜像，然后在去线上部署 稳当一点：

’推送镜像到镜像仓库之后 我们可以测试下 docker run一下 能不能起来这个镜像 然后在区

Docker run -d -p 666.8080 image（镜像地址，我省略了，不好找，反正就是这个测试的意思）

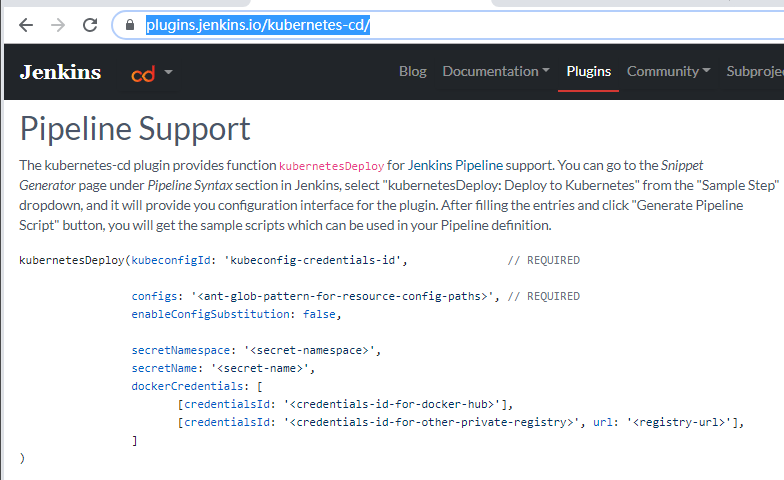
然后访问 这个666的端口看看 行不行，然后去安装K8S和jenkins集成的插件

Kubernetes Continuous Deploy插件：

用于将资源配置部署到Kubernetes。

插件介绍：<https://plugins.jenkins.io/kubernetes-cd>

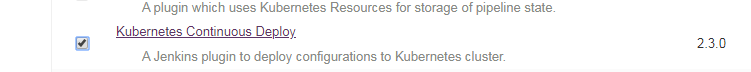
1）Jenkisn插件:Kubernetes Continuous Deploy

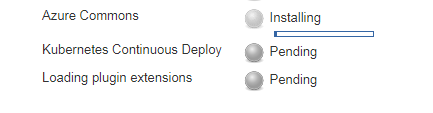


#K8S集成jenkins的插件的网站有pipeline的写法···

#注：主要两个是必须的 K8S的kubeconfigId文件 和K8S的资源文件 比如部署的 deploy.yaml这种类型的资源文件

2）#然后进去到jenkins插件里面找到





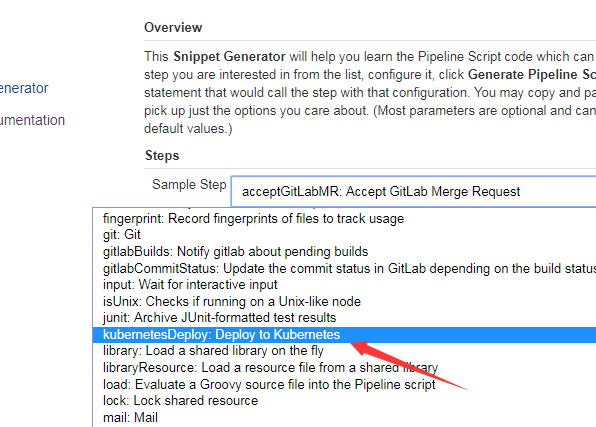
需要依赖这三个 一起安装了··

Azure Commons

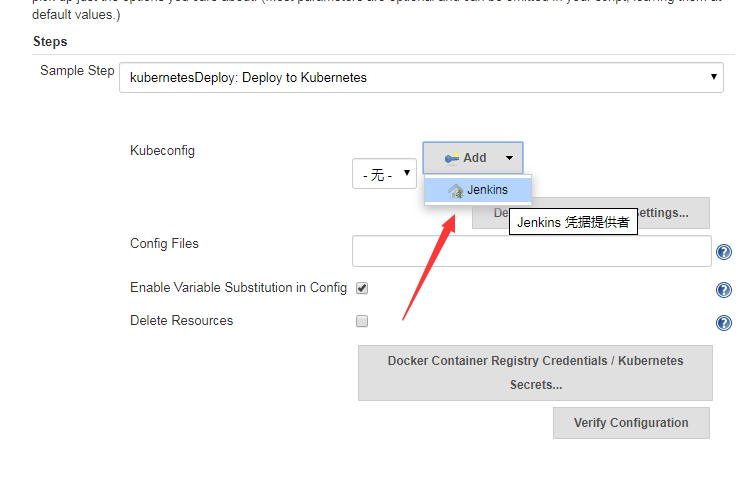
Kubernetes Continuous Deploy

Loading plugin extensions

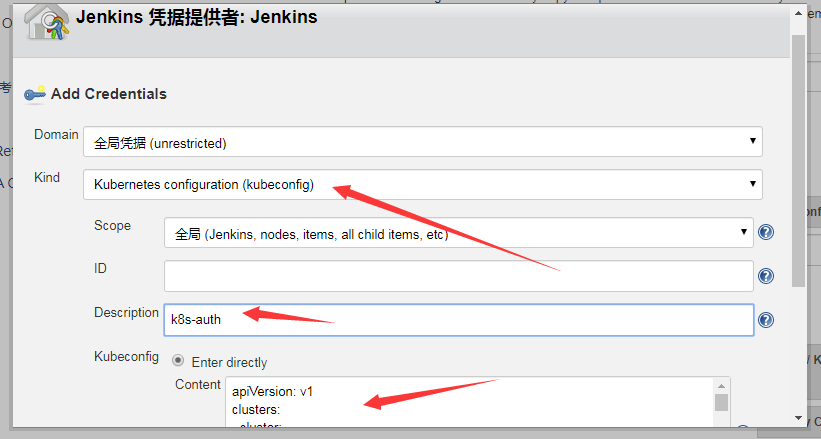
1. 然后找到这个插件生成pipeline脚本··
2. ）找到插件



2））点击这个插件进来需要添加一个jenkins凭据



3））设置凭据，我们需要把kubeconfig的.kube/config文件拷贝过来，这个文件一般在master的家目录



#文件如下，整个拷贝进去，拷贝一定要cat /root/.kube/config这样查看 拷贝啊 要不然 很容易错误 ，我就是这个地方错误 弄了半天

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUQ2ekNDQXRPZ0F3SUJBZ0lERkVrVU1BMEdDU3FHU0liM0RRRUJDd1VBTUdveEtqQW9CZ05WQkFvVElXTmoKWkRKaU9EazNaamMxTkdJME9XRTFPRE16Tm1

