

## 第三章 kubernetes 原生的 CI-CD 工具 Tekton

本节所讲内容:

实战 1: 基于 Jenkins+k8s+Git 等技术链构建企业级 DevOps 自动化容器云平台

实战 2: Jenkins 管理插件 BlueOcean 配置和使用

实战 3: Jenkins 实现 k8s 应用按照指定版本回滚

3.1 什么是 Tekton?

3.2 为什么要用 k8s 原生的 CI/CD 工具 Tekton?

3.3 Tekton 发布应用的流程

3.4 Tekton 概念

实战 4: 安装 Tekton

实战 5: 测试 Tekton 构建 CI/CD 流水线

实战 1: 基于 Jenkins+k8s+Git 等技术链构建企业级 DevOps 自动化容器云平台

1、安装 Jenkins

可用如下两种方法

1) 通过 docker 直接下载 jenkins 镜像, 基于镜像启动服务

2) 在 k8s 中部署 Jenkins 服务

2、安装 nfs 服务

#选择自己的任意一台机器, 我选择 k8s 的控制节点 xuegod63

1)、在 xuegod63 上安装 nfs, 作为服务端

```
[root@xuegod63 ~]# yum install nfs-utils -y
```

```
[root@xuegod63 ~]# systemctl start nfs
```

```
[root@xuegod63 ~]# systemctl enable nfs
```

2)、在 xuegod64 上安装 nfs, 作为客户端

```
[root@xuegod64 ~]# yum install nfs-utils -y
```

```
[root@xuegod64 ~]# systemctl start nfs
```

```
[root@xuegod64 ~]# systemctl enable nfs
```

3)、在 xuegod63 上创建一个 nfs 共享目录

```
[root@xuegod63 ~]# mkdir /data/v1 -p
```

```
[root@xuegod63 ~]# cat /etc/exports
```

```
/data/v1          192.168.1.0/24(rw,no_root_squash)
```

#使配置文件生效

```
[root@xuegod63 ~]# exportfs -arv
```

```
[root@xuegod63 ~]# systemctl restart nfs
```

3、在 kubernetes 中部署 jenkins

(1) 创建名称空间

```
[root@xuegod63 ~]# kubectl create namespace jenkins-k8s
```

(2) 创建 pv

```
[root@xuegod63 ~]# cat pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins-k8s-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.63
    path: /data/v1
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f pv.yaml
```

(3) 创建 pvc

```
[root@xuegod63 ~]# cat pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: jenkins-k8s-pvc
  namespace: jenkins-k8s
spec:
  resources:
    requests:
      storage: 10Gi
  accessModes:
    - ReadWriteMany
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f pvc.yaml
#查看 pvc 和 pv 绑定是否成功
[root@xuegod63 ~]# kubectl get pvc -n jenkins-k8s
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS
jenkins-k8s-pvc	Bound	jenkins-k8s-pv	10Gi	RWX	

(4) 创建一个 sa 账号

```
[root@xuegod63 ~]# kubectl create sa jenkins-k8s-sa -n jenkins-k8s
```

(5) 把上面的 sa 账号做 rbac 授权

```
[root@xuegod63 ~]# kubectl create clusterrolebinding jenkins-k8s-sa-cluster -n
jenkins-k8s --clusterrole=cluster-admin --serviceaccount=jenkins-k8s:jenkins-k8s-sa
```

#参考:

<https://kubernetes.io/zh/docs/reference/access-authn-authz/rbac/>

#### (6) 通过 deployment 部署 jenkins

注: 下面实验用到的镜像是 jenkins.tar.gz 和 jenkins-jnlp.tar.gz, 把这两个压缩包上传到 k8s 的 xuegod64 节点, 用如下方法手动解压:

```
[root@xuegod64 ~]# docker load -i jenkins.tar.gz
```

```
[root@xuegod64 ~]# docker load -i jenkins-jnlp.tar.gz
```

```
[root@xuegod63 ~]# cat jenkins-deployment.yaml
```

```
kind: Deployment
```

```
apiVersion: apps/v1
```

```
metadata:
```

```
  name: jenkins
```

```
  namespace: jenkins-k8s
```

```
spec:
```

```
  replicas: 1
```

```
  selector:
```

```
    matchLabels:
```

```
      app: jenkins
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: jenkins
```

```
    spec:
```

```
      serviceAccount: jenkins-k8s-sa
```

```
      containers:
```

```
        - name: jenkins
```

```
          image: xuegod/jenkins:v1
```

```
          imagePullPolicy: IfNotPresent
```

```
      ports:
```

```
        - containerPort: 8080
```

```
          name: web
```

```
          protocol: TCP
```

```
        - containerPort: 50000
```

```
          name: agent
```

```
          protocol: TCP
```

```
      resources:
```

```
        limits:
```

```
          cpu: 1000m
```

```
          memory: 1Gi
```

```
        requests:
```

```
          cpu: 500m
```

```
          memory: 512Mi
```

```
livenessProbe:
  httpGet:
    path: /login
    port: 8080
  initialDelaySeconds: 60
  timeoutSeconds: 5
  failureThreshold: 12
readinessProbe:
  httpGet:
    path: /login
    port: 8080
  initialDelaySeconds: 60
  timeoutSeconds: 5
  failureThreshold: 12
volumeMounts:
- name: jenkins-volume
  subPath: jenkins-home
  mountPath: /var/jenkins_home
volumes:
- name: jenkins-volume
  persistentVolumeClaim:
    claimName: jenkins-k8s-pvc
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f jenkins-deployment.yaml
[root@xuegod63 ~]# kubectl get pods -n jenkins-k8s
NAME                                READY   STATUS             RESTARTS   AGE
jenkins-74b4c59549-qvmrt           0/1     CrashLoopBackOff   1           15s
#上面可以看到 CrashLoopBackOff, 解决方法如下:
#查看 jenkins-74b4c59549-qvmrt 日志
[root@xuegod63 ~]# kubectl logs jenkins-74b4c59549-qvmrt -n jenkins-k8s
touch: cannot touch '/var/jenkins_home/copy_reference_file.log': Permission denied
#上面问题是因为/data/v1 目录权限问题, 按如下方法解决:
[root@xuegod63 ~]# chown -R 1000.1000 /data/v1/
[root@xuegod63 ~]# kubectl delete -f jenkins-deployment.yaml
[root@xuegod63 ~]# kubectl apply -f jenkins-deployment.yaml
[root@xuegod63 ~]# kubectl get pods -n jenkins-k8s
NAME                                READY   STATUS    RESTARTS   AGE
jenkins-74b4c59549-gp95l           1/1     Running   0           16s
(7) 把jenkins 前端加上 service, 提供外部网络访问
[root@xuegod63 ~]# cat jenkins-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: jenkins-service
```

```
namespace: jenkins-k8s
labels:
  app: jenkins
spec:
  selector:
    app: jenkins
  type: NodePort
  ports:
    - name: web
      port: 8080
      targetPort: web
      nodePort: 30002
    - name: agent
      port: 50000
      targetPort: agent
```

#更新资源清单文件

```
[root@xuegod63 ~]# kubectl apply -f jenkins-service.yaml
```

```
[root@xuegod63 ~]# kubectl get svc -n jenkins-k8s
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
jenkins-service	NodePort	10.110.34.67	<none>	8080:30002/TCP,50000:32331/TCP

#### 4、配置 Jenkins

在浏览器访问 jenkins 的 web 界面:

<http://192.168.1.63:30002/login?from=%2F>

入门

## 解锁 Jenkins

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

```
/var/jenkins_home/secrets/initialAdminPassword
```

请从本地复制密码并粘贴到下面。

管理员密码

#### 1) 获取管理员密码

在 nfs 服务端，也就是我们的 xuegod63 节点获取密码:

```
[root@xuegod63 ~]# cat /data/v1/jenkins-home/secrets/initialAdminPassword
f9b0b4400c4a4d6eae6762616db6d63
```

