持续集成：帮助开发人员更加频繁的将代码合并到共享分支或主干中，合并之后会自动触发构建应用，运行不同级别的代码扫描（sonarqube）和自动化测试（单元和集成测试）。

持续交付：将通过集成测试的代码合并到一个可以随时部署到生产环境的代码库。

持续部署：持续交付的延伸，就是将代码自动发布到生产环境中。

npm install

npm run build

1. 自动构建流水线
2. 无需构建，选择镜像发版

Jenkins、GitRunner。

Jenkins服务器需要安装git客户端

yum install git -y

Jenkins war包：<http://mirrors.jenkins.io/war-stable/>

java -jar jenkins.war --httpPort=28080

nohup java -jar jenkins.war --httpPort=28080 &

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

中文文档：<https://www.jenkins.io/zh/doc/book/pipeline/syntax/>

Jenkins Active Choice parameter：<https://plugins.jenkins.io/uno-choice/>

Gitlab 下载地址：<https://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el7/>

<https://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el8/>

阿里云镜像仓库：<https://cr.console.aliyun.com/cn-beijing/instances/namespaces>

阿里云客户端工具：

<https://help.aliyun.com/document_detail/121541.html>

[root@k8s-master02 ~]# aliyun configure

Configuring profile 'default' in 'AK' authenticate mode...

Access Key Id []: LTAI4G7pjYyJ7SnaUxWzfufy

Access Key Secret []: Zpx8OJQB0TuvzIuYNJYJHY6AijXh0K

Default Region Id []: cn-beijing

Default Output Format [json]: json (Only support json)

Default Language [zh|en] en: zh

Saving profile[default] ...Done.

Configure Done!!!

..............888888888888888888888 ........=8888888888888888888D=..............

...........88888888888888888888888 ..........D8888888888888888888888I...........

.........,8888888888888ZI: ...........................=Z88D8888888888D..........

.........+88888888 ..........................................88888888D..........

.........+88888888 .......Welcome to use Alibaba Cloud.......O8888888D..........

.........+88888888 ............. \*\*\*\*\*\*\*\*\*\*\*\*\* ..............O8888888D..........

.........+88888888 .... Command Line Interface(Reloaded) ....O8888888D..........

.........+88888888...........................................88888888D..........

..........D888888888888DO+. ..........................?ND888888888888D..........

...........O8888888888888888888888...........D8888888888888888888888=...........

............ .:D8888888888888888888.........78888888888888888888O ..............

[root@k8s-master02 ~]#

获取镜像TAG：aliyun cr GetRepoTags --RepoNamespace citools --RepoName docker | jq ".data.tags[].tag"

阿里云的命令空间：对应的就是harbor的repository

Repository下有镜像：

Docker:xxx

Kubectl:xxx

注意：使用BlueOcean创建Jenkinsfile时，任何步骤不能写中文

Kubectl set deployment deploy-name container-name=image-url -n namespace

测试k8s集群 test

Uat k8s集群 uat

生产k8s集群 prod

kubectl config use-context test/uat/prod

1. 代码仓库创建你们的项目
2. 开发去开发代码逻辑
3. Push到gitlab后执行构建
   1. 自动构建
      1. Env.gitlabBranch
   2. 手动构建
      1. BRANCH
   3. 定时构建
4. Jenkins调用k8s创建Pod执行构建
   1. 代码编译
   2. 代码扫描
5. 根据Dockerfile生成我们的镜像
   1. 放在对应项目的根目录下
      1. TAG
      2. Dockerfile –>项目根目录
      3. Harbor地址
      4. Harbor registry
      5. 应用名称
   2. 放在gitlab统一管理
   3. 每个job配置单独的变量
      1. Jar、war 🡪 基础镜像
      2. Html 🡪 html/
      3. . 🡪 工作目录 node server.js
         1. COPY 参数化
6. Push镜像到镜像仓库
7. Jenkins Slave kubectl 🡪 set 命令 更新我们的镜像
   1. 只更新镜像
   2. Helm更新
8. 判断程序是否启动
   1. -w
   2. 写脚本去判断
9. 程序启动后，调用测试Job

不构建的流水线：

1. Jenkins调用镜像仓库接口，返回镜像tag
2. 选择对于的tag进行发版到其他环境

agent {

kubernetes {

cloud 'kubernetes-default

slaveConnectTimeout 1200

yaml '''

apiVersion: v1

kind: Pod

spec:

containers:

- args: [\'$(JENKINS\_SECRET)\', \'$(JENKINS\_NAME)\']

image: 'registry.cn-beijing.aliyuncs.com/citools/jnlp:alpine'

name: jnlp

imagePullPolicy: IfNotPresent

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- command:

- "cat"

env:

- name: "LANGUAGE"

value: "en\_US:en"

- name: "LC\_ALL"

value: "en\_US.UTF-8"