VMVpHRTBZekJrTURRM1lURVFNQTRHQTFVRUN4TUhaR1ZtWVhWcwpkREVxTUNnR0ExVUVBeE1oWTJOa01tSTRPVGRtTnpVMFlqUTVZVFU0TXpNMlpUVmtZVFJqTUdRd05EZGhNQjRYCkRUSXdNREl3TWpBM016VXdNRm9YRFRRd01ERXlPREEzTkRBeE1Wb3

dhakVxTUNnR0ExVUVDaE1oWTJOa01tSTQKT1RkbU56VTBZalE1WVRVNE16TTJaVFZrWVRSak1HUXdORGRoTVJBd0RnWURWUVFMRXdka1pXWmhkV3gwTVNvdwpLQVlEVlFRREV5RmpZMlF5WWpnNU4yWTNOVFJpTkRsaE5UZ3pNelpsTldSaE5HTXdaREEwT

jJFd2dnRWlNQTBHCkNTcUdTSWIzRFFFQkFRVUFBNElCRHdBd2dnRUtBb0lCQVFEMmJrTEhJdHRjYWFWNHNlZ2swb1NUUTQwaTUzdGsKRVF4K3E0ZisrLzNSdERqaXprajBDWUNpSWMzeTNKZHRaSkI4MDBGWG8zNjBTTmlIaGQwZVF4b1pqaWo4b1k5Zgpj

Y0diY0R1VjluaUp3R0JSL0xoUGZ0aEJYbmhPbVlKakVicUFHMEVlTW9CNTBOdDVGUnBIVWZsMitTaUxYbVVXCmlpSExTY2JidkcwZlBZNzVieXR3em55ZDhVbzMwZW5KQk41QSt1MHh4OFdYZ0NmL05Nc3NSemhENVlvRXcyOTQKZ2h6dUV2TExKMGlSWVN

1Z01UajNMMDJkaFAxRDRDcXpQRmYwMWZYK0o4Q2cxSFd4VWl2S3dNdTJUQThMTHE3Tgp4dlpzY2YwZGlKaGNocWQ2aGcyQnozQkpFWkMyZmpraE1ieVc4cGZPQUFwZDFRRGJGdUlWNDdjRkFnTUJBQUdqCmdaa3dnWll3RGdZRFZSMFBBUUgvQkFRREFnS3

NNQThHQTFVZEV3RUIvd1FGTUFNQkFmOHdQQVlJS3dZQkJRVUgKQVFFRU1EQXVNQ3dHQ0NzR0FRVUZCekFCaGlCb2RIUndPaTh2WTJWeWRITXVZV056TG1Gc2FYbDFiaTVqYjIwdgpiMk56Y0RBMUJnTlZIUjhFTGpBc01DcWdLS0FtaGlSb2RIUndPaTh2W

TJWeWRITXVZV056TG1Gc2FYbDFiaTVqCmIyMHZjbTl2ZEM1amNtd3dEUVlKS29aSWh2Y05BUUVMQlFBRGdnRUJBSlFXaFZoOXYyZDZkN3p3RHJmTWwyR3kKVlAxUCt3dzcydXArRTdaVUNVSnpZZUs5a0JxaERkYmtFYXdYTGJwYzhKcy8yWlFWQjNsVE1L

T3ppcGh2VGNhVQpyb01BR2JzOWVqc0hJNjIzeHZXWVJ6OW5OV1B5M09GNVhWSThyV0p5eWFXRndoWjdHZGJ6V0x1VjlsOWp3dlpCClcrK2J2T2xJRW9ZRlpoZE5jc29haklyOWJJcmNZcFNYUVNpRzJqTWliVWF6UTEwTkI3b0FzZWhxanl3SDRubjkKbU4

yZEtnNXlHVHRuOWtUREF6dHJEOCtvTzdqNXRod0dNOTg3RTNFYkFiZzU1WVVBSytSNDdMenFRNlNMOUQrNApGZGZHdXZ2R2VtUHB1UE1tTVJTUklTY29kL2FYV2NSMHkyV2dHT1FvakxZWXFXVGd3UDNEQllFUGtqa0cxV3M9Ci0tLS0tRU5EIENFUlRJRk

lDQVRFLS0tLS0K

server: https://172.16.8.167:6443

name: ccd2b897f754b49a58336e5da4c0d047a

contexts:

- context:

cluster: ccd2b897f754b49a58336e5da4c0d047a

user: kubernetes-admin

name: kubernetes-admin@ccd2b897f754b49a58336e5da4c0d047a

current-context: kubernetes-admin@ccd2b897f754b49a58336e5da4c0d047a

kind: Config

preferences: {}

users:

- name: kubernetes-admin

user:

client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURSekNDQWkrZ0F3SUJBZ0lJYUVvRThtMGk5UGd3RFFZSktvWklodmNOQVFFTEJRQXdhakVxTUNnR0ExVUUKQ2hNaFkyTmtNbUk0T1RkbU56VTBZalE1WVRVNE

16TTJaVFZrWVRSak1HUXdORGRoTVJBd0RnWURWUVFMRXdkawpaV1poZFd4ME1Tb3dLQVlEVlFRREV5RmpZMlF5WWpnNU4yWTNOVFJpTkRsaE5UZ3pNelpsTldSaE5HTXdaREEwCk4yRXdIaGNOTWpBd01qQXlNRGN6TlRBd1doY05NekF3TVRNd01EYzBOR

FUyV2pBME1SY3dGUVlEVlFRS0V3NXoKZVhOMFpXMDZiV0Z6ZEdWeWN6RVpNQmNHQTFVRUF4TVFhM1ZpWlhKdVpYUmxjeTFoWkcxcGJqQ0NBU0l3RFFZSgpLb1pJaHZjTkFRRUJCUUFEZ2dFUEFEQ0NBUW9DZ2dFQkFOVUhSWTZzeElTU25hV3p5dFcxQmVT

Y0t6QXNpTkVGCnc1c3J3azUrQVNweTRNUVBQVERUelF1bTB1RWJCcThOZWxFcU5vWU81YlZlYlhtSmNqS1cwMVhCRDBvMXJiK2EKTEdYOXVMVS9sL3V0MWYzWTVqSDFDL0VZR0dtYTFYWjcxYzhqVjg5L1gzcmdzQ3VHRnRISXZncVV2UHpyRmljZQpNdWV

DM1JseWNlcHdZTExOUmJGS3B1Tk4zbjVUTDdJWXhScWtTSmNvZTRKMWp3bDNqaEFWM2J1a0JEanhRRTBuCk1kNEdISW1GQVBueDJab1JkOGR3THJHOE1ZbUpEQmtLdGhJSHRaM0xRS0VlRmhmd3NhSjl0dmREdW5qNHNpQTUKUFY2VGI4eFZoUCtqdjZYcX

