**Day7-07月24日-基于K8S的持续部署实践**

|  |  |  |
| --- | --- | --- |
| **时间** | **标题** | **内容** |
| **上午** | **Kubectl** | 1. 根据应用编写Deployments部署文件 2. Jenkins流水线中使用Sed替换文件信息 3. Jenkins 集成Kubectl工具完成应用发布 |
| **下午** | **Helm(GitOps)** | 1. Helm部署工具的基本概念 2. 编写Helm Charts模板 3. 使用Helm将应用部署到K8s 4. 将Charts托管到GitLab 实践GitOps |
|  |  |  |

【本次课程总结】

本次课程主要讲解基于Kubernetes的应用发布流水线的实现：

* 了解Docker的基础知识，完成CI流水线中的docker镜像上传步骤。
* 学习Kubernetes的基础知识， 手动编写Deployment模板，完成应用的发布。
* 将部署文件存储在Git仓库， 复用上次课程的"生成版本文件"步骤，来完成对该部署文件的版本更新。
* 学习k8s的发布策略，实践完成基于service的蓝绿部署和基于ingress的灰度发布（手动）。
* 使用kubectl完成对应用的发布， 调试并生成CD流水线。
* 学习Helm基础知识， 手动编写helm chart完成打包和发布更新回滚。
* 调试完成基于Helm的CI/CD GitOps流水线。

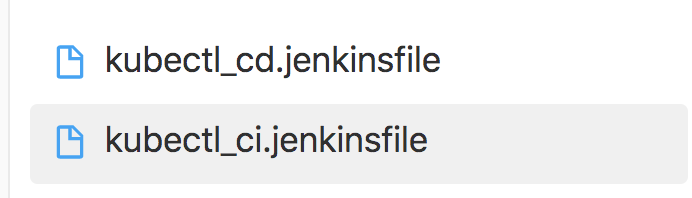
注意： 上午在创建jenkins slave 的时候遇到了坑点，花费了40多分钟排错，还好最后学习到了新的方式方法解决了此类存在已久的bug。 下午在编写sed 和awk 命令时花费了很多时间， sed 中默认使用 '/'分割在jenkins中要换一下，换成"#"没问题测试通过。 awk 中的位置参数 "$" 要用反斜杠转义"'{ print \$1}'"。

发布演示

此处为语雀视频卡片，点击链接查看：[基于Kubectl的k8s发布流水线实践.flv](https://www.yuque.com/zeyangli/bqsigl/zd160d#busYe)

此处为语雀视频卡片，点击链接查看：[基于Helm的k8s发布流水线实践.flv](https://www.yuque.com/zeyangli/bqsigl/zd160d#HvzpI)

**实验与素材：**

* 基于kubectl的GitOps实践素材：参考下面共享库代码 
* 基于Helm的GitOps实践素材： [📎helm.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124967824-f623a88d-a0fb-4bf2-8869-8761415a073b.zip)
* 做实验用的前端项目： [📎devops02-devops-ui-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124206926-0a33a2d7-caf4-4087-a6a1-a216c3c62661.zip)
* 蓝绿部署测试代码： [📎blue-green.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124309176-4ffe60a4-60bf-420a-89a4-485c49890d3b.zip)
* ingress-canary灰度测试代码： [📎ingress-gray.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124389144-7c10483e-8fc6-4bd8-8e30-c369dba47cd4.zip)
* Jenkins共享库和Jenkinsfile: [📎jenkinslib-day7.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124541434-f0eac9cc-90e2-4955-b3bc-c80884e78d7f.zip)
* 最终kubectl 环境库： [📎devops02-k8senv-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124466562-587d873c-d860-4313-8d30-ea4714da8ecc.zip)
* 最终helm chart库： [📎devops02-helm-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627124499574-074c91f1-7456-4c02-98ca-b06f7140cd82.zip)
* day7 sumlimetext 所有素材： [📎day7.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1627126094194-93702554-d9f5-4599-9ab9-d6f2d4e2df4a.zip)

**1. 基于Kubectl的GitOps CI/CD**

**1.1 CI流水线设计与实现**

此次项目我们使用了jenkins共享库来完成最佳实践。  jenkinsfile第一行就是要导入我们的共享库。@Library("devopslib@master") \_

我们在共享库中编写了一些特定的处理类:

* src/org/devops/mytools.groovy     存放小工具（代码下载， 邮件通知）
* src/org/devops/builds.groovy        构建类工具
* src/org/devops/sonarqube.groovy  代码扫描工具
* src/org/devops/artifacts.groovy     制品管理工具
* src/org/devops/gitlab.groovy         版本控制系统接口操作工具

在Jenkinsfile中导入这些类

// 导入类方法

def mytools = new org.devops.mytools()

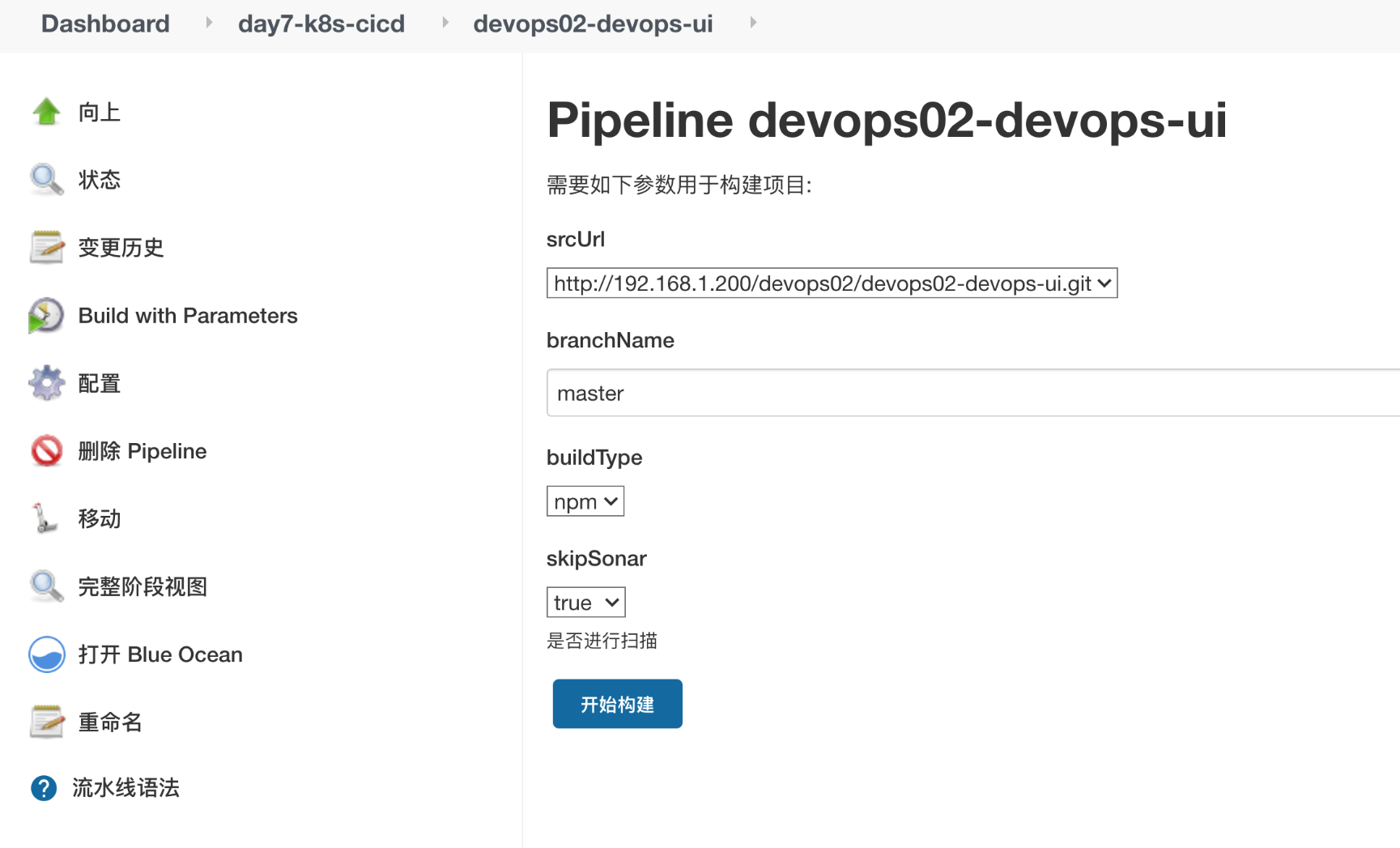
def sonar = new org.devops.sonar()

def nexus = new org.devops.nexus()

def gitlab = new org.devops.gitlab()

有些参数需要用户在Jenkins页面构建前填写的：

* 用户需要在jenkins作业中配置好要构建的代码仓库的地址。（一般不需要改， 可以使用选项参数）
* 每次构建需要输入此代码库要构建的代码分支。 （字符串参数）
* 选择本次构建要使用的构建工具（选项参数maven、ant、gradle、npm）
* 用户自行选择是否跳过代码扫描（选项参数 true、false）



//UI上面的参数

String branchName = "${env.branchName}" // 分支名称

String srcUrl = "${env.srcUrl}" // 仓库地址

String buildType = "${env.buildType}" // 构建类型

String skipSonar = "${env.skipSonar}" // 是否跳过扫描

为了适配不同的构建工具，专门维护了一个map用于管理不同的构建工具的path。标准化 标准化 标准化， 也就是后期所有的agent节点都要安装这些工具的时候，要使用约定好的目录位置。