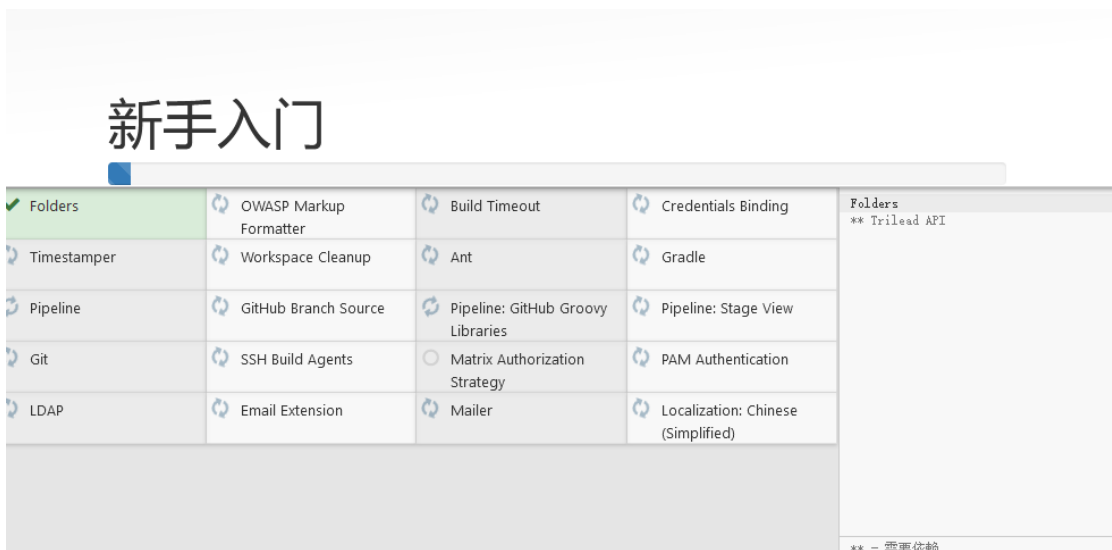
把获取到的密码拷贝到上面管理员密码下的方框里

点击继续, 出现如下界面



## 2) 安装插件

### 安装推荐的插件



离线安装 jenkins 插件:

<https://plugins.jenkins.io/>

插件安装好之后显示如下



The screenshot shows the 'Create the first administrator user' form in Jenkins. It has a blue header with the title. Below the title are five input fields: '用户名:' (Username) with a yellow background, '密码:' (Password) with a yellow background and masked characters, '确认密码:' (Confirm Password), '全名:' (Full Name), and '电子邮件地址:' (Email Address). All fields are currently empty.

### 3) 创建第一个管理员用户



This screenshot shows the same user creation form, but with the following values entered: '用户名:' is 'admin', '密码:' is masked with six dots, '确认密码:' is masked with six dots, '全名:' is 'admin', and '电子邮件地址:' is 'admin@163.com'.

用户名和密码都设置成 admin，线上环境需要设置成复杂的密码

修改好之后点击保存并完成，出现如下界面

点击保存并完成，出现如下界面



The screenshot shows the 'Instance Configuration' page. At the top is the title '实例配置'. Below it is a text input field for 'Jenkins URL:' containing 'http://192.168.80.199:30002/'. Below the input field is a paragraph of text explaining the purpose of the URL and providing instructions on how to set it correctly for various features like email notifications and PR status.

点击保存并完成，出现如下界面



The screenshot shows the 'Jenkins is ready!' completion screen. It features a large heading 'Jenkins已就绪!' and a sub-heading 'Jenkins安装已完成。'. At the bottom is a blue button labeled '开始使用Jenkins'.

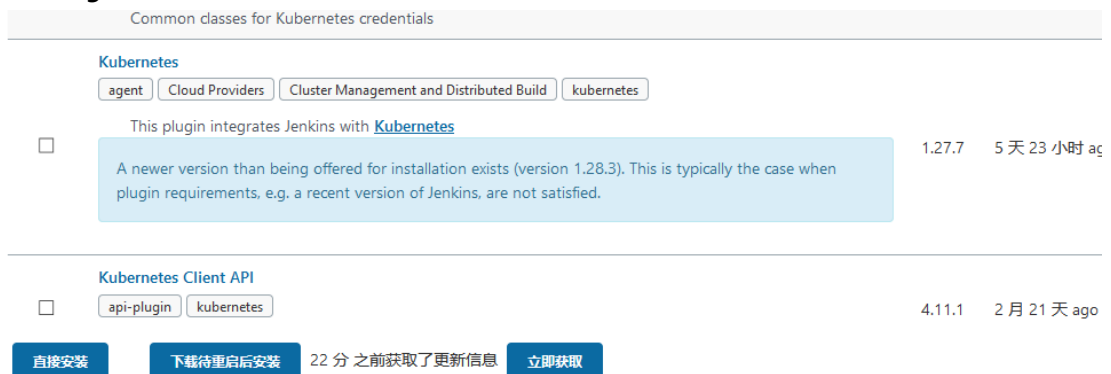
点击开始使用 Jenkins

## 5、测试 jenkins 的 CI/CD

#在 Jenkins 中安装 kubernetes 插件

### (1) 在 jenkins 中安装 k8s 插件

Manage Jnekins----->插件管理----->可选插件----->搜索 kubernetes----->出现如下

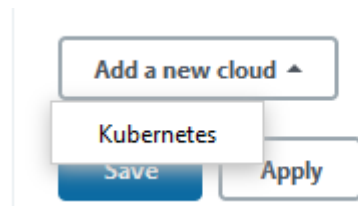


选中 kubernetes 之后----->点击下面的直接安装----->安装之后选择重新启动 jenkins--->  
<http://192.168.1.63:30002/restart>-->重启之后登陆 jenkins 即可

### #配置 jenkins 连接到我们存在的 k8s 集群

(1) 访问 <http://192.168.1.63:30002/configureClouds/>

新增一个云,在下拉菜单中选择 kubernetes 并添加



### (2) 填写云 kubernetes 配置内容

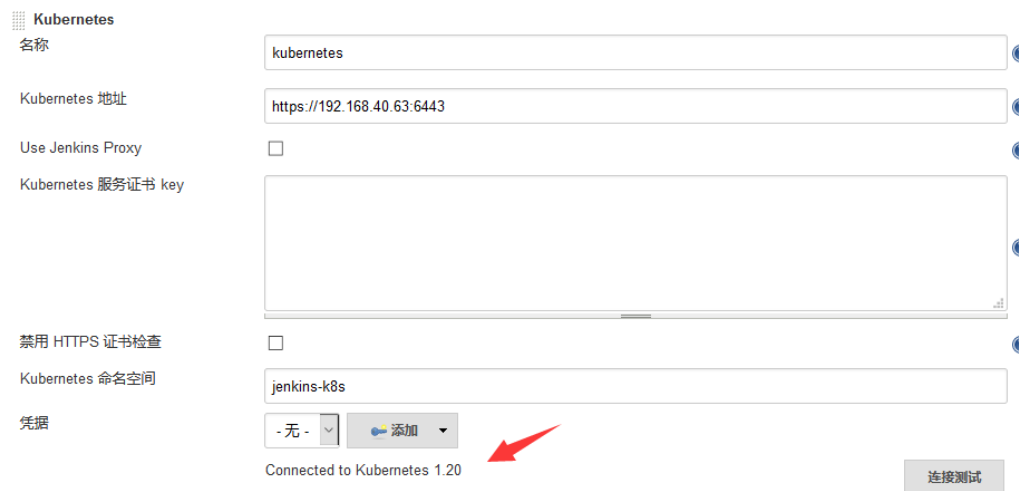
Kubernetes	
名称	<input type="text" value="kubernetes"/>
Kubernetes 地址	<input type="text" value="https://192.168.40.63:6443"/>

kubernetes

<https://192.168.1.63:6443>

### (3) 测试 jenkins 和 k8s 是否可以通信

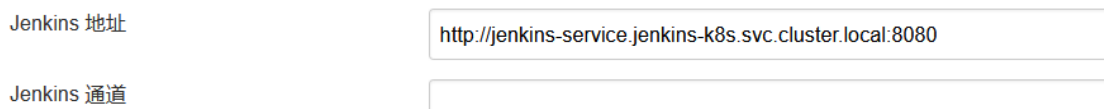




The image shows the Jenkins configuration page for a Kubernetes provider. The left sidebar has a 'Kubernetes' icon. The main form includes fields for '名称' (Name) set to 'kubernetes', 'Kubernetes 地址' (Address) set to 'https://192.168.40.63:6443', 'Use Jenkins Proxy' (unchecked), 'Kubernetes 服务证书 key' (empty text area), '禁用 HTTPS 证书检查' (unchecked), 'Kubernetes 命名空间' (Namespace) set to 'jenkins-k8s', and '凭据' (Credentials) set to '- 无 -'. A red arrow points to the '添加' (Add) button next to the credentials field. At the bottom, it says 'Connected to Kubernetes 1.20' and has a '连接测试' (Test Connection) button.