JOeEljcnF2K1F2RldTS2lhNTVJRVNMT3NKS2RGMmlXZU5wTDhXc0NBd0VBQWFNbgpNQ1V3RGdZRFZSMFBBUUgvQkFRREFnV2dNQk1HQTFVZEpRUU1NQW9HQ0NzR0FRVUZCd01DTUEwR0NTcUdTSWIzCkRRRUJDd1VBQTRJQkFRRG5mNkZpWTNaSzdvNDhrO

C9VQUNySEZEMmZETE82eXZ1Vzl4aDBSNWdYUmVFSDZ0NDQKY3loSzBYVGJuZmZFbHBKM1BJNzlOcW41aFMrOHhRRGswb1lJMGFQWnVyKzBMbFpVVk1PWmRnNXV5QlpsVlNDMwpIazl5bXAxY1lkcm5wdWpWeHUyYTVIZUZDcmlBRDdlNjR6UWExcjFEUTZX

aUxJT25KaTd1cm1sQ0NQL1FKeWt0CkdhM2R6OGRNRkluNitubnVFQytoeUdneWZSZUZIZ2hlM1NVbXQ0WlZwNTJLOEdSdE1JTko0UE4yYjU4NDRSVWsKQXpTbyt6cCthamkzUjE0SlRUem84U0EvMTdKY1ZmUldLQVdmdTYxNlRxWndrRTkzWUdzdzQrNXp

aZ0FWK3BQego5M2lUMDFzcEhQV0JKajd2Ri9mR3BrNHNXTzlMS05VWXFOaUUKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=

client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBMVFkRmpxekVoSktkcGJQSzFiVUY1SndyTUN5STBRWERteXZDVG40QktuTGd4QTg5Ck1OUE5DNmJTNFJzR3J3MTZVU28yaGc3bHRWNXRlWW

x5TXBiVFZjRVBTald0djVvc1pmMjR0VCtYKzYzVi9kam0KTWZVTDhSZ1lhWnJWZG52Vnp5Tlh6MzlmZXVDd0s0WVcwY2krQ3BTOC9Pc1dKeDR5NTRMZEdYSng2bkJnc3MxRgpzVXFtNDAzZWZsTXZzaGpGR3FSSWx5aDdnbldQQ1hlT0VCWGR1NlFFT1BGQ

VRTY3gzZ1ljaVlVQStmSFptaEYzCngzQXVzYnd4aVlrTUdRcTJFZ2UxbmN0QW9SNFdGL0N4b24yMjkwTzZlUGl5SURrOVhwTnZ6RldFLzZPL3BlcXMKM0VoeXVxLzVDOFZaSXFKcm5rZ1JJczZ3a3AwWGFKWjQya3Z4YXdJREFRQUJBb0lCQVFDSWFGQ2M3

SG9sM2ZBKwpnNk1WTndZZi9mUVZzOTZqNFlvQVdZUGhXUVFNRjJWZmhkYUw1b084dTBUN1NaLzlCYWRXUTY2SUp3L21GQ0lMCm1meGE4b00rdlJHNlZpYjhzbzlOZTZzRHZFWG9yY2FCOXBwb1VONGxzbVVuY2pMUDNMbnJHbG1IRDZ1bTlpczQKbzNNSFF

BQ1lxK2RjR0hpbDhjdHN2VG93cWp6UmZab05mVmRhVDNVK2NJaWp1ZXB3ekx2amNhOWUwNEsvMUxnOApOcVUwTmRMZnFBSjN0ck5YTjNIa1g4Y0NIOEk4aTFUMm5pU1hzNDVrMi9HU0ZhbUo4NW1BcTQva0l5dGtFZjFxCmUvYVFMVTRrV3lld0FVMGFEbV

E1cGNQZ1dLcVdDa1FtMEwrWjJJWnFpQTBVd2pzQnZUWFpqU3NoZXNFUFNCYTIKeS9xQ21LZEJBb0dCQU5kRmlOcEZVekFORzFVSUFWNlAxSjQwR3RQZmNRNklhb1VoMng4R0RmV1lqS2tsSnJFWgpTVDh5cGt0NmZSMEp0SUI3N1hQQ24xRHhlb21WN1VWb

FhiK3VTbFJaQmVmZUFIUUM1VGUrcHQ2bEE1MjlkVXZaClRKREVNNkw4S3lLYzNBdklkbm5BSUFyRE9aM0t3a01VQnNqd0dSVlRSSTdkd0JnSGZrNW9XQmVoQW9HQkFQMVYKRnR0MlhtMmJpQzliM1g2MjBmZlJNY3VoeEsyZEdJT095UGFySStVQk10U3pm

QVIxRjhwZkx1dVQwV1RkN0VwQQpaa0gvd0cvOTBhQ0xycGVFcW13RlBVSHdjSTRkTnhZNUxMcUw1RWFsemdMMUo4RDB4a29PNVlLQ3MwdFVGRWRkClBucEk4SVQ4VEFwMzh4eGdqdS9tVkszeE5zeElrY2ZYdDQ4cC9mMkxBb0dCQU1ENHVDRFJuQ3lxMHl

GNk42QUkKMXRPcWFaSWI3a0RVMElmZlBZbWw1OFdNcGlySi90RXFsWnB0VU56QVJIOVBQOXNkUEZKVTJITzZ3OW1wNkRRWgpGZHpKNUIrUVhkYm9Ed2Z2d0lJa0NYbnoralhncTlEK1RjQkg0UEh0a3MxSVI0allmanZObWV1TDdZNGhQOXJaCmpPdDdMME

VLajU1eDhjeVlpd1RHVEpHQkFvR0FGaDFNYm9JSndqVzJrTFBMUlJsK2dNeWhOUzU4OWM3bDZlNFgKK3ZoN25LMGJHOWhETmp4bWhFTUUzSW05MkxsS3ZEeWM4SnY3MTRybnRzVWF5NTFJSnovWjFoekFKcGZFVEV0WApiVDF6QXZ2TVhSTzdrYXVNZDhDU

EpuSjBabG9Ebnlzem5nY3AvK0hxWFd1VkxGLzllbEc0Q0tzcTVuQmJVWCt5ClJXd2NBUHNDZ1lFQWxIb0t0K0dVTzh5NHRNb21oVkpDT3Q0MWhNYjZtTHkvRHZXVUVkZnJLaVdEUzl5QVRURWQKWU9GRWg5Rk8wbk0wOE11eWJmdFR4MXpTUHNsZlo0UFoz