- name: "LANG"

value: "en\_US.UTF-8"

image: "registry.cn-beijing.aliyuncs.com/citools/maven:3.5.3"

imagePullPolicy: "IfNotPresent"

name: "build"

tty: true

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/root/.m2/repository"

name: "volume-maven-repo"

readOnly: false

- command:

- "cat"

env:

- name: "LANGUAGE"

value: "en\_US:en"

- name: "LC\_ALL"

value: "en\_US.UTF-8"

- name: "LANG"

value: "en\_US.UTF-8"

image: "registry.cn-beijing.aliyuncs.com/citools/kubectl:1.17.4"

imagePullPolicy: "IfNotPresent"

name: "kubectl"

tty: true

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/var/run/docker.sock"

name: "volume-docker"

readOnly: false

- mountPath: "/root/.kube"

name: "volume-kubeconfig"

readOnly: false

- command:

- "cat"

env:

- name: "LANGUAGE"

value: "en\_US:en"

- name: "LC\_ALL"

value: "en\_US.UTF-8"

- name: "LANG"

value: "en\_US.UTF-8"

image: "registry.cn-beijing.aliyuncs.com/citools/docker:19.03.9-git"

imagePullPolicy: "IfNotPresent"

name: "docker"

tty: true

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/var/run/docker.sock"

name: "volume-docker"

readOnly: false

- mountPath: "/etc/hosts"

name: "volume-hosts"

readOnly: false

restartPolicy: "Never"

securityContext: {}

volumes:

- hostPath:

path: "/var/run/docker.sock"

name: "volume-docker"

- hostPath:

path: "/usr/share/zoneinfo/Asia/Shanghai"

name: "volume-2"

- hostPath:

path: "/etc/hosts"

name: "volume-hosts"

- name: "volume-maven-repo"

emptyDir: {}

- name: "volume-kubeconfig"

secret:

secretName: "multi-kube-config"

'''

}

}

pipeline {

agent {

kubernetes {

cloud 'kubernetes-default'

slaveConnectTimeout 1200

yaml '''

apiVersion: v1

kind: Pod

spec:

containers:

- args: [\'$(JENKINS\_SECRET)\', \'$(JENKINS\_NAME)\']

image: 'registry.cn-beijing.aliyuncs.com/citools/jnlp:alpine'

name: jnlp

imagePullPolicy: IfNotPresent

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/etc/hosts"

name: "volume-hosts"

readOnly: false

- command:

- "cat"

env:

- name: "LANGUAGE"

value: "en\_US:en"

- name: "LC\_ALL"

value: "en\_US.UTF-8"

- name: "LANG"

value: "en\_US.UTF-8"

image: "registry.cn-beijing.aliyuncs.com/citools/maven:3.5.3"

imagePullPolicy: "IfNotPresent"

name: "build"

tty: true

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/root/.m2/"

name: "volume-maven-repo"

readOnly: false

- mountPath: "/etc/hosts"

name: "volume-hosts"

readOnly: false

- command:

- "cat"

env:

- name: "LANGUAGE"

value: "en\_US:en"

- name: "LC\_ALL"

value: "en\_US.UTF-8"

- name: "LANG"

value: "en\_US.UTF-8"

image: "registry.cn-beijing.aliyuncs.com/citools/kubectl:self-1.17"

imagePullPolicy: "IfNotPresent"

name: "kubectl"

tty: true

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/var/run/docker.sock"

name: "volume-docker"

readOnly: false

- mountPath: "/mnt/.kube/"

name: "volume-kubeconfig"

readOnly: false

- mountPath: "/etc/hosts"

name: "volume-hosts"

readOnly: false

- command:

- "cat"

env:

- name: "LANGUAGE"

value: "en\_US:en"

- name: "LC\_ALL"

value: "en\_US.UTF-8"

- name: "LANG"

value: "en\_US.UTF-8"

image: "registry.cn-beijing.aliyuncs.com/citools/docker:19.03.9-git"

imagePullPolicy: "IfNotPresent"

name: "docker"

tty: true

volumeMounts:

- mountPath: "/etc/localtime"

name: "volume-2"

readOnly: false

- mountPath: "/var/run/docker.sock"

name: "volume-docker"

readOnly: false

- mountPath: "/etc/hosts"

name: "volume-hosts"

readOnly: false

restartPolicy: "Never"

nodeSelector:

build: "true"

securityContext: {}

volumes:

- hostPath:

path: "/var/run/docker.sock"

name: "volume-docker"

- hostPath:

path: "/usr/share/zoneinfo/Asia/Shanghai"

name: "volume-2"

- hostPath:

path: "/etc/hosts"

name: "volume-hosts"

- name: "volume-maven-repo"

hostPath:

path: "/opt/m2"

- name: "volume-kubeconfig"

secret:

secretName: "multi-kube-config"

'''