//定义构建工具

buildMap = ["mvn" : "/usr/local/apache-maven-3.8.1",

"gradle": "/usr/local/gradle-6.8.3",

"golang": "/usr/local/go",

"npm" : "/usr/local/node-v14.16.1-linux-x64"]

// 项目使用的构建工具类型

buildType = "${env.buildType}"

我们在流水线中定义一些全局变量：

* 通过作业的名称拿到我们的制品库名称（devops02）
* 通过作业名称拿到业务简称（devops02）
* 通过作业名称拿到应用名称（devops02-devops-service/ devops02-devops-ui）
* 通过分支名称拿到对应的版本号（release-1.1.1   对应  1.1.1 ）

// 凭据列表

credentials = ["sonar" : '06bf5ee4-f571-4fe4-9b52-d17190ce54e5']

//服务器列表

servers = ["sonar": 'http://192.168.1.200:9000']

// 业务名称、应用名称、版本

env.releaseVersion = "${env.branchName}".split("-")[-1]

appName = "${JOB\_NAME}".split('/')[-1] // Day5/acmp-myapp-service

buName = appName.split('-')[0]

**流水线运行信息**

此条流水线运行在标签为"build"的节点上， 跳过默认的checkout步骤。

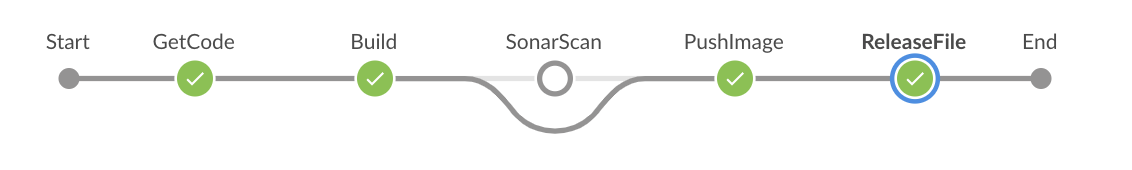
agent { label "build" }

options {

skipDefaultCheckout true

}

**阶段与节点**



下载代码

// 下载代码

stage("CheckOut"){

steps {

script {

checkout( [$class: 'GitSCM',

branches: [[name: "${env.branchName}"]],

extensions: [],

userRemoteConfigs: [[

credentialsId: 'bfe51cb6-9281-447d-86a7-adc06b964e48',

url: "${env.srcUrl}"]]])

}

}

}

构建代码和单元测试

// 构建代码

stage("Build"){

steps{

script{

mytools.CustomBuild(buildType,buildMap)

}

}

}

//单元测试

stage("UnitTest"){

steps{

script {

mytools.CustomTest(buildType,buildMap)

}

}

post {

success {

script{

mytools.UnitTest(buildType)

}

}

}

}

代码扫描

// 代码扫描

stage("CodeScan"){

when {

environment name: 'skipSonar', value: 'false'

}

steps{

script{

sonar.CodeScan(credentials, servers, buildType)

}

}

}

构建上传镜像

* 需要将docker镜像仓库的用户认证信息，以用户名和密码类型的凭据在Jenkins中存储；
* docker仓库：192.168.1.200:8088/${buName}/${appName}；
* 镜像标签： ${releaseVersion}-${nowDate} 即 版本号-日期；
* 步骤： 登录镜像仓库> 构建镜像 > 上传镜像 > 删除本地镜像（节省空间）；

//上传制品(jar、war >> docker images )

stage("PushArtifact"){

steps{

script{

//env.artifactUrl = nexus.PushArtifacts(buildType)

//println(env.artifactUrl)

def buildTime = sh returnStdout: true, script: 'date +%Y%m%d%H%M%S'

buildTime = buildTime - "\n"

env.imageTag = "${env.releaseVersion}-${buildTime}"

env.imageName = "192.168.1.200:8088/${buName}/${appName}:${env.imageTag}"

// 自定义index.html

sh "echo ${appName}----${env.releaseVersion} >index.html"

sh """

## 登录镜像仓库

docker login -u admin -pHarbor12345 192.168.1.200:8088

## 构建镜像

docker build -t ${env.imageName} -f ./Dockerfile .

## 上传镜像

docker push ${env.imageName}

sleep 2

## 清除镜像

docker rmi ${env.imageName}

"""

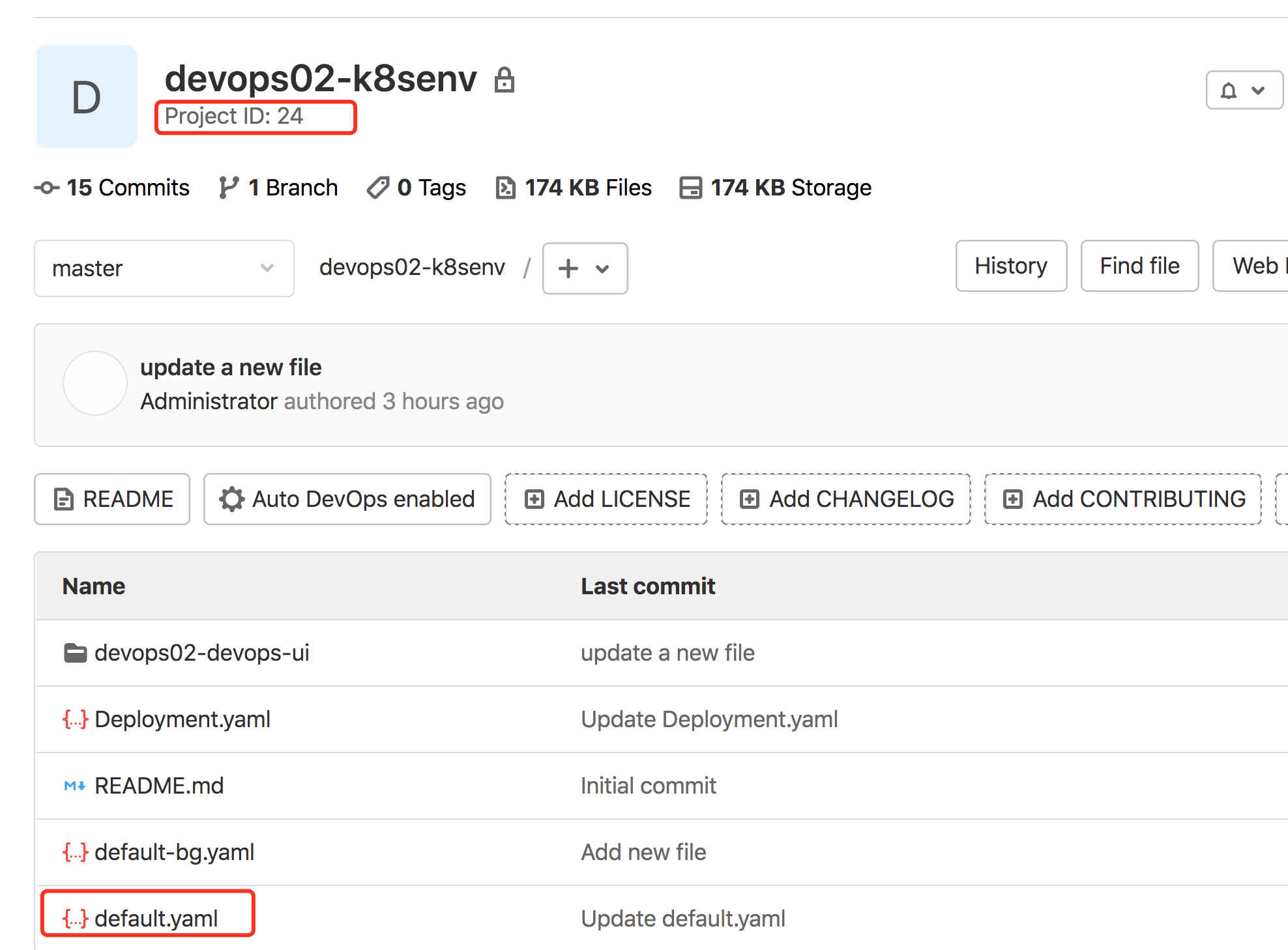
}

}

}

更新生成版本文件

* 步骤： 拿到 env仓库中的default.yaml模板文件， 然后替换内容，更新到版本库。



* 更新对象：
  + \_\_APPNAME\_\_（应用名称）对应Jenkins作业名称；
  + \_\_NAMESPACE\_\_ (名称空间) 对应业务名称；
  + \_\_INAGENAME\_\_(镜像名称)

// 生成版本文件

stage("ReleaseFile"){

steps {

script {

//下载版本库文件 devops02-k8senv/default.yaml

response = gitlab.GetRepoFile(24,"default.yaml", "master")

//println(response)

//替换文件中内容

yamlData = readYaml text: """${response}"""

sh "rm -fr default.yaml"

writeFile file: 'default.yaml', text: """${response}"""

sh """

## 替换APPNAME

sed -i 's#\_\_APPNAME\_\_#${appName}#g' default.yaml

sed -i 's#\_\_NAMESPACE\_\_#${buName}#g' default.yaml

sed -i 's#\_\_IMAGENAME\_\_#${env.imageName}#g' default.yaml

"""

newYaml = sh returnStdout: true, script: 'cat default.yaml'

println(newYaml)

//更新gitlab文件内容

base64Content = newYaml.bytes.encodeBase64().toString()

// 会有并行问题，同时更新报错

try {

gitlab.UpdateRepoFile(24,"${appName}%2f${env.releaseVersion}.yaml",base64Content, "master")