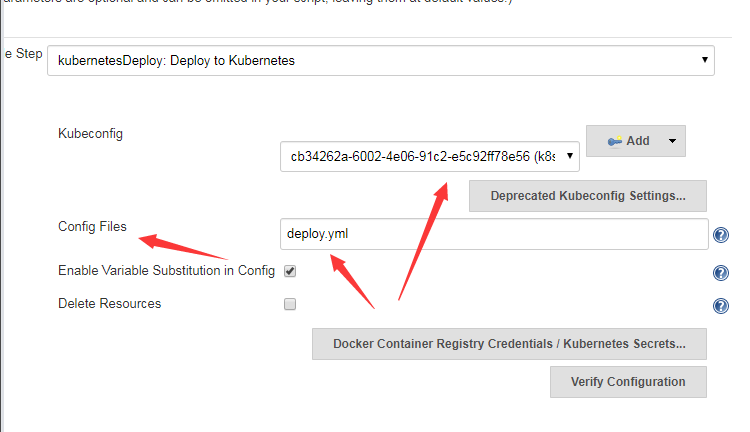
UjZhcmhyV3lmL3FRRllWeXlXQ01FMXN3Z05nVkZMVgpVRVFzZ3FXZCtvQW5DZEIwS0dhcjJlMk0vY21ubGZOdTJxMTNkbmROM20xeFZwa214elNGa29JPQotLS0tLUVORCBSU0EgUFJJVkFURSBLRVktLS0tLQo=

#然后取个名字k8s-auth,等下就能引用这个名字

4））然后ADD添加这个文件··就能选择这个kubeconfig的ID了···

然后有个config files文件···这个是你部署K8S项目的资源文件，一般是deploy.yaml文件·这个文件一般也放在gitl仓库

因为他会找这个git项目文件，不会从其他地方找



more deploy.yml

#一个简单的demo部署一个java项目

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: web

spec:

replicas: 3

selector:

matchLabels:

app: java-demo

template:

metadata:

labels:

app: java-demo

spec:

imagePullSecrets:

- name: $SECRET\_NAME

containers:

- name: tomcat

image: $IMAGE\_NAME

ports:

- containerPort: 8080

name: web

livenessProbe:

httpGet:

path: /

port: 8080

initialDelaySeconds: 60

timeoutSeconds: 5

failureThreshold: 12

readinessProbe:

httpGet:

path: /

port: 8080

initialDelaySeconds: 60

timeoutSeconds: 5

failureThreshold: 12

---

apiVersion: v1

kind: Service

metadata:

name: web

spec:

type: NodePort

selector:

app: java-demo

ports:

- protocol: TCP

port: 80

targetPort: 8080

---

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: web

spec:

rules:

- host: java.example.com

http:

paths:

- path: /

backend:

serviceName: web

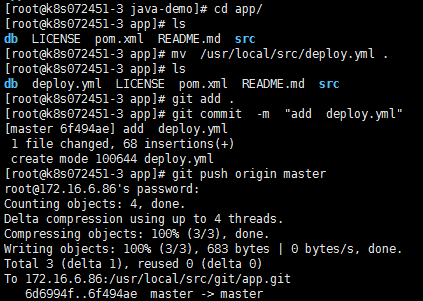
servicePort: 80

#一般我们把镜像的仓库密码···改成一个变量，因为很可能每次测试都不一样imagePullSecrets

#如果是线上的项目 还需要加上资源限制和健康检查，我这是实验个小项目···所以省略了

#还一个项目的镜像$IMAGE\_NAME 也可以改成一个变量，我们每次修改这两个变量 然后再kubectl apply -f deploy.yml

5）））#我们说了 需要把deploy.yml放到git仓库，jenkins会从这个仓库 找这个文件去部署



kubernetesDeploy configs: 'deploy.yml', kubeConfig: [path: ''], kubeconfigId: 'cb34262a-6002-4e06-91c2-e5c92ff78e56', secretName: '', ssh: [sshCredentialsId: '\*', sshServer: ''], textCredentials: [certificateAuthorityData: '', clientCertificateData: '', clientKeyData: '', serverUrl: 'https://']

#整个的K8S集成jenkins的pipeline的脚本如下，只要就是添加的第四步骤，注意这个格式 需要就是修改kubeconifgid，这个id就是之前那个秘钥了

我这边都是在变量这修改

阿里云镜像凭据配置

#我们deploy里面还需要一个K8S连接阿里云镜像的认证凭据

kubectl create secret docker-registry registry-pull-secret --docker-username=hzh@1695447190754546 --docker-password=123456a! --docker-server=registry-vpc.cn-hangzhou.aliyuncs.com --docker-email=12345678@qq.com

#注意这是我阿里云的镜像仓库地址 ， 邮件我乱写的·因为不写还报错 就随便写了个·

#最后一步骤保存 部署这个jenkins上的java项目啦



#我这边部署成功了，访问下这个ingress地址···我绑定了 DNS



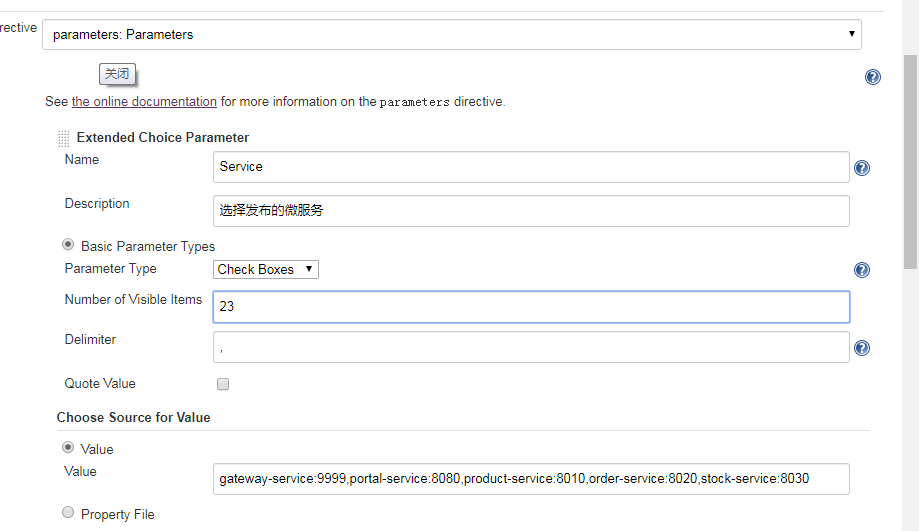
#我们这个项目没添加数据库，所有有些功能是用不了的，我这个项目可以添加一个数据库 在K8S集群之后，然后在部署一次

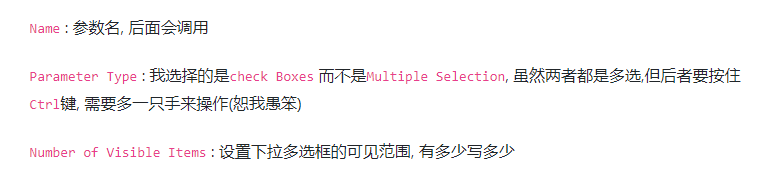
#友情提醒，Jenkinsfile和deploy.yml最好都放到git的项目下···这样就算jenkins坏了 也好还原