点击连接测试, 如果显示 **Connection test successful** 或者显示 **Connected to Kubernetes 1.20**

说明测试成功, Jenkins 可以和 k8s 进行通信



The image shows the 'Jenkins 地址' (Jenkins Address) field set to 'http://jenkins-service.jenkins-k8s.svc.cluster.local:8080'. Below it is the 'Jenkins 通道' (Jenkins Channel) field, which is currently empty.

配置 k8s 集群的时候 jenkins 地址需要写上面域名的形式, 配置之后执行如下:

**http://jenkins-service.jenkins-k8s.svc.cluster.local:8080**

应用----->保存

#配置 pod-template

(1) 配置 pod template

访问 <http://192.168.1.63:30002/configureClouds/>

添加 Pod 模板----->Kubernetes Pod Template--->按如下配置

Pod Templates

Pod Template

名称

test

命名空间

jenkins-k8s

标签列表

testhan

用法

只允许运行绑定到这台机器的Job

父级的 Pod 模板名称

容器列表

添加容器

环境变量

Pod 代理中的容器列表

添加环境变量

卷

该 Pod 中所有容器的环境变量

添加卷

注解

挂载到 Pod 代理中的卷列表

添加注解

## (2) 在上面的 pod template 下添加容器

添加容器----->Container Template----->按如下配置----->

Pod Template

名称

test

命名空间

jenkins-k8s

标签列表

testhan

用法

只允许运行绑定到这台机器的Job

父级的 Pod 模板名称

容器列表

Container Template

名称

jnlp

Docker 镜像

xuegod/jenkins-jnlp:v1

总是拉取镜像

☐

工作目录

/home/jenkins/agent

运行的命令

命令参数

分配伪终端

☒

Environment Variables

添加环境变量

设置到 Pod 节点中的环境变量列表

Docker 镜像: 使用 jenkins-jnlp.tar.gz 解压出来的镜像, 把这个镜像上传到 k8s 的各 node 节点, 手动解压: docker load -i jenkins-jnlp.tar.gz

解压出来的镜像是 xuegod/jenkins-jnlp:v1

在每一个 pod template 右下角都有一个 Advanced, 点击 Advanced, 出现如下

Service Account

Image pull secrets

jenkins-k8s-sa

在 Service Account 处输入 jenkins-k8s-sa, 这个 sa 就是我们最开始安装 jenkins 时的 sa

(3) 给上面的 pod template 添加卷

添加卷----->选择 Host Path Volume

卷

Host Path Volume

主机路径 /var/run/docker.sock

挂载路径 /var/run/docker.sock

删除卷

Host Path Volume

主机路径 /root/.kube

挂载路径 /home/jenkins/.kube

删除卷

/var/run/docker.sock

/var/run/docker.sock

/root/.kube

/home/jenkins/.kube

上面配置好之后, 应用----->保存

#添加 dockerhub 凭据

注意: 下面需要用到 dockerhub 存放镜像, 大家如果没有 dockerhub 可以自己申请一个

首页----->系统管理----->管理凭据-->全局-->添加凭据

Jenkins > 凭据

新建任务  
用户列表  
构建历史  
系统管理  
我的视图  
Lockable Resources  
新建视图

构建队列  
队列中没有构建任务

构建执行状态

凭据

类型	提供者	存储	域
图标: 小 中 大			

Stores scoped to Jenkins

提供者	存储
Jenkins	

添加凭据

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

[返回到凭据域列表](#) [添加凭据](#)

类型: Username with password

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

用户名: xuegod

密码: .....



ID: dockerhub

描述: jenkins dockerhub

[确定](#)

## 全局凭据 (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	名称	类型	描述
 <a href="#">dockerhub</a>	xuegod/***** (jenkins dockerhub)	Username with password	jenkins dockerhub 

图标: [小](#) [中](#) [大](#)

username: xuegod

password: 1989\*\*\*\*\*

ID: dockerhub

描述: 这个地方随便写

上面修改好之后选择确定即可

#测试通过 Jenkins 发布代码到 k8s 开发环境、测试环境、生产环境

在 k8s 的控制节点创建名称空间:

```
[root@xuegod63 ~]# kubectl create ns devlopment
namespace/devlopment created
```

```
[root@xuegod63 ~]# kubectl create ns production
namespace/production created
```

```
[root@xuegod63 ~]# kubectl create ns qatest
namespace/qatest created
```

回到 jenkins 首页:

新建一个任务----->输入一个任务名称处输入 jenkins-variable-test-deploy----->流水线-----

->确定----->在 Pipeline script 处输入如下内容

```
node('testhan') {
    stage('Clone') {
        echo "1.Clone Stage"
        git url: "https://github.com/luckylucky421/jenkins-sample.git"
        script {
            build_tag = sh(returnStdout: true, script: 'git rev-parse --short
HEAD').trim()
        }
    }
}
```

```
}
stage('Test') {
    echo "2.Test Stage"
}

stage('Build') {
    echo "3.Build Docker Image Stage"
    sh "docker build -t xuegod/jenkins-demo:${build_tag} ."
    //此处标签更改为自己的 dockerhub 的 username
}

stage('Push') {
    echo "4.Push Docker Image Stage"
    withCredentials([usernamePassword(credentialsId: 'dockerhub',
passwordVariable: 'dockerHubPassword', usernameVariable: 'dockerHubUser')]) {
        sh "docker login -u ${dockerHubUser} -p ${dockerHubPassword}"
        sh "docker push xuegod/jenkins-demo:${build_tag}"
        //此处标签更改为自己的 dockerhub 的 username
    }
}

stage('Deploy to dev') {
    echo "5. Deploy DEV"
    sh "sed -i 's/<BUILD_TAG>/${build_tag}/' k8s-dev-xuegod.yaml"
    sh "sed -i 's/<BRANCH_NAME>/${env.BRANCH_NAME}/' k8s-dev-xuegod.yaml"
    // sh "bash running-development.sh"
    sh "kubectl apply -f k8s-dev-xuegod.yaml --validate=false"
}

stage('Promote to qa') {
    def userInput = input(
        id: 'userInput',

        message: 'Promote to qa?',
        parameters: [
            [
                $class: 'ChoiceParameterDefinition',
                choices: "YES\nNO",
                name: 'Env'
            ]
        ]
    )
    echo "This is a deploy step to ${userInput}"
    if (userInput == "YES") {
        sh "sed -i 's/<BUILD_TAG>/${build_tag}/' k8s-qa-xuegod.yaml"
        sh "sed -i 's/<BRANCH_NAME>/${env.BRANCH_NAME}/' k8s-qa-xuegod.yaml"
        // sh "bash running-qa.sh"
```


```
        sh "kubectl apply -f k8s-qa-xuegod.yaml --validate=false"
        sh "sleep 6"
        sh "kubectl get pods -n qatest"
    } else {
        //exit
    }
}
stage('Promote to pro') {
    def userInput = input(

        id: 'userInput',
        message: 'Promote to pro?',
        parameters: [
            [
                $class: 'ChoiceParameterDefinition',
                choices: "YES\nNO",
                name: 'Env'
            ]
        ]
    )
    echo "This is a deploy step to ${userInput}"
    if (userInput == "YES") {
        sh "sed -i 's/<BUILD_TAG>/${build_tag}/' k8s-prod-xuegod.yaml"
        sh "sed -i 's/<BRANCH_NAME>/${env.BRANCH_NAME}/' k8s-prod-
xuegod.yaml"
        //        sh "bash running-production.sh"
        sh "cat k8s-prod-xuegod.yaml"
        sh "kubectl apply -f k8s-prod-xuegod.yaml --record --validate=false"
    }
}
}
```