} catch(e){

gitlab.CreateRepoFile(24,"${appName}%2f${env.releaseVersion}.yaml",base64Content, "master")

}

}

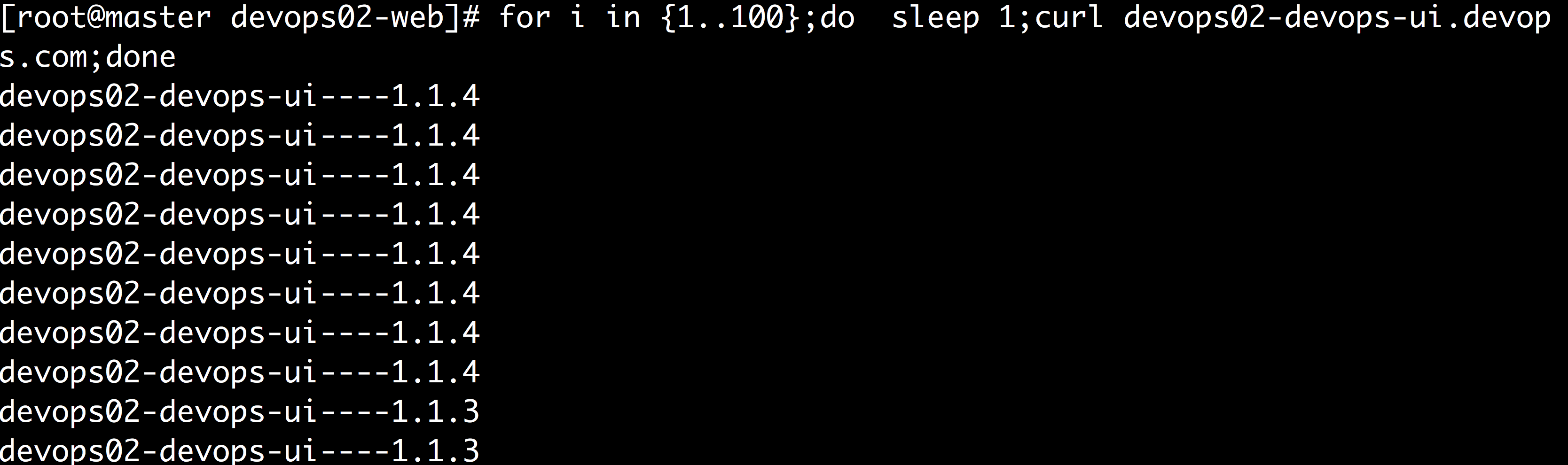
}

}

镜像仓库截图：



**1.2 CD流水线设计与实现**



此次项目我们使用了jenkins共享库来完成最佳实践。  jenkinsfile第一行就是要导入我们的共享库。@Library("devopslib@master") \_

我们在共享库中编写了一些特定的处理类:

* src/org/devops/mytools.groovy     存放小工具（代码下载， 邮件通知）
* src/org/devops/gitlab.groovy         版本控制系统接口操作工具

在Jenkinsfile中导入这些类

// 导入类方法

def mytools = new org.devops.mytools()

def gitlab = new org.devops.gitlab()

只有一个参数需要用户在Jenkins页面构建前填写的：

* 用户在页面输入发布版本： branchName， 例如 RELEASE-1.1.1



我们在流水线中定义一些全局变量：

* 通过作业名称拿到业务简称（anyops）
* 通过作业名称拿到应用名称（anyops-devops-service/ anyops-devops-ui）

// 业务名称、应用名称

env.releaseVersion = "${env.branchName}".split("-")[-1]

appName = "${JOB\_NAME}".split('/')[-1].split("\_")[0] // Day5/acmp-myapp-service

buName = appName.split('-')[0]

**流水线运行信息**

此条流水线运行在标签为"k8sbuild"的节点上， 跳过默认的checkout步骤。（课程中是在k8s的master节点启动了一个jnlp的Jenkins Slave）

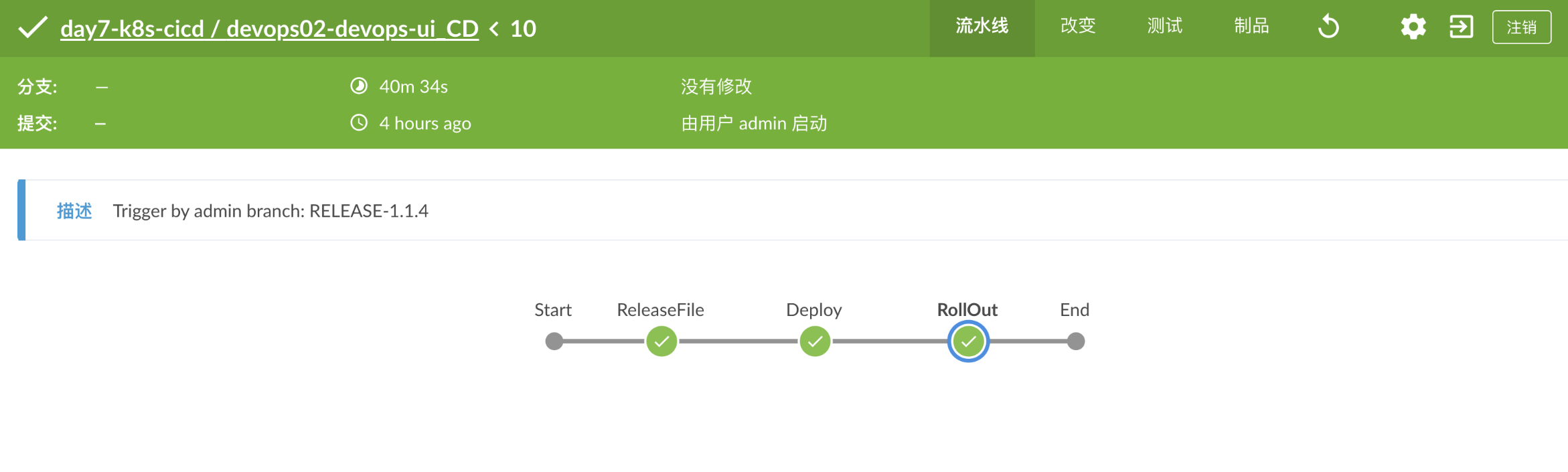
agent { label "k8sbuild"}

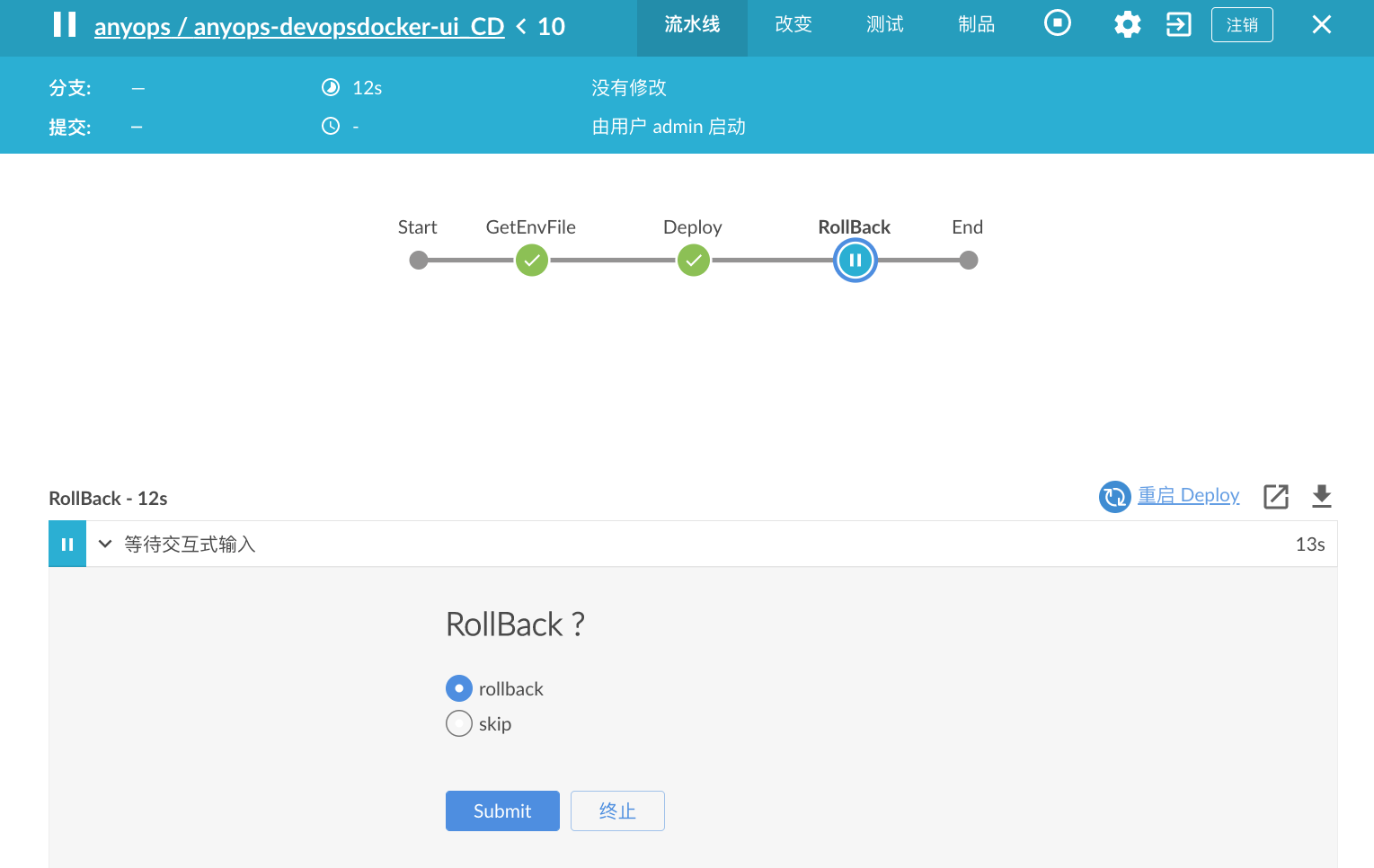
options {

skipDefaultCheckout true

}

**阶段与节点**





获取发布文件

* 下载CI流水线生成的版本文件到本地；

// 获取版本文件

stage("ReleaseFile"){

steps {

script {

//下载版本库文件 devops02-k8senv/default.yaml

response = gitlab.GetRepoFile(24,"${appName}%2f${env.releaseVersion}.yaml", "master")