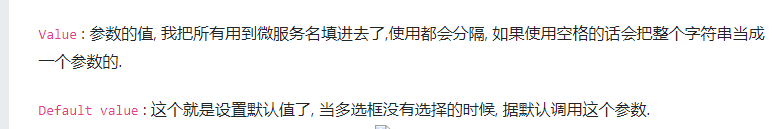
三 Pipeline声明式 第三步骤 创建多微服务镜像和推送多微服务镜像

#我们先描述一下 我们这个不好走

1. 我们这个推送会有一个for循环，循环创建镜像 和推送镜像··
2. 我们的dockerfile部署的项目 就放在拉取的镜像下面，···而且全部的微服务项目 都会放在 拉取的镜像，会很方便重新打包 部署
3. 我们会用到Extended Choice Parameter 需要说明一下 他的用法，只要是将来可以选择多个微服务一起部署，









我使用的分隔符是逗号

前三部分脚本 如下（有第四部分 修改配置的内容，按照道理要放第四部分的）

#!/usr/bin/env groovy

// 公共

def registry = "registry-vpc.cn-hangzhou.aliyuncs.com"

// 项目

def project = "goldplusgold"

// def app\_name = "demo"

// def image\_name=${registry}/${project}/k8s:\${service\_name}\_${BUILD\_NUMBER}

// def image\_name = "${registry}/${project}/k8s:${service\_name}\_{BUILD\_NUMBER}"

def git\_address = "172.16.6.86:/usr/local/src/jenkins"

// 认证

def secret\_name = "registry-pull-secret"

def docker\_registry\_auth = "376cf21a-9e38-495f-ae40-d2d32da36b81"

def git\_auth = "6d663f61-dd33-4771-b33c-285caf09ac64"

def k8s\_auth = "854ae16f-dfa2-46ee-a214-a4c4aed0768e"

pipeline {

agent {

kubernetes {

label "jenkins-slave"

yaml """

apiVersion: v1

kind: Pod

metadata:

name: jenkins-slave

namespace: ms

spec:

containers:

- name: jnlp

image: registry.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_jdk1.8

imagePullPolicy: Always

volumeMounts:

- name: docker-cmd

mountPath: /usr/bin/docker

- name: docker-sock

mountPath: /var/run/docker.sock

- name: m2

mountPath: /root/.m2

volumes:

- name: docker-cmd

hoatPath:

path: /usr/bin/docker

- name: docker-sock

hostPath:

path: /var/run/docker.sock

- name: m2

hostPath:

path: /tmp/m2

"""

}

}

parameters {

gitParameter branch: '', branchFilter: '.\*', defaultValue: 'master', description: '请选择你要发布的分支', name: 'Branch', quickFilterEnabled: false, selectedValue: 'NONE', sortMode: 'NONE', tagFilter: '\*', type: 'PT\_BRANCH\_TAG'

extendedChoice defaultValue: '', description: '选择发布的微服务', multiSelectDelimiter: ',' , name: 'Service', type: 'PT\_CHECKBOX', value: 'gateway-service:9999,portal-service:8080,product-service:8010,order-service:8020,stock-service:8030'

}

stages {

stage('拉取代码') {

steps{

checkout([$class: 'GitSCM',

branches: [[name: "${Branch}"]],

doGenerateSubmoduleConfiguratsons: false,

extensions: [], submoduleCfg: [],

userRemoteConfigs: [[credentialsId: '6d663f61-dd33-4771-b33c-285caf09ac64',

url: '172.16.6.86:/usr/local/src/jenkins']]])

}

}

stage('代码编译'){

steps {

sh """ mvn clean package -Dmaven.test.skip=true """

}

}

stage('构建镜像'){

steps {

withCredentials([usernamePassword(credentialsId: ' 376cf21a-9e38-495f-ae40-d2d32da36b81', passwordVariable: 'password', usernameVariable: 'username')]) {

sh """

docker login --username="${username}" --password="${password}" "${registry}"

for service in \$(echo "${Service}" | sed 's/,/ /g' ); do

service\_name=\${service%:\*}

image\_name="${registry}"/"${project}"/k8s:\${service\_name}\_"${BUILD\_NUMBER}"

cd \${service\_name}

if ls | grep biz &>/dev/null; then

cd \${service\_name}-biz

fi

docker build -t \${image\_name} .

docker push \${image\_name}

echo "${WORKSPACE}"

cd "${WORKSPACE}"

done

"""

}

}

}

stage('deyloy'){

steps{

sh """

for service in \$(echo "${Service}" | sed 's/,/ /g' ); do

service\_name=\${service%:\*}

sp\_name=\${service%-\*}

image\_name="${registry}"/"${project}"/k8s:\${service\_name}\_"${BUILD\_NUMBER}"

sed -i -r "s#(image: )(.\*)#\1\${image\_name}#g" "${WORKSPACE}"/k8s/\${sp\_name}.yaml

done

"""

}

}

}

}

#注释：一定要小心 各种坑啊 ，主要是双引号几个包裹的引号 比如pipeline { }这个引号是包裹全局……usernameVariable: 'username')]) {}这个引号是构建只是包裹第三部分，就是提交镜像需要用到

#已经循环修改了yaml文件··注意service\_name=\${service%:\*}这个文件 下面修改配置文件的时候需要换了····portal-service:8080第三步骤的时候需要service\_name=\${service%:\*} 就是文件变成为

portal-service，因为我目录就是portal-service这个样子。但是第四步骤 我文件是另外的样子了

（三步骤目录）

（四步骤修改的文件格式）

第四部分pipeline部署部分

在jenkins里面部署k8s脚本还有几个步骤需要达成

**1.这边这个部署需要jenkins-slave里面带有kubectl这个二进制脚本··要不然无法部署**

#所以我们jenkins-slave需要修改Dockerfile，添加一个kubectl，可以在docker in docker里面查看 K8S集群状态

**more Dockerfile**

FROM centos:7

LABEL maintainer hzh

RUN yum install -y java-1.8.0-openjdk maven curl git libtool-ltdl-devel && \

yum clean all && \

rm -rf /var/cache/yum/\* && \

mkdir -p /usr/share/jenkins

COPY slave.jar /usr/share/jenkins/slave.jar

COPY jenkins-slave /usr/bin/jenkins-slave

COPY settings.xml /etc/maven/settings.xml

COPY kubectl /usr/bin/kubectl

RUN chmod +x /usr/bin/jenkins-slave \

&& chmod +x /usr/bin/kubectl

ENTRYPOINT ["jenkins-slave"]