注意:

k8s-dev-xuegod.yaml

```
14         app: jenkins-demo
15     spec:
16         containers:
17         - image: xuegod/jenkins-demo:<BUILD_TAG>
18           imagePullPolicy: IfNotPresent
19         name: jenkins-demo
```



xuegod 变成自己的 dockerhub 用户名

k8s-qa-xuegod.yaml

```
14     app: jenkins-demo
15     spec:
16       containers:
17         - image: xuegod/jenkins-demo:<BUILD_TAG>
18           imagePullPolicy: IfNotPresent
19           name: jenkins-demo
```

xuegod 变成自己的 dockerhub 用户名

#### k8s-prod-xuegod.yaml

```
14     app: jenkins-demo
15     spec:
16       containers:
17         - image: xuegod/jenkins-demo:<BUILD_TAG>
18           imagePullPolicy: IfNotPresent
19           name: jenkins-demo
```

xuegod 变成自己的 dockerhub 用户名

应用----->保存----->立即构建

在左侧可以看到构建任务，如下所示：



Jenkins > jenkins-variable-test-deploy >

返回工作台  
状态  
变更历史  
立即构建  
删除 Pipeline  
配置  
完整阶段视图  
重命名  
流水线语法

### Pipeline jenkins-var

最近变更

### 阶段视图

Average stage times:

Stage	Duration	Changes
#2	Apr 05 13:46	No Changes
#1	Apr 05 13:45	No Changes

### Build History

构建历史

find

#	Time	Status
#2	2021-4-5 上午5:46	成功
#1	2021-4-5 上午5:45	失败

变更历史  
Console Output  
编辑构建信息  
Git Build Data  
Thread Dump

### 相关链接

- 最近一次构建(#2), 17 秒之前
- 最近未成功的构建(#2), 17 秒之前

点击 Console Output, 可以看到 pipeline 定义的步骤的详细信息

Jenkins > jenkins-variable-test-deploy > #1

```
93448d8c2605: Preparing
c54f8a17910a: Preparing
df64d3292fd6: Preparing
c54f8a17910a: Waiting
df64d3292fd6: Waiting
ac1c7fa88ed0: Layer already exists
93448d8c2605: Layer already exists
cc5fec2c1edc: Layer already exists
df64d3292fd6: Layer already exists
c54f8a17910a: Layer already exists
314d7e212ed7: Pushed
d9f0b9768884: Pushed
ce7bf6b: digest: sha256:30930f173b67daa63ab387f3457c047981c5f224271121a26e7cf083638c3c39 size:
[Pipeline]
[Pipeline] // withCredentials
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy to dev)
[Pipeline] echo
[Pipeline] echo
5. Deploy DEV
[Pipeline] sh
+ sed -i s/<BUILD_TAG>/ce7bf6b/ k8s-dev.yaml
[Pipeline] sh
+ sed -i s/<BRANCH_NAME>/null/ k8s-dev.yaml
[Pipeline] sh
+ kubectl apply -f k8s-dev.yaml --validate=false
deployment.apps/jenkins-demo created
service/demo created
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Promote to qa)
[Pipeline] input
[Pipeline] input requested
%`
```


点击 Input request, 部署应用到预生产环境

### Promote to qa?

Env YES

继续 终止



```
NAME READY STATUS RESTARTS AG
jenkins-demo-784885d9c9-59tsm 1/1 Running 0 7s
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Promote to pro)
[Pipeline] input
Input requested

```


点击 Input request, 部署应用到生产环境

## Promote to pro?

Env

继续

终止

```
- image: xianchao/jenkins-demo:ce7bf6b
  imagePullPolicy: IfNotPresent
  name: jenkins-demo
  env:
    - name: branch
      value: null
---
apiVersion: v1
kind: Service
metadata:
  name: jenkins-demo
  namespace: production
spec:
  selector:
    app: jenkins-demo
  type: NodePort
  ports:
    - port: 18888
      targetPort: 18888
      nodePort: 31890
[Pipeline] sh
+ kubectl apply -f k8s-prod.yaml --record --validate=false
deployment.apps/jenkins-demo created
service/jenkins-demo created
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

看到 Finished: SUCCESS,, 说明 pipeline 构建任务完成

### 报错 1