}

}

stages {

stage('pulling Code') {

parallel {

stage('pulling Code') {

when {

expression {

env.gitlabBranch == null

}

}

steps {

git(branch: "${BRANCH}", credentialsId: 'cdce3d8e-a859-45ac-9926-ac34236bb744', url: "${REPO\_URL}")

}

}

stage('pulling Code by trigger') {

when {

expression {

env.gitlabBranch != null

}

}

steps {

git(url: "${REPO\_URL}", branch: env.gitlabBranch, credentialsId: 'cdce3d8e-a859-45ac-9926-ac34236bb744')

}

}

}

}

stage('initConfiguration') {

steps {

script {

CommitID = sh(returnStdout: true, script: "git log -n 1 --pretty=format:'%h'").trim()

CommitMessage = sh(returnStdout: true, script: "git log -1 --pretty=format:'%h : %an %s'").trim()

def curDate = sh(script: "date '+%Y%m%d-%H%M%S'", returnStdout: true).trim()

TAG = curDate[0..14] + "-" + CommitID + "-" + BRANCH

}

}

}

stage('Building') {

parallel {

stage('Building') {

steps {

container(name: 'build') {

sh """

echo "Building Project..."

${BUILD\_COMMAND}

"""

}

}

}

stage('Scan Code') {

steps {

sh 'echo "Scan Code"'

}

}

}

}

stage('Build image') {

steps {

withCredentials([usernamePassword(credentialsId: 'REGISTRY\_USER', passwordVariable: 'Password', usernameVariable: 'Username')]) {

container(name: 'docker') {

sh """

docker build -t ${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${TAG} .

docker login -u ${Username} -p ${Password} ${HARBOR\_ADDRESS}

docker push ${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${TAG}

"""

}

}

}

}

stage('Deploy') {

when {

expression {

DEPLOY != "false"

}

}

steps {

container(name: 'kubectl') {

sh """

cat ${KUBECONFIG\_PATH} > /tmp/1.yaml

/usr/local/bin/kubectl config use-context ${CLUSTER} --kubeconfig=/tmp/1.yaml

export KUBECONFIG=/tmp/1.yaml

/usr/local/bin/kubectl set image ${DEPLOY\_TYPE} -l ${DEPLOY\_LABEL} ${CONTAINER\_NAME}=${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${TAG} -n ${NAMESPACE}

"""

}

}

}

}

environment {

CommitID = ''

CommitMessage = ''

TAG = ''

}

}

openssl pkcs12 -export -out /tmp/default.pfx -inkey admin-key.pem -in admin.pem -certfile ca.pem

kubeconfig配置多集群

[root@k8s-master01 pki]# cp ~/.kube/config ./multi-cluster.yaml

[root@k8s-master01 pki]# kubectl config set-cluster test --certificate-authority=ca.pem --embed-certs=true --server=https://192.168.1.88:8443 --kubeconfig=multi-cluster.yaml

Cluster "test" set.

[root@k8s-master01 pki]# kubectl config set-credentials test-admin --client-certificate=admin.pem --client-key=admin-key.pem --emebd-certs=true --kubeconfig=multi-cluster.yaml

Error: unknown flag: --emebd-certs

See 'kubectl config set-credentials --help' for usage.

[root@k8s-master01 pki]# kubectl config set-credentials test-admin --client-certificate=admin.pem --client-key=admin-key.pem --embed-certs=true --kubeconfig=multi-cluster.yaml

User "test-admin" set.

[root@k8s-master01 pki]# kubectl config set-context test --cluster=test --user=test-admin --kubeconfig=multi-cluster.yaml

Context "test" created.

自动构建NodeJS应用：

<https://github.com/selaworkshops/npm-demo-app>

def get\_tags = [ "bash", "-c", "curl -s -u 'HarborUsername:HarborPassword' -X GET -H 'Content-Type: application/json' '${HARBOR\_ADDRESS}/api/repositories/$HARBOR\_PROJECT%2F${ImageName}/tags' | jq .[].name -r | grep -v '^\$' | sort -r" ]

return get\_tags.execute().text.tokenize('\n')

pipeline {

agent any

stages {

stage('Hello') {

steps {

sh """

echo ${IMAGE\_TAG}

kubectl config use-context --kubeconfig=${KUBECONFIG\_PATH} ${CLUSTER}

kubectl --kubeconfig=${KUBECONFIG\_PATH} set image ${DEPLOY\_TYPE} -l ${DEPLOY\_LABEL} ${CONTAINER\_NAME}=${HARBOR\_ADDRESS}/${REGISTRY\_DIR}/${IMAGE\_NAME}:${IMAGE\_TAG} -n ${NAMESPACE}

kubectl --kubeconfig=${KUBECONFIG\_PATH} get po -l ${DEPLOY\_LABEL} -n ${NAMESPACE} -w

"""

}

}

}

}