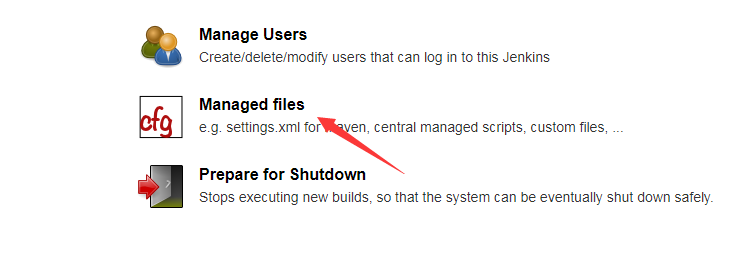
**2.需要kubeconfig文件，要不然也无法部署，在jenkins添加kubeconfig配置的方法：**

之前node单个微服务的时候用到了 kubernetes Continuous Deploy插件（这个不知道能不能一次部署多个微服务）

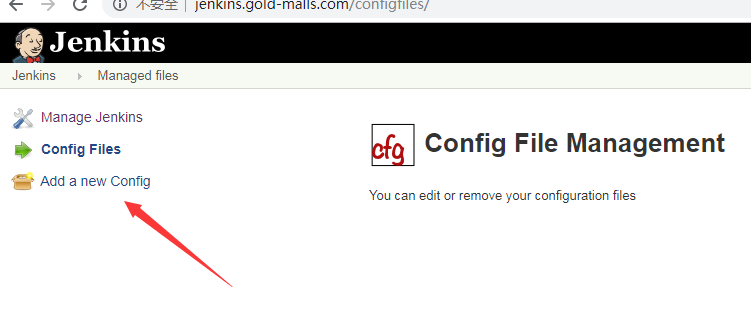
现在集群部署多个微服务 再介绍一个比较好的插件（这个肯定可以部署多个微服务的）

这个插件叫**[Config File Provider Plugin](https://plugins.jenkins.io/config-file-provider)**  我已经部署好了

1）点击[**Manage Jenkins**](http://jenkins.gold-malls.com/manage) 找到 下面的 Managed files

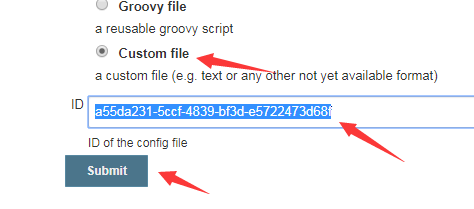


2）点击这个add a new config

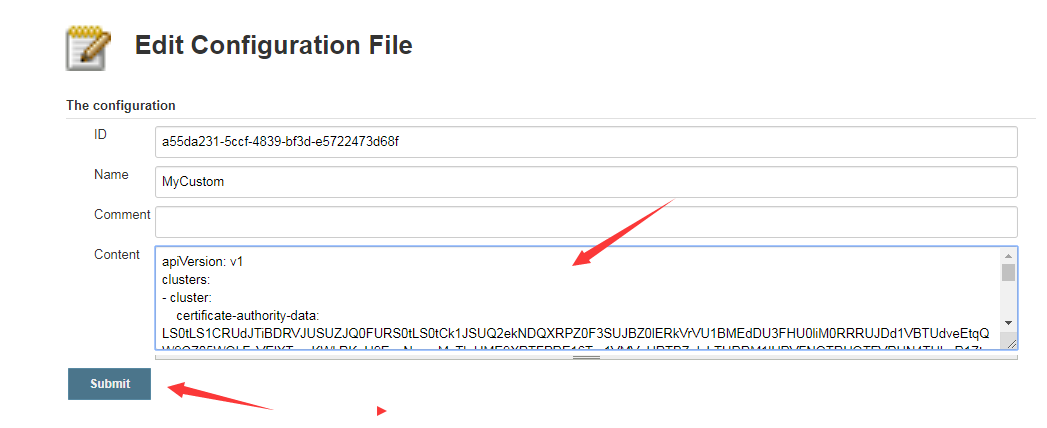


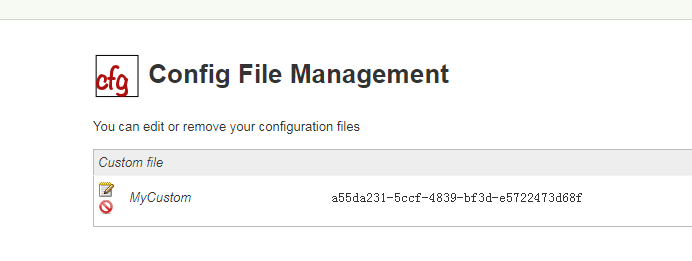
选择这个自定义一个文件，拷贝这一串ID 很有用的，然后提交

Id：a55da231-5ccf-4839-bf3d-e5722473d68f 这个id等下有用



#这里需要把你kubeconfig文件内容黏贴进来cat /root/.kube/config 就是这个文件的内容····如果不是kuebadm部署的 也就是二进制部署的 还需要先生成config这个文件···





然后我们去 pipeline脚本引用 这个id

在全局pipeline的脚本 添加这个变量 等下要引用的

def k8s\_auth = "a55da231-5ccf-4839-bf3d-e5722473d68f"

#然后怎么引用这个配置呢 在pipeline里面需要增加一个配置格式

configFileProvider([configFile(fileId: "${k8s\_auth}",

targetLocation: "admin.kubeconfig")]){ } 注意这个括号 要包裹配置

#然后加上配置

kubectl create -f "${WORKSPACE}"/k8s/\${sp\_name}.yaml --record --kubeconfig admin.kubeconfig

注意：上面(fileId: "${k8s\_auth}", 很重要 下面来一个部署的完整配置 （已经测试过）

**第四步骤完整部署配置如下**

#!/usr/bin/env groovy

// 公共

def registry = "registry-vpc.cn-hangzhou.aliyuncs.com"

// 项目

def project = "goldplusgold"

// def app\_name = "demo"

// def image\_name=${registry}/${project}/k8s:\${service\_name}\_${BUILD\_NUMBER}

// def image\_name = "${registry}/${project}/k8s:${service\_name}\_{BUILD\_NUMBER}"

def git\_address = "172.16.6.86:/usr/local/src/jenkins"

// 认证

def secret\_name = "registry-pull-secret"

def docker\_registry\_auth = "1beb95b8-faa9-40e9-95df-5ac03d9a423a"

def git\_auth = "8eb30d90-1392-4edd-abe4-7f8b7838ef68"

def k8s\_auth = "2a80f35e-3b67-439a-85b0-b82bec48b3b5"

pipeline {

agent {

kubernetes {

label "jenkins-slave"

yaml """

apiVersion: v1

kind: Pod

metadata:

name: jenkins-slave

namespace: ms

spec:

containers:

- name: jnlp

image: registry-vpc.cn-hangzhou.aliyuncs.com/goldplusgold/k8s:jenkins\_slave\_v1

imagePullPolicy: Always

privileged: true

volumeMounts:

- name: docker-sock

mountPath: /var/run/docker.sock

- name: m2

mountPath: /root/.m2

volumes:

- name: docker-sock

hostPath:

path: /var/run/docker.sock

- name: m2

hostPath:

path: /tmp/m2

"""

}

}

parameters {

gitParameter branch: '', branchFilter: '.\*', defaultValue: 'master', description: '请选择你要发布的分支', name: 'Branch', quickFilterEnabled: false, selectedValue: 'NONE', sortMode: 'NONE', tagFilter: '\*', type: 'PT\_BRANCH\_TAG'

extendedChoice defaultValue: '', description: '选择发布的微服务', multiSelectDelimiter: ',' , name: 'Service', type: 'PT\_CHECKBOX', value: 'gateway-service:9999,portal-service:8080,product-service:8010,order-service:8020,stock-service:8030'

}

stages {

stage('拉取代码') {

steps{

checkout([$class: 'GitSCM',

branches: [[name: "${Branch}"]],

doGenerateSubmoduleConfiguratsons: false,

extensions: [], submoduleCfg: [],

userRemoteConfigs: [[credentialsId: "${git\_auth}",

url: "${git\_address}"]]])

}

}

stage('代码编译'){

steps {

sh """ mvn clean package -Dmaven.test.skip=true

docker ps -a

"""

}

}

stage('构建镜像'){

steps {

withCredentials([usernamePassword(credentialsId: "${docker\_registry\_auth}", passwordVariable: 'password', usernameVariable: 'username')]) {

sh """

docker login --username="${username}" --password="${password}" "${registry}"

for service in \$(echo "${Service}" | sed 's/,/ /g' ); do

service\_name=\${service%:\*}

image\_name="${registry}"/"${project}"/k8s:\${service\_name}\_"${BUILD\_NUMBER}"\_`git rev-parse --short HEAD`

cd \${service\_name}

if ls | grep biz &>/dev/null; then

cd \${service\_name}-biz

fi

docker build -t \${image\_name} .

docker push \${image\_name}

echo "${WORKSPACE}"

cd "${WORKSPACE}"

done

"""

}

}

}

stage('deyloy'){

steps{

configFileProvider([configFile(fileId: "${k8s\_auth}",

targetLocation: "admin.kubeconfig")]){

sh """

for service in \$(echo "${Service}" | sed 's/,/ /g' ); do

service\_name=\${service%:\*}

sp\_name=\${service%-\*}

image\_name="${registry}"/"${project}"/k8s:\${service\_name}\_"${BUILD\_NUMBER}"\_`git rev-parse --short HEAD`

sed -i -r "s#(image: )(.\*)#image: \${image\_name}#g" "${WORKSPACE}"/k8s/\${sp\_name}.yaml

cat "${WORKSPACE}"/k8s/\${sp\_name}.yaml

kubectl apply -f "${WORKSPACE}"/k8s/\${sp\_name}.yaml --kubeconfig admin.kubeconfig --record

done

"""

}

}

}

}

}

#注意 我这个格式 真的是一个标点都不能动····很多次莫名其妙的错误，配置出来的，git rev-parse --short HEAD这是能取到commit的前几个id

小节：出错了很多地方，而且是第一天OK第二天莫名其妙的错误 太伤心了，列举一下出错的地方

1：各种符号台容易出错了，少一个标点，有时候没任何出错提示·完全靠自己（尤其是那些pipeline需要包裹所有内容的 两个大括号，）

2．我的jenkins-slave镜像第一天还OK的，第二天莫名其妙的 docker不能找到 我已经挂载好了··还是不行 没办法 只能重新部署dockefile，把docker拷贝到镜像

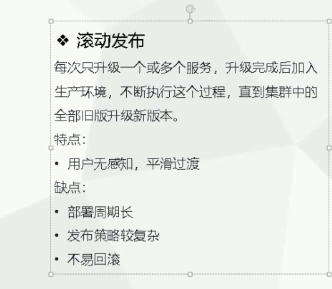
3.我所有的什么密码 id都改成了变量，有的时候用变量真的不行·改来改去又好了 我很无语

4.根据自己的错误慢看研究失误把···可能再一次部署还会出现意想不到的问题，多搞几套脚本，我这次是声明式的pipeline脚本 ，下次研究下node脚本式的pipleline部署方法

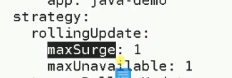
脚本式的上面也有方法，但是for循环微服务的 好像没有，下次弄一下

下一章节：





#K8S默认策略就是滚动发布····



#默认规则 允许一个不就绪····多生成一个pod

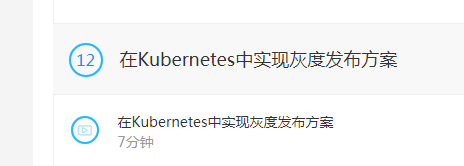
也可以手动修改···设置成百分比····也可以调整成50%，需要pod很多的情况下···

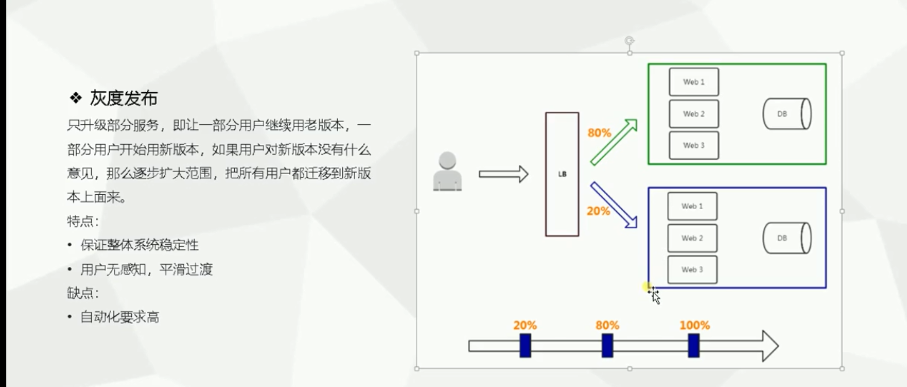
生产一定设置健康检查·····那样就算准备就绪了 也可能 没有绑定到mysql上····所以还需要存活检查，看看能不能访问···只是就绪可能还有问题

反正两个检查·就更OK

#这个章节 ，内容很少·····不做重点，这个章节也没有K8S 的CICD的内容

下一章节灰度发布····K8S都不太支持的···





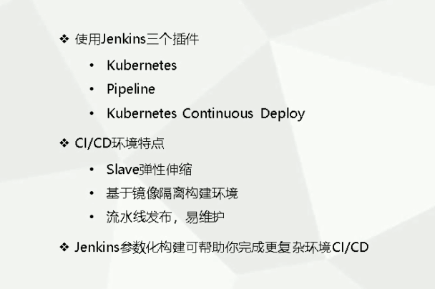
360 灰度发布思路 使用两套，两个deployment都关联一个service，在程序层面去控制 这个deploy缩容多少 那个deploy扩容多少