```
Login Succeeded
[Pipeline] sh
+ docker push xuegod/jenkins-demo:7ff1569
The push refers to repository [docker.io/xuegod/jenkins-demo]
8d0c43ace4d0: Preparing
ac919cbea595: Preparing
ac1c7fa88ed0: Preparing
cc5fec2c1edc: Preparing
93448d8c2605: Preparing
c54f8a17910a: Preparing
df64d3292fd6: Preparing
c54f8a17910a: Waiting
df64d3292fd6: Waiting
denied: requested access to the resource is denied
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 1
Finished: FAILURE
```

**解决方法: 请修改 Pipeline script 处标签更改为自己的 dockerhub 的 username。**

#### 6、验证开发、预生产、生产环境部署的应用是否正常运行

[root@xuegod63 ~]# kubectl get pods -n development

NAME	READY	STATUS	RESTARTS	AGE
jenkins-demo-784885d9c9-b4jkg	1/1	Running	0	6m18s

[root@xuegod63 ~]# kubectl get pods -n qatest

NAME	READY	STATUS	RESTARTS	AGE
jenkins-demo-784885d9c9-59tsm	1/1	Running	0	5m19s

[root@xuegod63 ~]# kubectl get pods -n production

NAME	READY	STATUS	RESTARTS	AGE
jenkins-demo-784885d9c9-42hz4	1/1	running	0	5m38s

#### 实战 2: Jenkins 管理插件 BlueOcean 配置和使用

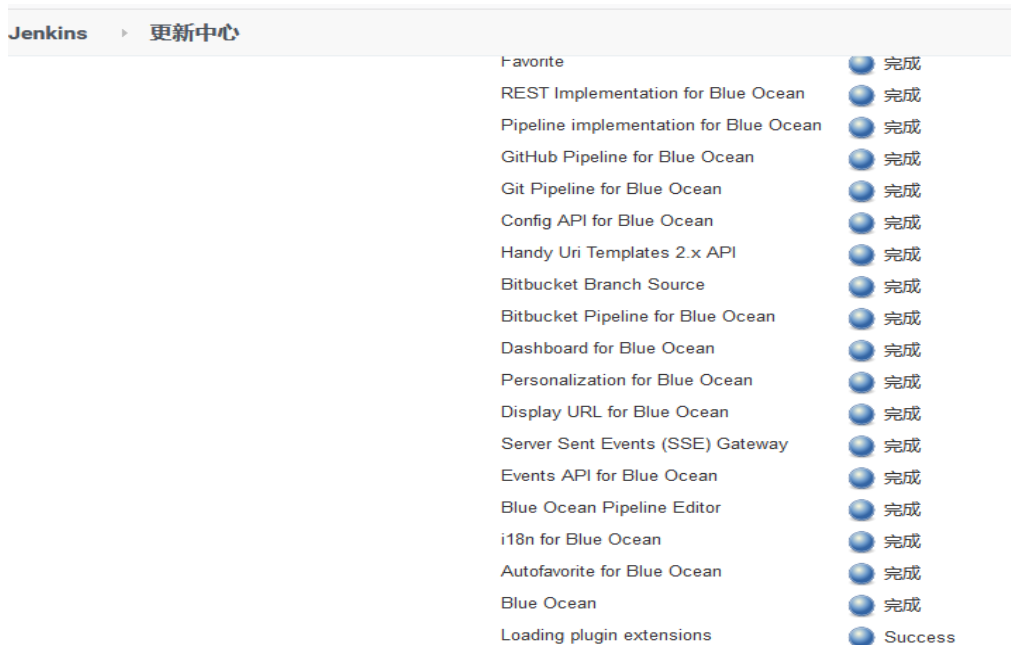
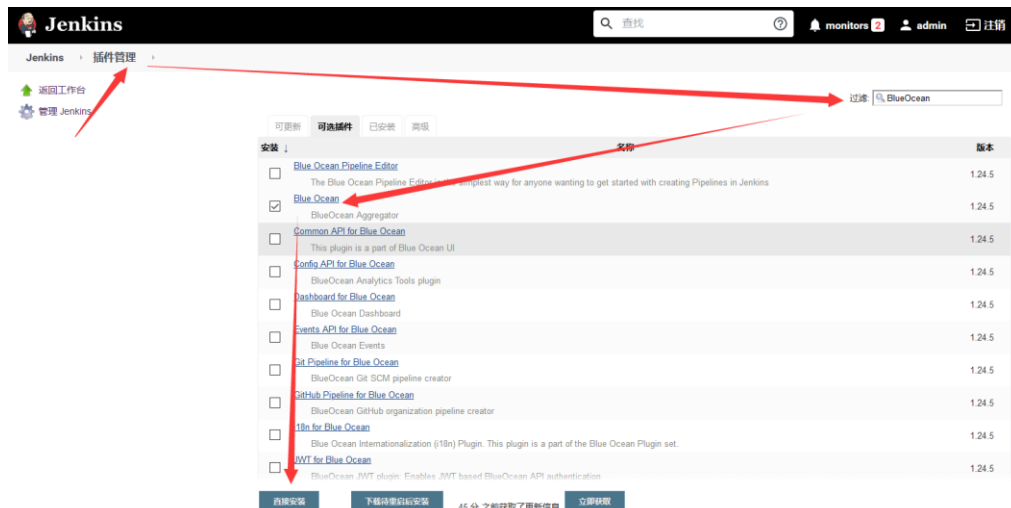
##### # BlueOcean 是什么?

为了适应 Jenkins Pipeline 和 Freestyle jobs 任务, Jenkins 推出了 BlueOcean UI, 其目的就是让程序员执行任务时, 降低工作流程的复杂度和提升工作流程的清晰度, 它具有如下特征:

- 1、清晰的可视化: 对 CI/CD pipelines, 可以快速直观的观察项目 pipeline 状态。
- 2、pipeline 可编辑: 引导用户通过直观的、可视化的过程来创建 Pipeline, 从而使 Pipeline 的创建变得平易近人。
- 3、pipeline 精确度: 通过 UI 直接介入 pipeline 的中间问题。

##### #安装 BlueOcean 插件

主页—>系统管理—>插件管理—>可选插件—>过滤 BlueOcean—>直接安装



安装完成后, 重启 jenkins

<http://192.168.1.63:30002/restart>

你确定要重启 Jenkins 吗?

是

重启进入 jenkins 之后, 在主页左侧可以看到打开 Blue Ocean

 **Jenkins**

**Jenkins** >

 新建任务

 用户列表

 构建历史

 项目关系

 检查文件指纹

 系统管理

 我的视图

 打开 Blue Ocean 

 Lockable Resources

 新建视图


**构建队列** 


队列中没有构建任务


**构建执行状态** 


1 空闲


2 空闲




**Jenkins** **流水线** **配置管理**  **注销**



**流水线**  Search pipelines... **创建流水线**





名称	健康状态	分支	PR
jenkins-variable-test-deploy		-	-




**Jenkins** **流水线** **配置管理**  **注销**

 jenkins-variable-test-deploy   **活动** **分支** **Pull Requests**

 **运行**  **Disable**

状态	运行	提交	消息	持续时间	完成
	2	-	由用户 admin 启动	9s	20 minutes ago 
	1	-	由用户 admin 启动	6m 0s	15 minutes ago 



jenkins-variable-test-deploy 1 >

流水线 改变 测试 制品 注册

分支: 6m 0s 没有修改  
提交: 16 minutes ago 由用户 admin 启动

Start Clone Test Build Push Deploy to dev Promote to qa Promote to pro End

Promote to pro - 1m 17s

> 等待交互式输入 1m 15s

> This is a deploy step to YES - Print Message <1s

> sed -i 's/<BUILD\_TAG>/ce7bf6b/' k8s-prod.yaml - Shell Script <1s

> sed -i 's/<BRANCH\_NAME>/null/' k8s-prod.yaml - Shell Script <1s

> cat k8s-prod.yaml - Shell Script <1s

> kubectl apply -f k8s-prod.yaml --record --validate=false - Shell Script <1s

Start Clone Test Build Push Deploy to dev Promote to qa Promote to pro End

第一步克隆代码

Clone - 16s

1.Clone Stage - Print Message <1s

1.Clone Stage

https://github.com/lucky421/jenkins-sample.git - Git 13s

```
1 The recommended git tool is: NONE
2 No credentials specified
3 Cloning the remote git repository
4 Cloning repository https://github.com/lucky421/jenkins-sample.git
5 > git init /home/jenkins/workspace/jenkins-harbor # timeout=10
6 Fetching upstream changes from https://github.com/lucky421/jenkins-sample.git
7 > git --version # timeout=10
8 > git fetch --tags --progress -- https://github.com/lucky421/jenkins-sample.git +refs/heads/*:refs/remotes/origin/* # timeout=10
9 Avoid second fetch
10 Checking out revision ce7bf6bde7eb643015c39c886a639945ec4db (refs/remotes/origin/master)
11 > git config remote.origin.url https://github.com/lucky421/jenkins-sample.git # timeout=10
12 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
13 > git rev-parse refs/remotes/origin/master:commit # timeout=10
14 > git config core.sparsecheckout # timeout=10
15 > git checkout -f ce7bf6bde7eb643015c39c886a639945ec4db # timeout=10
16 > git branch -a -v --no-above # timeout=10
17 > git checkout -b master ce7bf6bde7eb643015c39c886a639945ec4db # timeout=10
18 Commit message: 'update k8s-prod.yaml'
19 > git rev-list --no-walk ce7bf6bde7eb643015c39c886a639945ec4db # timeout=10
20 > git rev-parse --short HEAD - Shell Script 2s
21 + git rev-parse --short HEAD
```

Start Clone Test Build Push Deploy to dev Promote to qa Promote to pro End

第二步测试阶段

Test - <1s

2.Test Stage - Print Message

2.Test Stage

Start Clone Test Build Push Deploy to dev Promote to qa Promote to pro End

第三步构建镜像

Build - 4s

3.Build Docker Image Stage - Print Message

3.build docker image Stage

docker build -t 192.168.40.132/jenkins-demo/jenkins-demo:ce7bf6b - Shell Script

```
1 + docker build -t 192.168.40.132/jenkins-demo/jenkins-demo:ce7bf6b .
2 Sending build context to Docker daemon 166.9kB
3 Step 1/5 : FROM golang:1.10.4-alpine
4 ----> b36365650f3
5 Step 2/5 : ADD . /go/src/app
6 ----> 8f7d595db02
7 Step 3/5 : WORKDIR /go/src/app
8 ----> Running in 653b8cc0a06
9 Removing intermediate container 653b8cc0a06
10 ----> 280e8b9e8cc
11 Step 4/5 : RUN go build -v -o /go/src/app/jenkins-app
12 ----> Running in 428150891508
13 app
14 Removing intermediate container 428150891508
15 ----> 7bf6c8f361a
16 Step 5/5 : CMD ["./jenkins-app"]
17 ----> Running in 48d1fabad16d
18 Removing intermediate container 48d1fabad16d
19 ----> 6c9519174bf7
20 Successfully built 6c9519174bf7
21 Successfully tagged 192.168.40.132/jenkins-demo/jenkins-demo:ce7bf6b
```

学神 IT 教育官方 QQ 群: 957231097 或唐老师 QQ: 3340273106 领取更多资料



Push - 4s