//println(response)

//替换文件中内容

yamlData = readYaml text: """${response}"""

sh "rm -fr ${env.releaseVersion}.yaml"

writeFile file: "${env.releaseVersion}.yaml", text: """${response}"""

}

}

}

应用发布

* 创建名称空间，更新应用，查看REVISION历史。

// 发布

stage("Deploy"){

steps{

script{

// kubectl 更新应用

sh "kubectl apply -f ${env.releaseVersion}.yaml"

// 获取应用状态

5.times{

sh "sleep 2; kubectl -n ${buName} get pod | grep ${appName}"

}

}

}

}

回滚步骤（交互）

* 给用户提供两个选项 ：yes执行回滚，no跳过;
* 使用kubectl rollout undo 默认回滚上个版本；

// rollout

stage("RollOut"){

input {

message "是否继续回滚"

ok "Yes"

submitter "zeyang,aa"

parameters {

choice(choices: ['yes', 'no'], name: 'opts')

// string(name: 'opts', defaultValue: 'no', description: 'yes/no')

}

}

steps{

script{

echo "opts ---> ${opts}"

if ("${opts}" == "yes"){

// 回滚命令

sh " kubectl rollout undo deployment/${appName} -n ${buName}"

}

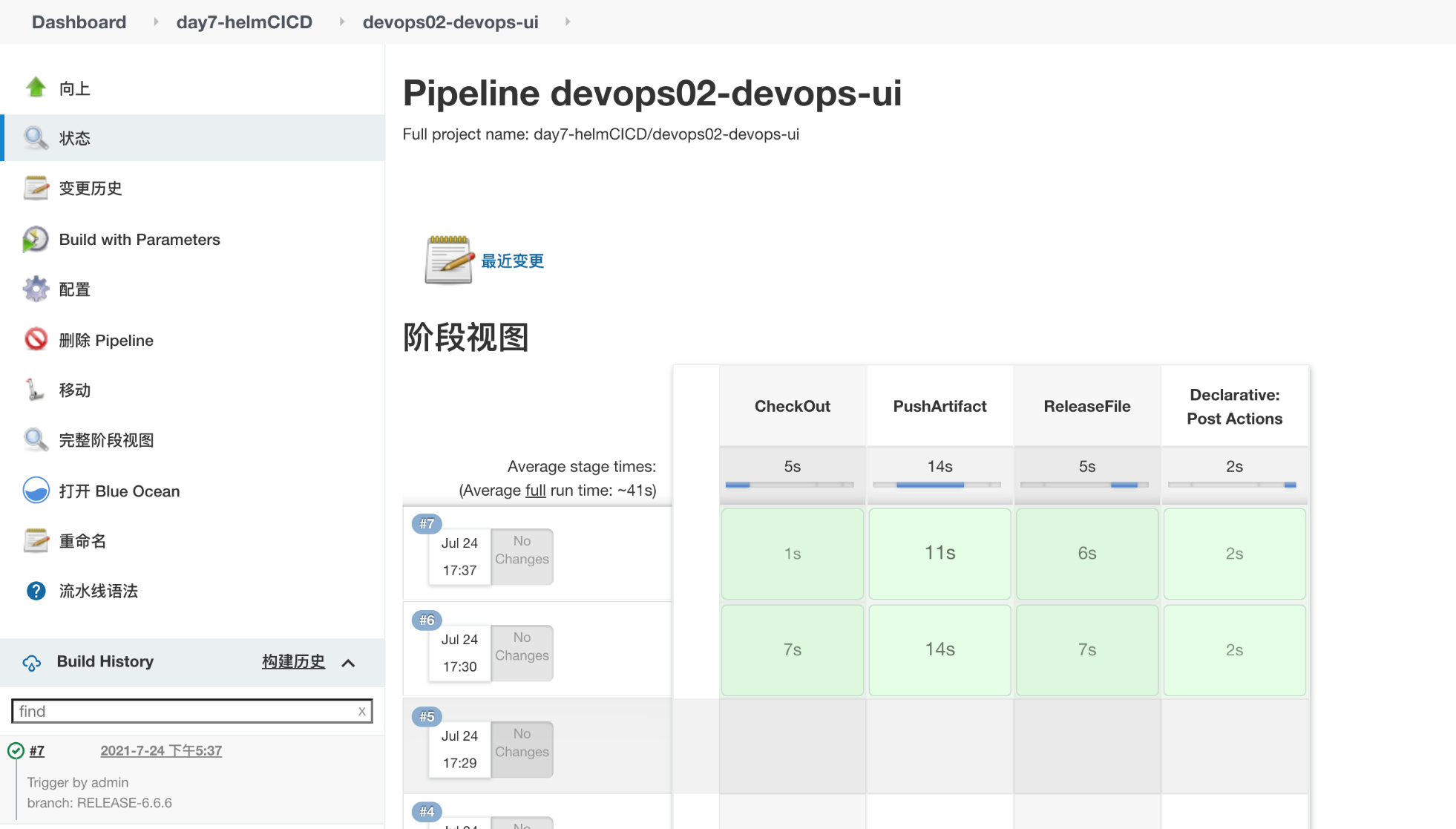
}

}

}

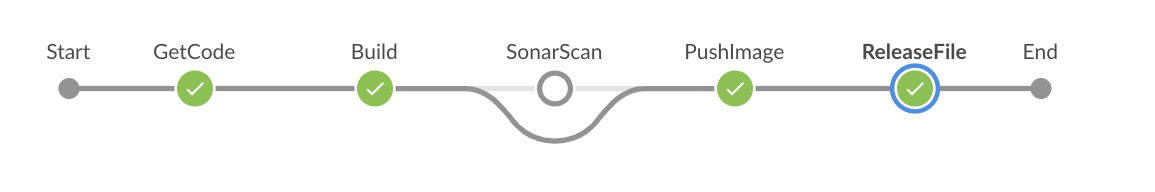
**2. 基于Helm的GitOps CI/CD**

**2.1 CI流水线设计与实现**



此部分可以参考上面“基于kubectl发布”阶段， 相同的阶段代码。

**阶段与节点**



此部分可以参考上面“基于kubectl发布”阶段， 相同的阶段代码。

更新生成版本文件

此处的版本文件不再是一个文件了， 而是一个chart包。如何生成：

helm create "${appName}"

cd "${appName}"

git push 到版本库

* 步骤： 拿到 helm chart仓库中的values.yaml模板文件， 然后替换镜像，更新到版本库。
* 更新对象：
  + image.repository (镜像仓库名称)
  + image.tag （镜像标签）

// 生成版本文件

stage("ReleaseFile"){

steps {

script {

//下载helm values.yaml

//下载版本库文件 devops02-devops-ui/values.yaml

response = gitlab.GetRepoFile(25,"${appName}%2fvalues.yaml", "master")

//println(response)

//替换文件中内容

yamlData = readYaml text: """${response}"""

yamlData.image.repository = "192.168.1.200:8088/${buName}/${appName}"

yamlData.image.tag = "${env.imageTag}"

sh "rm -fr values.yaml"

writeYaml charset: 'UTF-8', file: 'values.yaml', data: yamlData

newYaml = sh returnStdout: true, script: 'cat values.yaml'

println(newYaml)

//更新gitlab文件内容

base64Content = newYaml.bytes.encodeBase64().toString()

// 会有并行问题，同时更新报错

try {

gitlab.UpdateRepoFile(25,"${appName}%2fvalues.yaml",base64Content, "master")

} catch(e){

gitlab.CreateRepoFile(25,"${appName}%2fvalues.yaml",base64Content, "master")

}

}

}

}

**2.2 CD流水线设计与实现**







下载chart仓库

// 下载代码

stage("CheckOut"){

steps {

script {

checkout( [$class: 'GitSCM',

branches: [[name: "${env.branchName}"]],

extensions: [],

userRemoteConfigs: [[

credentialsId: 'bfe51cb6-9281-447d-86a7-adc06b964e48',

url: "${env.srcUrl}"]]])

}

}

}

helm发布应用

* 创建名称空间，打包，获取REVISION历史（便于回滚）

// 发布

stage("Deploy"){

steps{

script{

//Helm 发布

sh """

kubectl create ns ${buName} || echo true

helm package "${appName}/"

helm install "${appName}" ./"${appName}"-\*.tgz -n ${buName} || helm upgrade "${appName}" ./"${appName}"-\*.tgz -n ${buName}

helm history "${appName}" -n ${buName}

"""

env.revision = sh returnStdout: true, script: """helm history ${appName} -n ${buName} | grep -v 'REVISION' | awk '{print \$1}' """

println("${env.revision}")

println("${env.revision.split('\n').toString()}")

env.REVISION = "${env.revision.split('\n').toString()}"

println("${env.REVISION}")

// 获取应用状态

5.times{

sh "sleep 2; kubectl -n ${buName} get pod | grep ${appName}"

}

}

}

}

回滚步骤

* 交互式用户选择REVISION， 触发Helm rollback；

// rollout

stage("RollBack"){

steps{

script{

def result = input message: 'RollBack?', ok: 'submit', parameters: [choice(choices: "${env.REVISION}", description: '', name: 'revision')]

env.result = result - "\n"

echo "Actions is ${env.result}, doing......."

if ( "${env.result}" != ""){

sh """ helm rollback ${appName} ${env.result} -n ${buName} """

} else {

println("Skip rollback .....")