在程序方面去控制，K8S本生的命令是不太好的去完成的····主要还是弄两个deploy 都放在一个service 这样就能有两个镜像···



最后做一个小节：

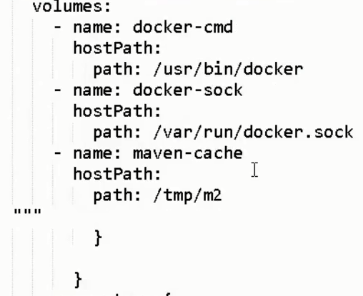
我们完成了下面的这个CICD的阶段，jenkins和K8S已经强强联手了



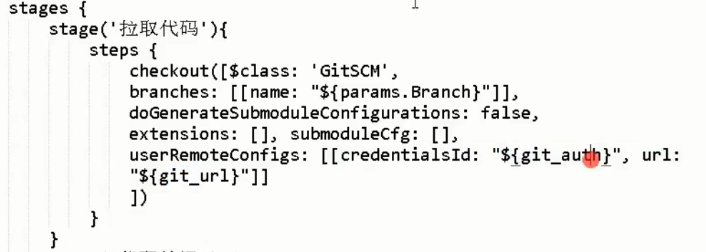
截图阿良helm部署K8S项目的pipeline的cicd的模板，留着在这里参考把

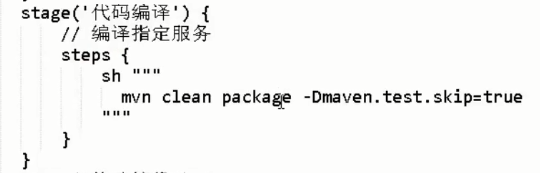


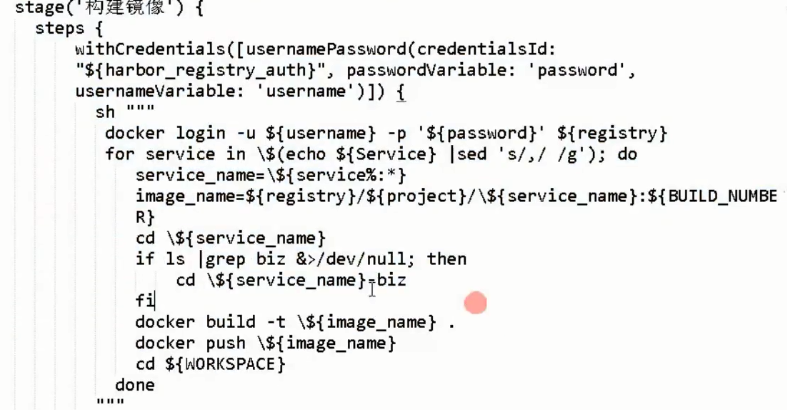




















#extendedChice这个插件就是为了部署多个微服务·才使用的

这个插件需要安装的

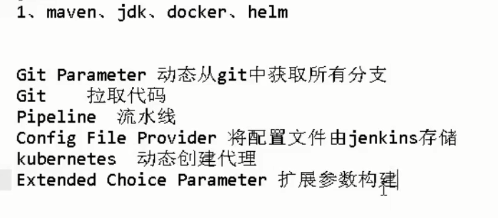
#需要的插件模板

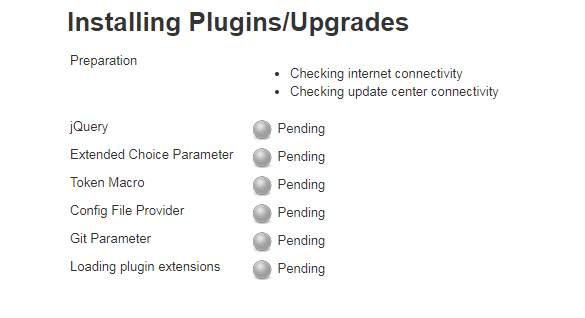
Extended Choice Parameter

Config File Provider

Git Parameter

（其他的我安装完成了）





简书部署K8S的jenkins cicd文档，竟然和我的差不多 看看

<https://www.jianshu.com/p/211ea65fc4fd>

写的很好的pipeline的cicd

<https://www.jb51.net/article/160500.htm>

parameters {

gitParameter branch: '', branchFilter: '.\*', defaultValue: 'master', description: '选择发布的分支', name: 'Branch', quickFilterEnabled: false, selectedValue: 'NONE', sortMode: 'NONE', tagFilter: '\*', type: 'PT\_BRANCH'

}

gateway-service:9999,portal-service:8080,product-service:8010,order-service:8020,stock-service:8030

优雅退出容器的方法 preStop

明阳部署的K8S的脚本格式<https://github.com/cnych> <https://www.qikqiak.com/k8s-book/> 他有多个地址

|  |  |
| --- | --- |
|  | node('haimaxy-jnlp') { |
|  | stage('Prepare') { |
|  | echo "1.Prepare Stage" |
|  | checkout scm |
|  | script { |
|  | build\_tag = sh(returnStdout: true, script: 'git rev-parse --short HEAD').trim() |
|  | if (env.BRANCH\_NAME != 'master') { |
|  | build\_tag = "${env.BRANCH\_NAME}-${build\_tag}" |
|  | } |
|  | } |
|  | } |
|  | stage('Test') { |
|  | echo "2.Test Stage" |
|  | echo "skip this step,but is not foevery!" |
|  | echo "add mvn and sonar!" |
|  | } |
|  | stage('Build') { |
|  | echo "3.Build Docker Image Stage" |
|  | sh "docker build -t maxmonkey950/jenkins-demo:${build\_tag} ." |
|  | } |
|  | stage('Push') { |
|  | echo "4.Push Docker Image Stage" |
|  | withCredentials([usernamePassword(credentialsId: 'dockerHub', passwordVariable: 'dockerHubPassword', usernameVariable: 'dockerHubUser')]) { |
|  | sh "docker login -u ${dockerHubUser} -p ${dockerHubPassword}" |
|  | sh "docker push maxmonkey950/jenkins-demo:${build\_tag}" |
|  | } |
|  | } |
|  | stage('Deploy') { |
|  | echo "5. Deploy Stage" |
|  | if (env.BRANCH\_NAME == 'master') { |
|  | input "确认要部署线上环境吗？" |
|  | } |
|  | sh "sed -i 's/<BUILD\_TAG>/${build\_tag}/' k8s.yaml" |
|  | sh "sed -i 's/<BRANCH\_NAME>/${env.BRANCH\_NAME}/' k8s.yaml" |
|  | sh "kubectl apply -f k8s.yaml --record" |
|  | } |
|  | } |

--record