```
✓ 4. Push Docker Image Stage -- Print Message
1 4. Push Docker Image Stage
✓ docker login 192.168.40.132 -u ${dockerHubUser} -p ${dockerHubPassword} -- Shell Script
1 Warning: A secret was passed to "sh" using groovy String interpolation, which is insecure.
2 Affected argument(s) used the following variable(s): [dockerHubUser, dockerHubPassword]
3 See https://jenkins.io/redirect/groovy-string-interpolation for details.
4 + docker login 192.168.40.132 -u **** -p ****
5 WARNING! Using --password via the CLI is insecure. Use --password-stdin.
6 WARNING! Your password will be stored unencrypted in /home/jenkins/.docker/config.json.
7 Configure a credential helper to remove this warning: see
8 https://docs.docker.com/engine/reference/commandline/login/#credentials-store
9 Login Succeeded
10
✓ docker push 192.168.40.132/jenkins-demo/jenkins-demo:ce7bf6b -- Shell Script
1 + docker push 192.168.40.132/jenkins-demo/jenkins-demo:ce7bf6b
2 The push refers to repository [192.168.40.132/jenkins-demo/jenkins-demo]
3 edc20475336f: Preparing
4 12be6ceef7d: Preparing
5 ac1c7fa88ed9: Preparing
6 cc3fec1c1cd: Preparing
7 93448dc1cd8: Preparing
8 c54fa81793ba: Preparing
9 df64d202fde: Preparing
10 93448dc1cd8: Waiting
11 c54fa81793ba: Waiting
12 df64d202fde: Waiting
13 cc3fec1c1cd: Waiting
14 ac1c7fa88ed9: Waiting
15 12be6ceef7d: Pushed
16 ac1c7fa88ed9: Layer already exists
17 cc3fec1c1cd: Layer already exists
18 93448dc1cd8: Layer already exists
19 c54fa81793ba: Layer already exists
20 df64d202fde: Layer already exists
21 edc20475336f: Pushed
22 ce7bf6b: digest: sha256:c8cf5e895cf08096c9abe21d0daa8bde4e4291677799ed5ae411fb2554ffa size: 1764
```



Deploy to dev - 1s

```
✓ 5. Deploy DEV -- Print Message
1 5. Deploy DEV
✓ sed -i s/'<BUILD_TAG>/ce7bf6b/' k8s-dev-harbor.yaml -- Shell Script
1 + sed -i s/'<BUILD_TAG>/ce7bf6b/' k8s-dev-harbor.yaml
✓ sed -i s/'<BRANCH_NAME>/null/' k8s-dev-harbor.yaml -- Shell Script
1 + sed -i s/'<BRANCH_NAME>/null/' k8s-dev-harbor.yaml
✓ kubectl apply -f k8s-dev-harbor.yaml --validate=false -- Shell Script
1 + kubectl apply -f k8s-dev-harbor.yaml --validate=false
2 deployment.apps/jenkins-demo configured
3 service/demo unchanged
```



Promote to qa - 12s

```
✓ 等待交互式输入
1 Input requested
2 Approved by admin
✓ This is a deploy step to YES -- Print Message
1 This is a deploy step to YES
✓ sed -i s/'<BUILD_TAG>/ce7bf6b/' k8s-qa-harbor.yaml -- Shell Script
✓ sed -i s/'<BRANCH_NAME>/null/' k8s-qa-harbor.yaml -- Shell Script
✓ kubectl apply -f k8s-qa-harbor.yaml --validate=false -- Shell Script
✓ sleep 6 -- Shell Script
✓ kubectl get pods -n qatest -- Shell Script
```



第八步: 部署应用到生产环境

Promote to pro - 8s

✓ 等待交互式输入

1

Input requested  
approved by admin

✓ > This is a deploy step to YES — Print Message

✓ > sed -i 's/<BUILD\_TAG>/ce7bf6b/' k8s-prod-harbor.yaml — Shell Script

✓ > sed -i 's/<BRANCH\_NAME>/null/' k8s-prod-harbor.yaml — Shell Script

✓ > cat k8s-prod-harbor.yaml — Shell Script

✓ > kubectl apply -f k8s-prod-harbor.yaml --record --validate=false — Shell Script



Promote to pro - 12s

⏸ 等待交互式输入

Promote to pro?

☒ YES

☐ NO

继续

终止



Push - 4s

✓ > 4.Push Docker Image Stage — Print Message

✓ > docker login 192.168.40.132 -u \${dockerHubUser} -p \${dockerHubPassword} — Shell Script

✓ > docker push 192.168.40.132/jenkins-demo/jenkins-demo:ce7bf6b — Shell Script

### 实战 3: Jenkins 实现 k8s 应用按照指定版本回滚

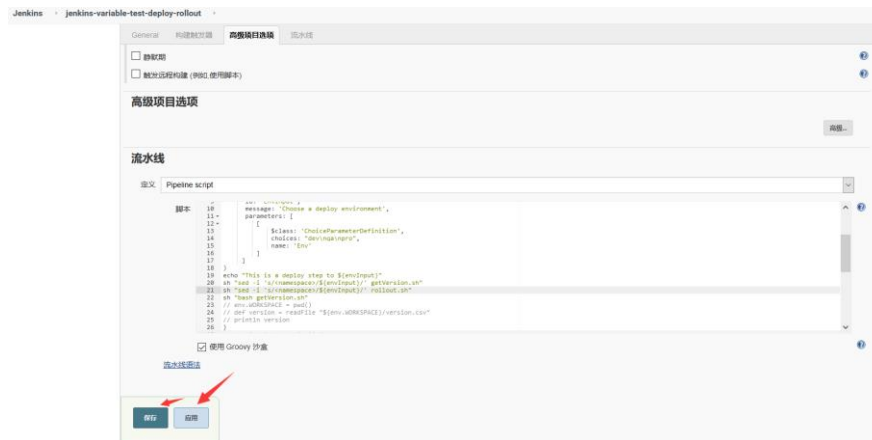
回到首页:

新建一个任务----->输入一个任务名称处输入 jenkins-variable-test-deploy-rollout----->流水线----->确定----->在 Pipeline script 处输入如下内容

```
node('testhan') {
  stage('git clone') {
    git url: "https://github.com/luckylucky421/jenkins-rollout"
    sh "ls -al"
    sh "pwd"
  }
  stage('select env') {
    def envInput = input(
      id: 'envInput',
      message: 'Choose a deploy environment',
      parameters: [
        [
          $class: 'ChoiceParameterDefinition',
          choices: "development\nqatest\nproduction",
          name: 'Env'
        ]
      ]
    )
    echo "This is a deploy step to ${envInput}"
    sh "sed -i 's/<namespace>/${envInput}/' getVersion.sh"
    sh "sed -i 's/<namespace>/${envInput}/' rollout.sh"
    sh "bash getVersion.sh"
    // env.WORKSPACE = pwd()
    // def version = readFile "${env.WORKSPACE}/version.csv"
    // println version
  }
  stage('select version') {
    env.WORKSPACE = pwd()
    def version = readFile "${env.WORKSPACE}/version.csv"
    println version
    def userInput = input(id: 'userInput',
                          message: '选择回滚版本',
                          parameters: [
                            [
                              $class: 'ChoiceParameterDefinition',
                              choices: "${version}\n",
                              name: 'Version'
                            ]
                          ]
    )
    sh "sed -i 's/<version>/${userInput}/' rollout.sh"
  }
  stage('rollout deploy') {
    sh "bash rollout.sh"
  }
}
```