}

}

}

}

**3. 核心技术点讲解**

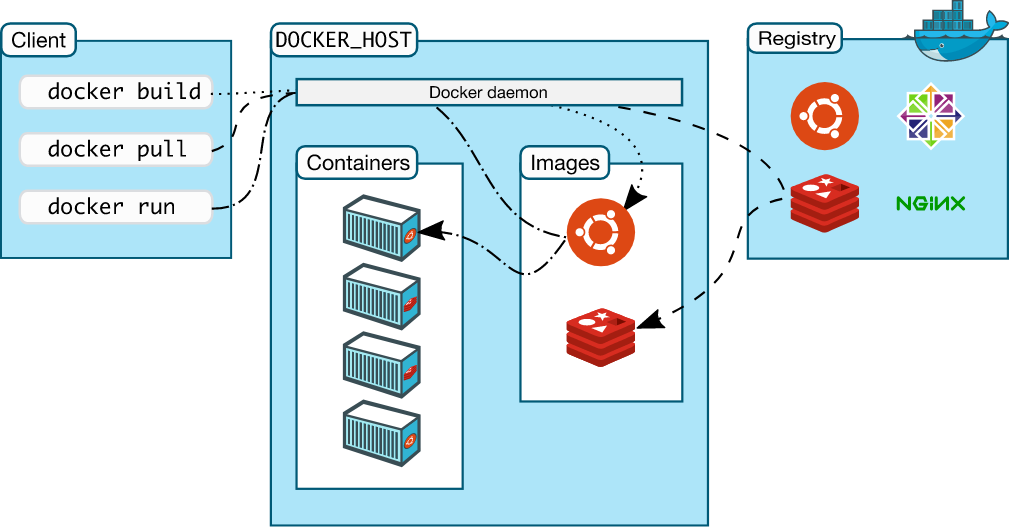
**3.1 Docker基础**

Docker官网: <https://www.docker.com/>

Docker中文社区: <http://www.docker.org.cn/>

Docker HUB: <https://hub.docker.com/>

Docker 使用**客户端-服务器 (C/S) 架构**模式。Docker 客户端会与 Docker 守护进程进行通信。Docker 守护进程会处理复杂繁重的任务，例如建立、运行、发布Docker 容器。Docker 客户端和守护进程可以运行在同一个系统上，当然也可以使用 Docker 客户端去连接一个远程的 Docker 守护进程。Docker 客户端和守护进程之间通过 socket 或者 RESTful API 进行通信。



* Docker守护进程: Docker 守护进程运行在一台主机上。用户并不直接和守护进程进行交互，而是通过Docker 客户端间接和其通信。
* Docker客户端: 实际上是 docker 的二进制程序，是主要的用户与 Docker 交互方式。它接收用户指令并且与背后的 Docker 守护进程通信，
* Docker 镜像: Docker 容器运行时的只读模板，每一个镜像由一系列的层 (layers) 组成
* Docker 仓库: 仓库用来保存镜像，可以理解为代码控制中的代码仓库。同样的，Docker 仓库也有公有和私有的概念。
* Docker 容器: 一个Docker容器包含了所有的某个应用运行所需要的环境。每一个 Docker 容器都是从 Docker 镜像创建的。

**特性**

* **灵活**：复杂的应用程序也可以容器化。
* **轻量级**：容器利用并共享了主机内核，在系统资源方面比虚拟机更加有效。
* **可移植性**：可以在本地构建，部署到云并在任何地方运行。
* **松散耦合**：容器是高度自给自足并封装的容器，使在不破坏其他容器的情况下更换或升级。
* **可扩展**：在数据中心内增加并自动分布容器副本。
* **安全**：容器将积极的约束和隔离应用于流程，而无需用户方面的任何配置。

**Docker安装**

yum源：<https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/centos/docker-ce.repo>

可以替换文件中的url，这样下载速度更快。

:%s/download.docker.com/mirrors.tuna.tsinghua.edu.cn\/docker-ce/g

Docker version 19.03.12, build 48a66213fe

dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo

dnf list docker-ce

dnf install docker-ce --nobest -y

systemctl start docker

systemctl enable docker

docker -v

**Docker指令**

[root@myserver ~]# docker -h

Flag shorthand -h has been deprecated, please use --help

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:

--config string Location of client config files (default "/root/.docker")

-c, --context string Name of the context to use to connect to the daemon (overrides

DOCKER\_HOST env var and default context set with "docker context

use")

-D, --debug Enable debug mode

-H, --host list Daemon socket(s) to connect to

-l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal")

(default "info")

--tls Use TLS; implied by --tlsverify

--tlscacert string Trust certs signed only by this CA (default "/root/.docker/ca.pem")

--tlscert string Path to TLS certificate file (default "/root/.docker/cert.pem")

--tlskey string Path to TLS key file (default "/root/.docker/key.pem")

--tlsverify Use TLS and verify the remote

-v, --version Print version information and quit

Management Commands:

builder Manage builds

config Manage Docker configs

container Manage containers

context Manage contexts

engine Manage the docker engine

image Manage images

network Manage networks

node Manage Swarm nodes

plugin Manage plugins

secret Manage Docker secrets

service Manage services

stack Manage Docker stacks

swarm Manage Swarm

system Manage Docker

trust Manage trust on Docker images

volume Manage volumes

Commands:

attach Attach local standard input, output, and error streams to a running container

build Build an image from a Dockerfile

commit Create a new image from a container's changes

create Create a new container

import Import the contents from a tarball to create a filesystem image

load Load an image from a tar archive or STDIN

login Log in to a Docker registry

logout Log out from a Docker registry

push Push an image or a repository to a registry

rmi Remove one or more images

save Save one or more images to a tar archive (streamed to STDOUT by default)

tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

update Update configuration of one or more containers

**info        系统信息**

[root@myserver ~]# docker info

Client:

Debug Mode: false

Server:

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 18.09.1

Storage Driver: overlay2

Backing Filesystem: xfs

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

**version     版本信息**

[root@myserver ~]# docker version

Client: Docker Engine - Community

Version: 19.03.12

API version: 1.39

Go version: go1.13.10

Git commit: 48a66213fe

Built: Mon Jun 22 15:46:54 2020

OS/Arch: linux/amd64

Experimental: false

Server: Docker Engine - Community

Engine:

Version: 18.09.1

API version: 1.39 (minimum version 1.12)

Go version: go1.10.6

Git commit: 4c52b90

Built: Wed Jan 9 19:06:30 2019

OS/Arch: linux/amd64

Experimental: false

wait Block until one or more containers stop, then print their exit codes

**search 查找镜像**

docker search hello-world

**pull 下载镜像(docker hub)**

[root@myserver ~]# docker pull hello-world

Using default tag: latest

latest: Pulling from library/hello-world

0e03bdcc26d7: Pull complete

Digest: sha256:d58e752213a51785838f9eed2b7a498ffa1cb3aa7f946dda11af39286c3db9a9

Status: Downloaded newer image for hello-world:latest

docker.io/library/hello-world:latest

**images 查看本地的镜像**

[root@myserver ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-world latest bf756fb1ae65 6 months ago 13.3kB

**run        运行容器**

* -i : 交互模式
* -t： 分配一个终端
* -d： 后台运行
* --name： 指定容器的名称
* -p: 端口绑定

docker run [options] IMAGE

docker run hello-world

docker run --name mywebserver -it nginx

[root@myserver ~]# docker run --name mywebserver -itd nginx

8d485b358cc693c1e2b2d9b5c1d1a8092265d79bdcf1ee54d3d2f615f4f22064

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 6 seconds ago Up 5 seconds 80/tcp mywebserver

**ps 查看容器进程（ -a 所有的，正在运行、已经停止的）**

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@myserver ~]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

01bdedde3725 hello-world "/hello" 2 minutes ago Exited (0) 2 minutes ago hopeful\_mendel

**rm 删除容器 （-f 强制）**

## 指定容器的名称

[root@myserver ~]# docker rm mywebserver

mywebserver

[root@myserver ~]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

01bdedde3725 hello-world "/hello" 9 minutes ago Exited (0) 9 minutes ago hopeful\_mendel

## 指定容器的ID删除

[root@myserver ~]# docker rm 01bdedde3725

01bdedde3725

[root@myserver ~]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@myserver ~]#

**start/restart/stop/ 容器控制**

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 20 minutes ago Up 20 minutes 80/tcp mywebserver

[root@myserver ~]# docker stop mywebserver

mywebserver

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@myserver ~]# docker start mywebserver

mywebserver

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 21 minutes ago Up 3 seconds 80/tcp mywebserver

[root@myserver ~]# docker restart mywebserver

mywebserver

**stats 显示容器使用的资源（CPU）**

docker stats mywebserver

[root@myserver ~]# docker stats mywebserver

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

8d485b358cc6 mywebserver 0.00% 2.309MiB / 3.664GiB 0.06% 3.44kB / 0B 0B / 0B 2

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PID

**logs 查看容器日志（-f 动态更新）**

[root@myserver ~]# docker logs -f mywebserver

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration

/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/

/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh

10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf

10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf

/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh

/docker-entrypoint.sh: Configuration complete; ready for start up

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration

/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/

/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh

10-listen-on-ipv6-by-default.sh: IPv6 listen already enabled, exiting

/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh

/docker-entrypoint.sh: Configuration complete; ready for start up

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration

/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/

/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh

10-listen-on-ipv6-by-default.sh: IPv6 listen already enabled, exiting

/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh

/docker-entrypoint.sh: Configuration complete; ready for start up

**pause/unpause 暂停状态**

[root@myserver ~]# docker pause mywebserver

mywebserver

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 27 minutes ago Up 5 minutes (Paused) 80/tcp mywebserver

[root@myserver ~]# docker unpause mywebserver

mywebserver

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 27 minutes ago Up 6 minutes 80/tcp mywebserver

[root@myserver ~]#

**rename      重命名**

docker rename CONTAINER NEW\_NAME

[root@myserver ~]# docker rename mywebserver webserver

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 29 minutes ago Up 8 minutes 80/tcp webserver

**inspect 显示容器的详细属性信息**

[root@myserver ~]# docker inspect webserver

[

{

"Id": "8d485b358cc693c1e2b2d9b5c1d1a8092265d79bdcf1ee54d3d2f615f4f22064",

"Created": "2020-07-09T01:55:08.20199984Z",

"Path": "/docker-entrypoint.sh",

"Args": [

"nginx",

"-g",

"daemon off;"

],

"State": {

"Status": "running",

"Running": true,

"Paused": false,

"Restarting": false,

"OOMKilled": false,

"Dead": false,

"Pid": 6429,

"ExitCode": 0,

"Error": "",

"StartedAt": "2020-07-09T02:16:41.996582526Z",

"FinishedAt": "2020-07-09T02:16:40.968627884Z"

},

**kill 关闭容器进程**

[root@myserver ~]# docker kill webserver

webserver

[root@myserver ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@myserver ~]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8d485b358cc6 nginx "/docker-entrypoint.…" 33 minutes ago Exited (137) 7 seconds ago webserver

[root@myserver ~]#

**top 查看容器进程**

[root@myserver ~]# docker top webserver

UID PID PPID C STIME TTY TIME CMD

root 7153 7136 0 22:30 pts/0 00:00:00 nginx: master process nginx -g daemon off;

101 7211 7153 0 22:30 pts/0 00:00:00 nginx: worker process

[root@myserver ~]# docker top webserver 7153

PID TTY STAT TIME COMMAND

7153 pts/0 Ss+ 0:00 nginx: master process nginx -g daemon off;

[root@myserver ~]#

**exec 进入容器执行命令**

[root@myserver ~]# docker exec -it webserver bash

root@8d485b358cc6:/# ls

curl 访问服务器nginx 成功

**port 显示容器端口绑定信息**

[root@myserver ~]# docker port webserver

80/tcp -> 0.0.0.0:8000

**cp 文件复制**

[root@myserver ~]# docker cp /etc/hosts webserver:/tmp

[root@myserver ~]# docker exec -it webserver ls /tmp

hosts

[root@myserver ~]# docker cp webserver:/tmp/hosts ./

[root@myserver ~]# ls hosts

hosts

**diff 容器文件系统发生的变化**

[root@myserver ~]# docker diff webserver

C /etc

C /etc/nginx

C /etc/nginx/conf.d

C /etc/nginx/conf.d/default.conf

C /run

A /run/nginx.pid

C /tmp

A /tmp/hosts

C /var

C /var/cache

C /var/cache/nginx

A /var/cache/nginx/fastcgi\_temp

A /var/cache/nginx/proxy\_temp

A /var/cache/nginx/scgi\_temp

A /var/cache/nginx/uwsgi\_temp

A /var/cache/nginx/client\_temp

**export 导出容器文件系统到本地归档**

[root@myserver ~]# docker export -o test.tar webserver

[root@myserver ~]# mkdir docker

[root@myserver ~]# mv test.tar docker/

[root@myserver ~]# cd docker/

[root@myserver docker]# ls

test.tar

[root@myserver docker]# tar xf test.tar

l[root@myserver docker]# ls

bin dev docker-entrypoint.sh home lib64 mnt proc run srv test.tar usr

boot docker-entrypoint.d etc lib media opt root sbin sys tmp var

[root@myserver docker]# cd tmp/

[root@myserver tmp]# ls

hosts

**import 导入压缩文件到镜像**

[root@myserver docker]# ls

bin dev docker-entrypoint.sh etc lib media opt root sbin sys tmp var

boot docker-entrypoint.d Dockerfile home lib64 mnt proc run srv test.tar usr

[root@myserver docker]# docker import test.tar webserver:v2

sha256:5aa7a154ce2f07e8dc94d331d8809ea40ea5fab4ffa8935e2092771f863e3c0f

[root@myserver docker]# docker history webserver:v2

IMAGE CREATED CREATED BY SIZE COMMENT

5aa7a154ce2f 15 seconds ago 130MB Imported from -

[root@myserver docker]#

**history 查看镜像的更新历史**

[root@myserver ~]# docker history nginx

IMAGE CREATED CREATED BY SIZE COMMENT

2622e6cca7eb 4 weeks ago /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon… 0B

<missing> 4 weeks ago /bin/sh -c #(nop) STOPSIGNAL SIGTERM 0B

<missing> 4 weeks ago /bin/sh -c #(nop) EXPOSE 80 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ENTRYPOINT ["/docker-entr… 0B

<missing> 4 weeks ago /bin/sh -c #(nop) COPY file:cc7d4f1d03426ebd… 1.04kB

<missing> 4 weeks ago /bin/sh -c #(nop) COPY file:b96f664d94ca7bbe… 1.96kB

<missing> 4 weeks ago /bin/sh -c #(nop) COPY file:d68fadb480cbc781… 1.09kB

<missing> 4 weeks ago /bin/sh -c set -x && addgroup --system -… 62.9MB

<missing> 4 weeks ago /bin/sh -c #(nop) ENV PKG\_RELEASE=1~buster 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ENV NJS\_VERSION=0.4.1 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ENV NGINX\_VERSION=1.19.0 0B

<missing> 4 weeks ago /bin/sh -c #(nop) LABEL maintainer=NGINX Do… 0B

<missing> 4 weeks ago /bin/sh -c #(nop) CMD ["bash"] 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ADD file:4d35f6c8bbbe6801c… 69.2MB

**login /logout     Docker registry**

[root@myserver ~]# docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: devopsvip

Password:

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

[root@myserver ~]#

[root@myserver ~]# docker logout

Removing login credentials for https://index.docker.io/v1/

**search 查找镜像**

docker search hello-world

**pull 下载镜像(docker hub)**

[root@myserver ~]# docker pull hello-world

Using default tag: latest

latest: Pulling from library/hello-world

0e03bdcc26d7: Pull complete

Digest: sha256:d58e752213a51785838f9eed2b7a498ffa1cb3aa7f946dda11af39286c3db9a9

Status: Downloaded newer image for hello-world:latest

docker.io/library/hello-world:latest

**images 查看本地的镜像**

[root@myserver ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-world latest bf756fb1ae65 6 months ago 13.3kB

**tag重命名**

[root@myserver ~]# docker tag hello-world:latest devopsvip/docker:latest

[root@myserver ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx latest 2622e6cca7eb 4 weeks ago 132MB

centos 7 b5b4d78bc90c 2 months ago 203MB

devopsvip/docker latest bf756fb1ae65 6 months ago 13.3kB

hello-world latest bf756fb1ae65 6 months ago 13.3kB

**push  上传镜像到镜像仓库**

[root@myserver ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx latest 2622e6cca7eb 4 weeks ago 132MB

centos 7 b5b4d78bc90c 2 months ago 203MB

devopsvip/docker latest bf756fb1ae65 6 months ago 13.3kB

hello-world latest bf756fb1ae65 6 months ago 13.3kB

[root@myserver ~]# docker push devopsvip/docker:latest

The push refers to repository [docker.io/devopsvip/docker]

9c27e219663c: Preparing

denied: requested access to the resource is denied

[root@myserver ~]# docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: devopsvip

Password:

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

[root@myserver ~]# docker push devopsvip/docker:latest

The push refers to repository [docker.io/devopsvip/docker]

9c27e219663c: Mounted from library/hello-world

latest: digest: sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042 size: 525

**rmi 删除镜像**

[root@myserver ~]# docker rmi devopsvip/docker:latest

Untagged: devopsvip/docker:latest

Untagged: devopsvip/docker@sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042

[root@myserver ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx latest 2622e6cca7eb 4 weeks ago 132MB

centos 7 b5b4d78bc90c 2 months ago 203MB

hello-world latest bf756fb1ae65 6 months ago 13.3kB

[root@myserver ~]#

**commit 将容器变化为镜像**

[root@myserver ~]# docker commit -a "devopsvipxx" -m "hhahh" webserver webserver:v1

sha256:d315f2bb406067262798358e85aaef06f2d99dfc1adbc8380dd9cc5d0f809e76

[root@myserver ~]# docker history webserver:v1

IMAGE CREATED CREATED BY SIZE COMMENT

d315f2bb4060 2 seconds ago nginx -g daemon off; 1.39kB hhahh

2622e6cca7eb 4 weeks ago /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon… 0B

<missing> 4 weeks ago /bin/sh -c #(nop) STOPSIGNAL SIGTERM 0B

<missing> 4 weeks ago /bin/sh -c #(nop) EXPOSE 80 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ENTRYPOINT ["/docker-entr… 0B

<missing> 4 weeks ago /bin/sh -c #(nop) COPY file:cc7d4f1d03426ebd… 1.04kB

<missing> 4 weeks ago /bin/sh -c #(nop) COPY file:b96f664d94ca7bbe… 1.96kB

<missing> 4 weeks ago /bin/sh -c #(nop) COPY file:d68fadb480cbc781… 1.09kB

<missing> 4 weeks ago /bin/sh -c set -x && addgroup --system -… 62.9MB

<missing> 4 weeks ago /bin/sh -c #(nop) ENV PKG\_RELEASE=1~buster 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ENV NJS\_VERSION=0.4.1 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ENV NGINX\_VERSION=1.19.0 0B

<missing> 4 weeks ago /bin/sh -c #(nop) LABEL maintainer=NGINX Do… 0B

<missing> 4 weeks ago /bin/sh -c #(nop) CMD ["bash"] 0B

<missing> 4 weeks ago /bin/sh -c #(nop) ADD file:4d35f6c8bbbe6801c… 69.2MB

**build 构建镜像**

Dockerfile

FROM nginx

docker build .

docker build -f aa/Dockerfile .