```
}  
}
```



点击应用->保存-立即构建

Jenkins > jenkins-variable-test-deploy-rollout

- 返回工作台
- 状态
- 变更历史
- 立即构建
- 删除 Pipeline
- 配置
- 完整阶段视图
- 打开 Blue Ocean
- 重命名
- 流水线语法

### Pipeline jenkins-variable-test-deploy-rollout

最近变更

#### 阶段视图

	git clone	select env
Average stage times:	17s	513ms
#2 Apr 05 14:34 No Changes	17s	(paused for 2min 52s)
#1 Apr 05 14:28 No Changes		

#### 相关链接

- 最近一次构建(#2) 2 分 51 秒之前
- 最近失败的构建(#1) 8 分 56 秒之前
- 最近未成功的构建(#1) 8 分 56 秒之前
- 最近完成的构建(#1) 8 分 56 秒之前

#### Build History

构建历史

find X

2021-4-5 上午6:34

#2

#1

变更历史

Console Output

编辑构建信息

Git Build Data

需要输入而暂停

打开 Blue Ocean

Thread Dump

暂停 (恢复)

回站

Jenkinsjenkins-variable-test-deploy-rollout#2

```
[Pipeline] { (git clone)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/lucky421/jenkins-rollout
> git init /home/jenkins/agent/workspace/jenkins-variable-test-deploy-rollout # timeout=10
Fetching upstream changes from https://github.com/lucky421/jenkins-rollout
> git --version # timeout=10
> git --version # 'git version 2.11.0'
> git fetch --tags --progress -- https://github.com/lucky421/jenkins-rollout +refs/heads/*:refs/remotes/origin/*
Avoid second fetch
Checking out Revision e96ca52804b96206c021ef057e9a737d97896420 (refs/remotes/origin/master)
> git config remote.origin.url https://github.com/lucky421/jenkins-rollout # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git config core.sparsecheckout # timeout=10
> git checkout -f e96ca52804b96206c021ef057e9a737d97896420 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git checkout -b master e96ca52804b96206c021ef057e9a737d97896420 # timeout=10
Commit message: "Create rollout.sh"
First time build. Skipping changelog.
[Pipeline] sh
+ ls -al
total 8
drwxr-xr-x 3 root root 57 Apr  5 14:35 .
drwxr-xr-x 4 root root 98 Apr  5 14:35 ..
drwxr-xr-x 8 root root 162 Apr  5 14:35 .git
-rw-r--r-- 1 root root 136 Apr  5 14:35 getVersion.sh
-rw-r--r-- 1 root root 85 Apr  5 14:35 rollout.sh
[Pipeline] sh
+ pwd
/home/jenkins/agent/workspace/jenkins-variable-test-deploy-rollout
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (select env)
[Pipeline] input
Input requested
*

```

Jenkinsjenkins-variable-test-deploy-rollout#2需要输入而暂停

- 返回到项目
- 状态
- 变更历史
- Console Output

## Choose a deploy environment

Env dev

继续

终止

### 3.1 什么是 Tekton?

Tekton 是一个功能强大且灵活的 Kubernetes 原生开源框架,是谷歌开源的,功能强大且灵活,开源社区也正在快速的迭代和发展壮大,主要用于创建持续集成和交付(CI/CD)系统。通过抽象底层实现细节,用户可以跨多云平台和本地系统进行构建、测试和部署。另外,基于 kubernetes CRD 定义的 pipeline 流水线也是 Tekton 最重要的特征。

#### 扩展: CRD 是什么?

CRD 全称是 CustomResourceDefinition:

在 Kubernetes 中一切都可视为资源, Kubernetes 1.7 之后增加了对 CRD 自定义资源二次开发能力来扩展 Kubernetes API,通过 CRD 我们可以向 Kubernetes API 中增加新资源类型,而不需要修改 Kubernetes 源码来创建自定义的 API server,该功能大大提高了 Kubernetes 的扩展能力。当你创建一个新的 CustomResourceDefinition (CRD)时, Kubernetes API 服务器将为你指定的每个版本创建一个新的 RESTful 资源路径,我们可以根据该 api 路径来创建一些我们自定义的类型资源。CRD 可以是命名空间的,也可以是集群范围的,由 CRD 的作用域(scope)字段中所指定的,与现有的内置对象一样,删除名称空间将删除该名称空间中的所有自定义对象。customresourcedefinition 本身没有名称空间,所有名称空间都可以使用。

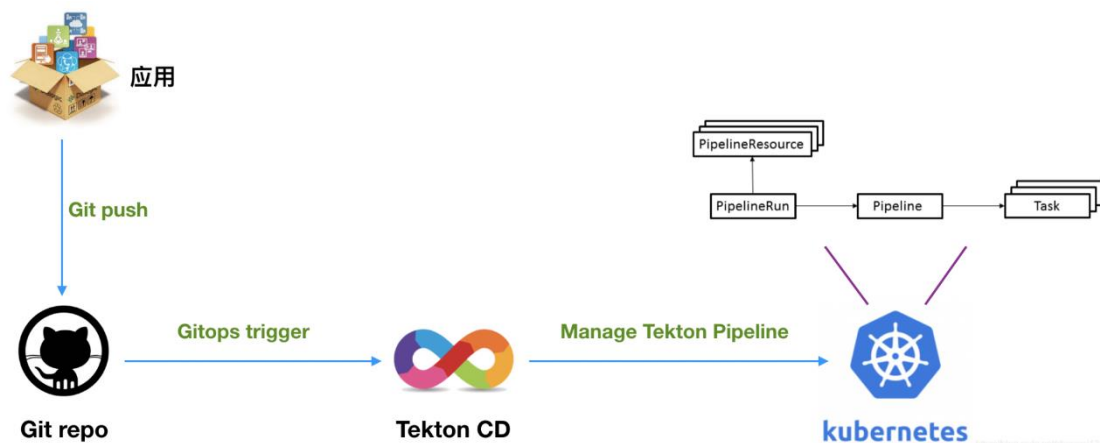
### 3.2 为什么要用 k8s 原生的 CI-CD 工具 Tekton?

持续集成是云原生应用的支柱技术之一，因此在交付基于云原生的一些支撑产品的时候，CICD 是一个无法拒绝的需求。为了满足这种需要，自然而然会想到对 Jenkins(X)或者 Gitlab 进行集成，也有创业公司出来的一些小工具比如 Argo Rollout。Tekton 是一款 k8s 原生的应用发布框架，主要用来构建 CI/CD 系统。它原本是 knative 项目里面一个叫做 build-pipeline 的子项目，用来作为 knative-build 的下一代引擎。然而，随着 k8s 社区里各种各样的需求涌入，这个子项目慢慢成长为一个通用的框架，能够提供灵活强大的能力去做基于 k8s 的构建发布。Tekton 其实只提供 Pipeline 这个一个功能，Pipeline 会被直接映射成 K8s Pod 等 API 资源。而比如应用发布过程的控制，灰度和上线策略，都是我们自己编写 K8s Controller 来实现的，也就意味着 Tekton 不会在 K8s 上盖一个“大帽子”，比如我们想看发布状态、日志等是直接通过 K8s 查看这个 Pipeline 对应的 Pod 的状态和日志，不需要再面对另外一个 API。

Tekton 功能:

1. Kubernetes 原生的 Tekton 的所有配置都是使用 CRD 方式进行编写存储的，非常易于检索和使用。
2. 配置和流程分离: Tekton 的 Pipeline 和配置可以分开编写，使用名称进行引用。
3. 轻量级核心的 Pipeline 非常轻便: 适合作为组件进行集成，另外也有周边的 Dashboard、Trigger、CLI 等工具，能够进一步挖掘其潜力。
4. 可复用、组合的 Pipeline 构建方式: 非常适合在集成过程中对 Pipeline 进行定制。

### 3.3 使用 Tekton 自动化发布应用流程



这里的流程大致是:

- 1、用户把需要部署的应用先按照一套标准的应用定义写成 YAML 文件 (类似 Helm Chart) ;
- 2、用户把应用定义 YAML 推送到 Git 仓库里;
- 3、Tekton CD (一个 K8s Operator) 会监听到相应的改动，根据不同条件生成不同的 Tekton Pipelines;

**Tekton CD 的操作具体分为以下几种情况:**

- 1、如果 Git 改动里有一个应用 YAML 且该应用不存在，那么将渲染和生成 Tekton Pipelines 用来创建应用。

- 2、如果 Git 改动里有一个应用 YAML 且该应用存在, 那么将渲染和生成 Tekton Pipelines 用来升级应用。这里我们会根据应用定义 YAML 里的策略来做升级, 比如做金丝雀发布、灰度升级。
- 3、如果 Git 改动里有一个应用 YAML 且该应用存在且标记了“被删除”, 那么将渲染和生成 Tekton Pipelines 用来删除应用。确认应用被删除后, 我们才从 Git 里删除这个应用的 YAML。

#### 实战 4: 安装 Tekton