[root@myserver docker]# docker build .

Sending build context to Docker daemon 269.2MB

Step 1/1 : FROM webserver:v1

---> d315f2bb4060

Successfully built d315f2bb4060

[root@myserver docker

[root@myserver docker]# docker build -t webserver:v2 .

Sending build context to Docker daemon 269.2MB

Step 1/1 : FROM webserver:v1

---> d315f2bb4060

Successfully built d315f2bb4060

Successfully tagged webserver:v2

**load 通过归档文件加载镜像**  
  
**save 将镜像制作归档文件**

[root@myserver docker]# docker save -o webserver-v2.tar webserver:v2

[root@myserver docker]# docker rmi webserver:v2

Untagged: webserver:v2

Deleted: sha256:5aa7a154ce2f07e8dc94d331d8809ea40ea5fab4ffa8935e2092771f863e3c0f

Deleted: sha256:eef9ee7ccbdfd908b67057aa17a6d7292bbc0e7cb04a57896d25ce24a6e115b7

[root@myserver docker]# docker load -i webserver-v2.tar

eef9ee7ccbdf: Loading layer 134.6MB/134.6MB

Loaded image: webserver:v2

[root@myserver docker]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

webserver v2 5aa7a154ce2f 18 hours ago 130MB

webserver v1 d315f2bb4060 18 hours ago 132MB

<none> <none> 46a71bf11111 18 hours ago 132MB

<none> <none> 5ab53db1229d 18 hours ago 132MB

nginx latest 2622e6cca7eb 4 weeks ago 132MB

centos 7 b5b4d78bc90c 2 months ago 203MB

devopsvip/docker latest bf756fb1ae65 6 months ago 13.3kB

hello-world latest bf756fb1ae65 6 months ago 13.3kB

[root@myserver docker]#

create 创建一个容器

[root@myserver docker]# docker create --name webnewserver nginx

bbc81d2c72831167da41a6ccda4e06e6efe51eba81cf6daace0f49e5815e88db

[root@myserver docker]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@myserver docker]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

bbc81d2c7283 nginx "/docker-entrypoint.…" 17 seconds ago Created webnewserver

d2813b963195 nginx "/docker-entrypoint.…" 19 hours ago Created youthful\_ramanujan

4fe81888bdd1 nginx "/docker-entrypoint.…" 24 hours ago Exited (255) 11 minutes ago 0.0.0.0:8000->80/tcp webserver

[root@myserver docker]# docker start webnewserver

webnewserver

[root@myserver docker]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

bbc81d2c7283 nginx "/docker-entrypoint.…" 41 seconds ago Up 4 seconds 80/tcp webnewserver

[root@myserver docker]#

**update 更新容器的配置**

[root@myserver docker]# docker update --memory 1G --memory-swap 1G webaserver

[root@myserver docker]# docker stats webaserver

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

bd8375d39858 webaserver 0.00% 2.445MiB / 1GiB 0.24% 3.69kB / 0B

**Dockerfile构建镜像**

Dockerfile: 用来构建镜像的文件，文件中包含指令。docker build

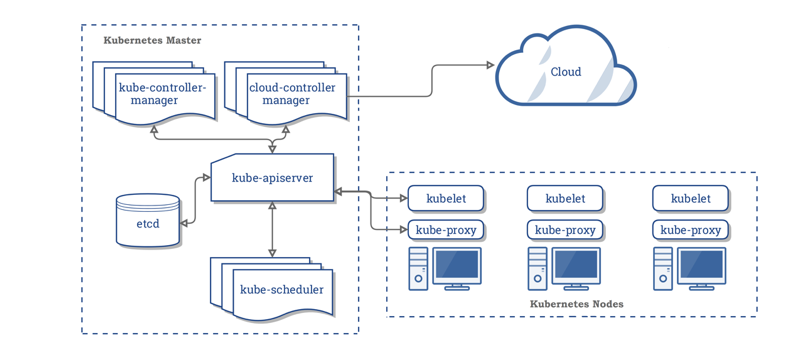
docker build -t image:v1.0 .

docker build -t image:v1.0 -f /opt/dockerfile .

* -f 指定dockerfile
* -t 指定镜像名称

**3.2 Kubernetes基础**

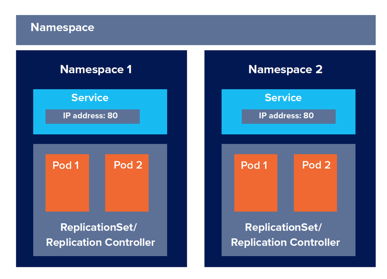
master：控制节点 node: 工作节点



**Namespace**

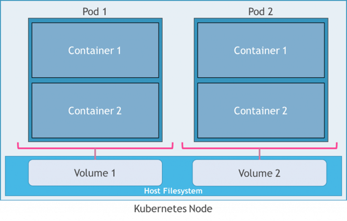
Kubernetes 支持多个虚拟集群，它们底层依赖于同一个物理集群。 这些虚拟集群被称为命名空间。Kubernetes中默认的Namespace是default。

kubectl get namespace 可以查看当前集群的所有名称空间



**Volume**

* emptyDir： 空目录
* hostPath： 本地目录
* nfs： 网络存储

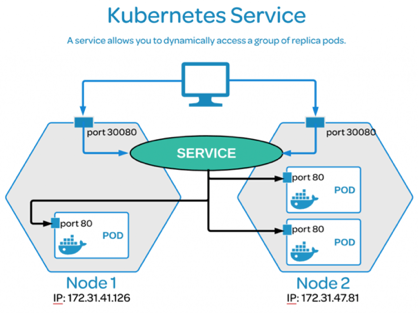


**Deployment**

一个部署控制器提供声明更新pod。在Deployment对象中描述所需的状态，然后Deployment控制器将实际状态以受控的速率更改为所需的状态。可以定义部署以创建新的副本集，或删除现有部署并在新部署中采用其所有资源。

**Service**

服务发现机制。 Kubernetes为Pods提供自己的IP地址和一组Pod的单个DNS名称，并且可以在它们之间进行负载平衡。



**3.3 部署与发布策略**

**滚动更新**

apiVersion: apps/v1beta2

kind: Deployment

metadata:

name: nginx

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 1 ## 最多1个pod处于不可工作状态

maxSurge: 2 ## 升级时可以比预期多出2个pod

minReadySeconds: 5 ## 容器启动后等待5秒

**蓝绿发布**

创建新的部署，然后修改service的标签来将流量指向新的部署。

[root@master b-g]# kubectl get pod

NAME READY STATUS RESTARTS AGE

nginxapp-1.19-bdb8dc6c4-2jfsw 1/1 Running 0 12m

nginxapp-1.19-bdb8dc6c4-79qrh 1/1 Running 0 12m

nginxapp-1.19-bdb8dc6c4-hbwkx 1/1 Running 0 12m

nginxapp-1.20-786464458-4zh5t 1/1 Running 0 91s

nginxapp-1.20-786464458-5bfwq 1/1 Running 0 91s

nginxapp-1.20-786464458-fwgxb 1/1 Running 0 91s

v1.19-blue.yaml

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: nginxapp

version: "1.19"

name: nginxapp-1.19

namespace: default

spec:

replicas: 3

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: nginxapp

version: "1.19"

template:

metadata:

labels:

k8s-app: nginxapp

version: "1.19"

namespace: default

name: nginxapp

spec:

containers:

- name: nginxapp

image: nginx:1.19

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

name: web

service.yaml

kind: Service

apiVersion: v1

metadata:

name: nginxapp

namespace: default

spec:

ports:

- name: web

port: 80

targetPort: 80

selector:

k8s-app: nginxapp

version: "1.19"

v1.20-green.yaml

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: nginxapp

version: "1.20"

name: nginxapp-1.20

namespace: default

spec:

replicas: 3

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: nginxapp

version: "1.20"

template:

metadata:

labels:

k8s-app: nginxapp

version: "1.20"

namespace: default

name: nginxapp

spec:

containers:

- name: nginxapp

image: nginx:20

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

name: web

启动busybox容器调试

kubectl run --restart=Never -it --image \

infoblox/dnstools dnstools

**灰度发布**

<https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#canary>

nginx-1.19

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: nginxapp

version: "1.19"

name: nginxapp-1-19

namespace: default

spec:

replicas: 3

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: nginxapp

version: "1.19"

template:

metadata:

labels:

k8s-app: nginxapp

version: "1.19"

namespace: default

name: nginxapp

spec:

containers:

- name: nginxapp

image: nginx:1.19

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

name: web

---

kind: Service

apiVersion: v1

metadata:

name: nginxapp-1-19

namespace: default

spec:

ports:

- name: web

port: 80

targetPort: 80

selector:

k8s-app: nginxapp

version: "1.19"

nginx-1.20

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: nginxapp

version: "1.20"

name: nginxapp-1-20

namespace: default

spec:

replicas: 3

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: nginxapp

version: "1.20"

template:

metadata:

labels:

k8s-app: nginxapp

version: "1.20"

namespace: default

name: nginxapp

spec:

containers:

- name: nginxapp

image: nginx:1.20

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

name: web

---

kind: Service

apiVersion: v1

metadata:

name: nginxapp-1-20

namespace: default

spec:

ports:

- name: web

port: 80

targetPort: 80

selector:

k8s-app: nginxapp

version: "1.20"

创建资源：

[root@master ingress-gray]# kubectl get pod

NAME READY STATUS RESTARTS AGE

nginxapp-1-19-bdb8dc6c4-6pd66 1/1 Running 0 4s

nginxapp-1-19-bdb8dc6c4-8fqj6 1/1 Running 0 4s

nginxapp-1-19-bdb8dc6c4-txvwp 1/1 Running 0 4s

nginxapp-1-20-786464458-2hp6p 1/1 Running 0 5s

nginxapp-1-20-786464458-6t4sv 1/1 Running 0 5s

nginxapp-1-20-786464458-psmwf 1/1 Running 0 5s

[root@master ingress-gray]# kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.1.0.1 <none> 443/TCP 266d

nginxapp-1-19 ClusterIP 10.1.215.94 <none> 80/TCP 13s

nginxapp-1-20 ClusterIP 10.1.163.181 <none> 80/TCP 14s

spinnaker-demo NodePort 10.1.86.122 <none> 80:30010/TCP 226d

waypoint NodePort 10.1.5.220 <none> 9701:37798/TCP,9702:17531/TCP 168d

创建ingress

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: nginxapp-ingress

namespace: default

spec:

rules:

- host: nginxapp.devops.com

http:

paths:

- path: /

backend:

serviceName: nginxapp-1-19

servicePort: 80

测试：

## 编辑hosts文件

192.168.10.100 nginxapp.devops.com

for i in {1..10};do curl nginxapp.devops.com;done

**基于权重的发布**

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: nginxapp-canary

namespace: default

annotations:

kubernetes.io/ingress.class: nginx

nginx.ingress.kubernetes.io/canary: "true"

nginx.ingress.kubernetes.io/canary-weight: "10"

spec:

rules:

- host: nginxapp.devops.com

http:

paths:

- path: /

backend:

serviceName: nginxapp-1-20

servicePort: 80

**Header报头流量分发**

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: nginxapp-canary

namespace: default

annotations:

kubernetes.io/ingress.class: nginx

nginx.ingress.kubernetes.io/canary: "true"

nginx.ingress.kubernetes.io/canary-by-header: "Region"

nginx.ingress.kubernetes.io/canary-by-header-value: "bj"

spec:

rules:

- host: nginxapp.devops.com

http:

paths:

- path: /

backend:

serviceName: nginxapp-1-20

servicePort: 80

for i in {1..10};do curl -H "region: bj" nginxapp.devops.com;done

**根据cookie流量分发**

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: nginxapp-canary

namespace: default

annotations:

kubernetes.io/ingress.class: nginx

nginx.ingress.kubernetes.io/canary: "true"

nginx.ingress.kubernetes.io/canary-by-cookie: "from"

spec:

rules:

- host: nginxapp.devops.com

http:

paths:

- path: /

backend:

serviceName: nginxapp-1-20

servicePort: 80

[root@master ingress-gray]# curl nginxapp.devops.com --cookie "from=always"

<h1>Welcome to nginx!</h1>

[root@master ingress-gray]# curl nginxapp.devops.com --cookie "from=a"

green

[root@master ingress-gray]# curl nginxapp.devops.com --cookie "from=a"

green

**3.4 Kubectl**

参考：<https://kubernetes.io/zh/docs/reference/kubectl/overview/>

**升级**

kubectl create -f xxx.yaml

kubectl apply -f xxx.yaml

**回滚**

## 查看历史

kubectl rollout history deployment/anyops-devopsdocker-ui

## 查看具体某一个历史版本信息

kubectl rollout history deployment/anyops-devopsdocker-ui --revision=2

## 回滚上个版本

kubectl rollout undo deployment/anyops-devopsdocker-ui -n anyops

## 回滚指定版本

kubectl rollout undo deployment/nginx --to-revision=2

**3.5 Helm**

下载连接：<https://github.com/helm/helm/releases> 一旦你成功安装了Helm客户端，就可以继续使用Helm管理chart和 添加稳定的仓库。

使用Nexus配置helm chart仓库 <https://cloud.tencent.com/developer/article/1647771>

**基本概念**

Chart 代表着 Helm 包。它包含在 Kubernetes 集群内部运行应用程序，工具或服务所需的所有资源定义。

Repository（仓库） 是用来存放和共享 charts 的地方。

Release 是运行在 Kubernetes 集群中的 chart 的实例。

Helm 安装 charts 到 Kubernetes 集群中，每次安装都会创建一个新的 release。你可以在 Helm 的 chart repositories 中寻找新的 chart。

**常用命令**

## 查找chart

helm search hub 从 artifact hub中搜索

helm search repo 从本地的仓库搜索

## 仓库管理

helm repo list

helm repo add

helm repo add dev https://example.com/dev-charts

helm search repo dev

## 安装chart

helm install RELEASE\_NAME CHART\_NAME

helm install happy-panda bitnami/wordpress

### 本地 chart 压缩包

helm install foo foo-0.1.1.tgz

### 解压后的 chart 目录

helm install foo path/to/foo

### 完整的 URL

helm install foo https://example.com/charts/foo-1.2.3.tgz

## 追踪release的状态

helm status happy-panda

## 查看chart中的可配置选项

helm show values CHART\_NAME

helm install -f values.yaml bitnami/wordpress --generate-name

数据传递：

--values /-f 指定yaml文件

--set 命令行方式对值进行覆盖

## 升级

helm upgrade -f panda.yaml happy-panda bitnami/wordpress

## 回滚

helm history [RELEASE] 命令来查看一个特定 release 的修订版本号。

helm rollback

helm rollback [RELEASE] [REVISION]

**自定义charts**

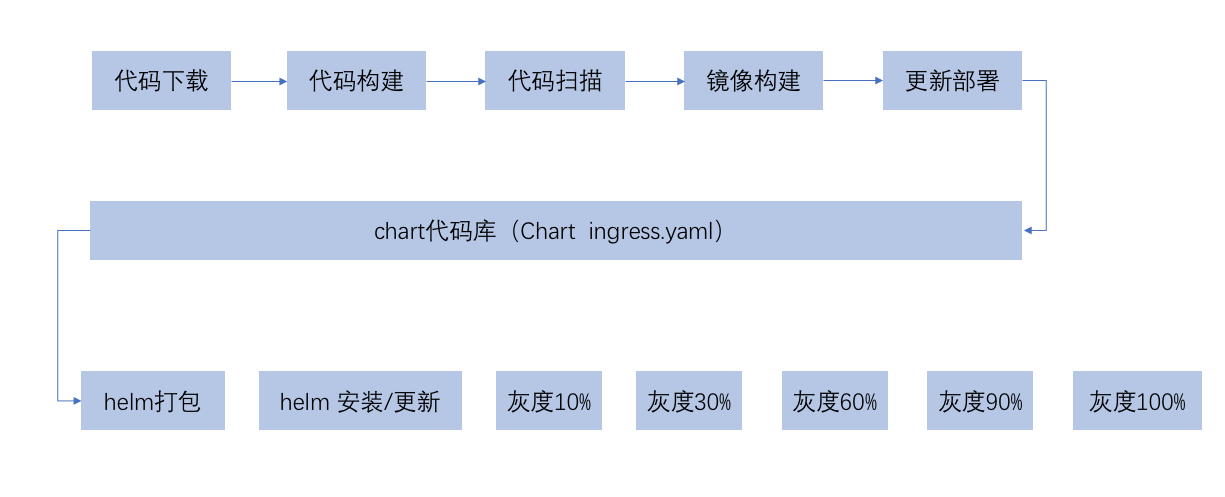
helm create RELEASE\_NAME

helm package nginxapp

helm install nginxapp ./nginxapp-0.1.0.tgz

工作流设计： 前端项目演示

* CI流水线：
  + 构建docker镜像
  + 上传到镜像仓库
  + 更新values.yaml 文件
* CD流水线：
  + 下载chart代码
  + helm package
  + helm install || helm upgrade
  + helm rollback || helm deploy



* V1 旧版本正常运行中， 用户请求流量都在V1
* V2 新版本开始发布， ingress控制， 通过header/cookie/weight 将一部分流量引到V2   10%
* 更新Ingress ，    流量扩大到30%
* 更新Ingress ，    流量扩大到60%
* 更新Ingress ，    流量扩大到90%
* 更新Ingress ，    流量扩大到100%
* 此时测试没问题， 用V2 helm chart 部署替换V1旧版本。

FAQs