```
#把 tekton-0-12-0.tar.gz 和 busybox-1-0.tar.gz 上传到 xuegod64 机器上, 手动解压:
docker load -i xuegod-tekton-0-12-0.tar.gz
docker load -i busybox-1-0.tar.gz
#编写安装 tekton 资源清单文件
[root@xuegod63 ~]# kubectl apply -f release.yaml
#验证 pod 是否创建成功
[root@xuegod63 ~]# kubectl get pods -n tekton-pipelines
```

NAME	READY	STATUS	RESTARTS	AGE
tekton-pipelines-controller-d64889fb9-9b68f	1/1	Running	0	52s
tekton-pipelines-webhook-7c8664944c-ctsgf	1/1	Running	0	52s

#### 3.4 Tekton 概念

Tekton 为 Kubernetes 提供了多种 CRD 资源对象, 可用于定义我们的流水线, 主要有以下几个 CRD 资源对象:

- 1) Task: 表示执行命令的一系列步骤, task 里可以定义一系列的 steps, 例如编译代码、构建镜像、推送镜像等, 每个 step 实际由一个 Pod 里的容器执行。
- 2) TaskRun: task 只是定义了一个模版, taskRun 才真正代表了一次实际的运行, 当然你也可以自己手动创建一个 taskRun, taskRun 创建出来之后, 就会自动触发 task 描述的构建任务。
- 3) Pipeline: 一组任务, 表示一个或多个 task、PipelineResource 以及各种定义参数的集合。
- 4) PipelineRun: 类似 task 和 taskRun 的关系, pipelineRun 也表示某一次实际运行的 pipeline, 下发一个 pipelineRun CRD 实例到 Kubernetes 后, 同样也会触发一次 pipeline 的构建。
- 5) PipelineResource: 表示 pipeline 输入资源, 比如 github 上的源码, 或者 pipeline 输出资源, 例如一个容器镜像或者构建生成的 jar 包等。

#### 实战 5: 测试 Tekton 构建 CI/CD 流水线

我们测试一个简单的 golang 程序。应用程序代码, 测试及 dockerfile 文件可在如下地址获取:

<https://github.com/luckylucky421/tekton-demo>

- 1、clone 应用程序代码进行测试, 创建一个 task 任务

```
[root@xuegod63 ~]# cat task-test.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: test
spec:
  resources:
```

```
inputs:
- name: repo
  type: git
steps:
- name: run-test
  image: golang:1.14-alpine
  workingDir: /workspace/repo
  command: ["go"]
  args: ["test"]
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f task-test.yaml
task.tekton.dev/test created
#查看 Task 资源
[root@xuegod63 ~]# kubectl get Task
NAME    AGE
test    23s
```

#上面内容解释说明:

resources 定义了我们的任务中定义的步骤中需要输入的内容, 这里我们的步骤需要 Clone 一个 Git 仓库作为 go test 命令的输入。Tekton 内置了一种 git 资源类型, 它会自动将代码仓库 Clone 到 /workspace/\$input\_name 目录中, 由于我们这里输入被命名成 repo, 所以代码会被 Clone 到 /workspace/repo 目录下面。然后下面的 steps 就是来定义执行运行测试命令的步骤, 这里我们直接在代码的根目录中运行 go test 命令即可, 需要注意的是命令和参数需要分别定义。

## 2、创建 pipelineresource 资源对象

通过上面步骤我们定义了一个 Task 任务, 但是该任务并不会立即执行, 我们必须创建一个 TaskRun 引用它并提供所有必需输入的数据才行。这里我们就需要将 git 代码库作为输入, 我们必须先创建一个 PipelineResource 对象来定义输入信息, 创建一个名为 pipelineresource.yaml 的资源清单文件, 内容如下所示:

```
[root@xuegod63 ~]# cat pipelineresource.yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineResource
metadata:
  name: xuegod-tekton-example
spec:
  type: git
  params:
    - name: url
      value: https://github.com/luckylucky421/tekton-demo
    - name: revision
      value: master
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f pipelineresource.yaml
```

## 3、创建 taskrun 任务

```
[root@xuegod63 ~]# cat taskrun.yaml
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: testrun
spec:
  taskRef:
    name: test
  resources:
    inputs:
      - name: repo
        resourceRef:
          name: xuegod-tekton-example
```

#更新资源清单文件

```
[root@xuegod63 ~]# kubectl apply -f taskrun.yaml
```

#上面资源清单文件解释说明

这里通过 taskRef 引用上面定义的 Task 和 git 仓库作为输入, resourceRef 也是引用上面定义的 PipelineResource 资源对象。

#创建后, 我们可以通过查看 TaskRun 资源对象的状态来查看构建状态

```
[root@xuegod63 ~]# kubectl get taskrun
```

NAME	SUCCEEDED	REASON	STARTTIME
testrun	Unknown	Running	6s

```
[root@xuegod63 ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
testrun-pod-x9rkn	2/2	Running	0	9s

当任务执行完成后, Pod 就会变成 Completed 状态了:

```
[root@xuegod63 ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
testrun-pod-x9rkn	0/2	Completed	0	72s

我们可以通过 kubectl describe 命令来查看任务运行的过程, 首先就是通过 initContainer 中的一个 busybox 镜像将代码 Clone 下来, 然后使用任务中定义的镜像来执行命令。当任务执行完成后, Pod 就会变成 Completed 状态了, 我们可以查看容器的日志信息来了解任务的执行结果信息:

```
[root@xuegod63 ~]# kubectl logs testrun-pod-x9rkn --all-containers
```

```
{"level":"info","ts":1617616592.58145,"caller":"git/git.go:136","msg":"Successfully cloned https://github.com/lucky421/tekton-demo @ c6c2a85091d538a13c44f85bcee9e861c362b0d3 (grafted, HEAD, origin/master) in path /workspace/repo"}
```

```
{"level":"info","ts":1617616592.6319332,"caller":"git/git.go:177","msg":"Successfully initialized and updated submodules in path /workspace/repo"}
```

PASS

```
ok _/workspace/repo 0.003s
```

#通过上面可以看到我们的测试已经通过了。

总结: 我们已经在 Kubernetes 集群上成功安装了 Tekton, 定义了一个 Task, 并通过 YAML 清单和创建 TaskRun 对其进行了测试。

## 总结:

实战 1: 基于 Jenkins+k8s+Git 等技术链构建企业级 DevOps 自动化容器云平台

实战 2: Jenkins 管理插件 BlueOcean 配置和使用

实战 3: Jenkins 实现 k8s 应用按照指定版本回滚

3.1 什么是 Tekton?

3.2 为什么要用 k8s 原生的 CI/CD 工具 Tekton?

3.3 Tekton 发布应用的流程

3.4 Tekton 概念

实战 4: 安装 Tekton

实战 5: 测试 Tekton 构建 CI/CD 流水线