**Day6-07月17日-云主机场景的持续部署实践**

|  |  |  |
| --- | --- | --- |
| **时间** | **标题** | **内容** |
| **上午 下午** | **VM部署实践** | 1. VM虚机/云主机场景下的CD流水线实现 2. 使用Shell编写服务启动脚本 3. Jenkins 集成Ansible 实现应用发布 4. 部署策略实现：蓝绿发布、灰度发布 |
|  | 代码库 | [📎devops02-env-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626513915555-efbede97-9223-4a95-9338-1dfc40b50cb3.zip)  [📎jenkinslib-day6.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626513920237-56a3516f-606b-4e17-aad6-5daabc566b27.zip)  [📎devops02-devops-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626513926913-9541a5ca-9f88-4f33-b77d-3d88bf562f8f.zip)[📎devops02-devops-ui-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626513927016-38fed569-875c-489e-86de-3f5cd680c4f0.zip) |

简短总结

此处为语雀视频卡片，点击链接查看：[day6总结.mp4](https://www.yuque.com/zeyangli/bqsigl/mtswiq#vGFXj)

流水线效果演示：

此处为语雀视频卡片，点击链接查看：[2021-07-17 17-42-42.flv](https://www.yuque.com/zeyangli/bqsigl/mtswiq#cIEoz)

**0. 准备工作项目标准化**

公司里面要使用流水线要做持续集成CI/CD的项目越来越多，这对流水线的设计和开发有不同的要求。我们经常听到用户的反馈：

* 各种**不同语言的技术栈， 如何使流水线适配**呢？ 从不同技术栈维护一套流水线模版，到我们使用共享库进行统一的管理和维护。
* 对于不同的项目，大家管理代码的方式也不同。可能还有一部分用户在使用Svn等不同的版本控制系统。
* 不同的项目，开发模式也不太一样， 编译构建工具不同，发布的方式也有不同的地方...

等等，不止上面的问题。所以在做流水线的使用应该提前把项目团队的规范定义好， 这样后期项目改造后可以直接集成CI/CD流水线。更加便捷。

**跟进项目团队信息**

|  |  |
| --- | --- |
| **信息项** | **描述** |
| 业务简称/编号 | devops02 |
| 开发模式 | 特性分支开发，版本分支发布，主干分支作为最新代码 |
| 项目类型与构建方式 | 前端： vue项目， npm打包， 制品目录 dist |
| 后端：springboot项目， maven打包， 制品目录 target |
| 发布主机环境（vm） | LB： 192.168.1.200 |
| Server： 192.168.1.230~192.168.1.232 |

**制定项目CI/CD规范**

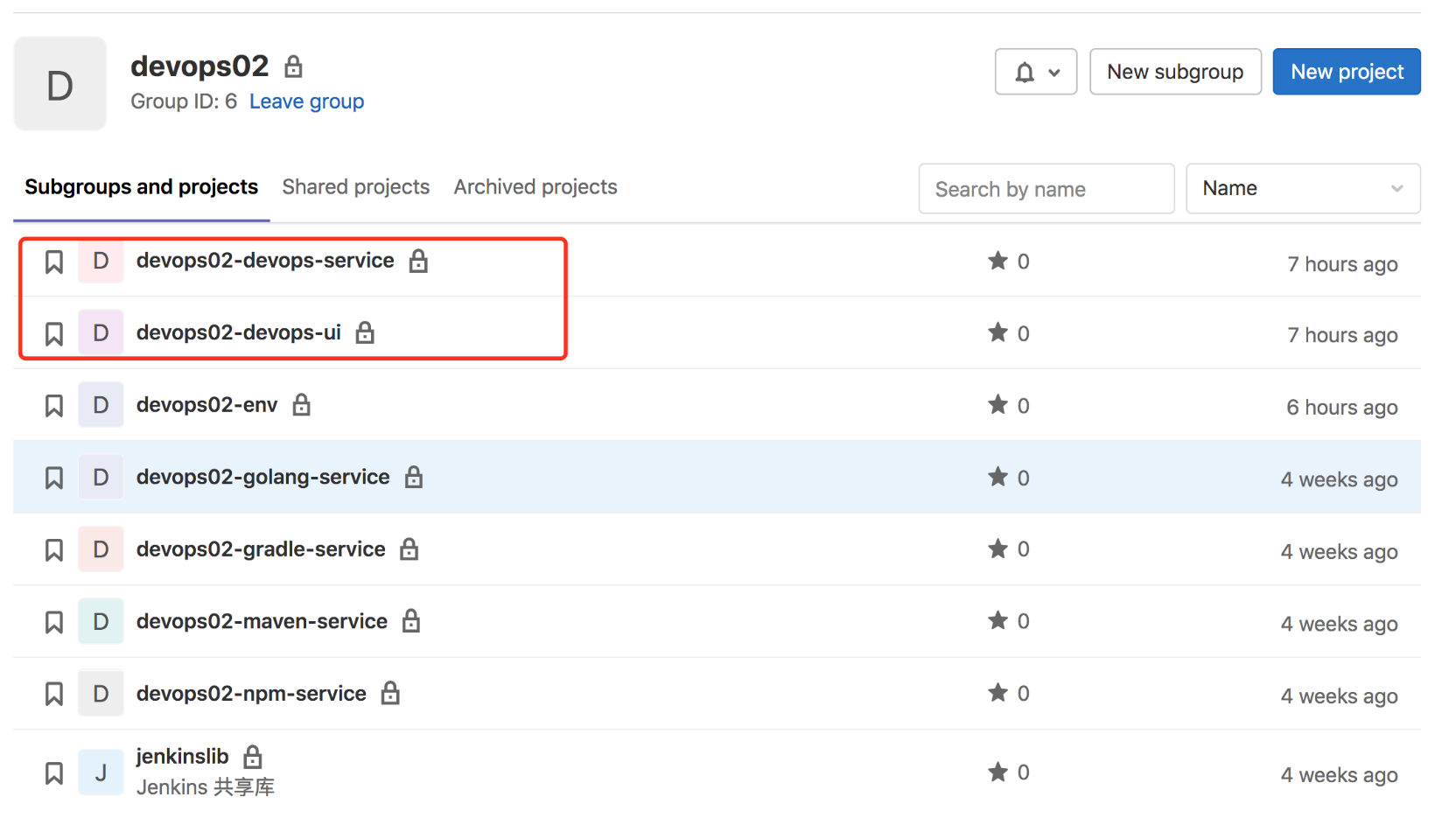
通过上面的信息，我们采用如下规范：

|  |  |
| --- | --- |
| 工具链 |  |
| GitLab 代码库 | 仓库组： devops02 |
| 项目仓库后端 devops02-devops-service 前端 devops02-devops-ui |
| Jenkins作业 | 文件夹: devops02 |
|  | 作业命名： 后端 devops02-devops-service 前端 devops02-devops-ui |
| CI构建规范 | 前端项目采用npm打包后统一放到**dist**目录下, 静态文件以tgz打包。 |
| 后端项目采用maven打包后统一放到**target**目录下，以jar包。 |
|  |  |
| Sonar代码报告 | 前端项目：devops02-devops-ui 后端项目： devops02-devops-service |
| 项目团队可以使用devops02命名的自定义质量规则和质量阈。 |
|  |  |
| Nexus制品库目录 | com/devops02/devops02-devops-service/version/devops02-devops-service-version.jar |
| com/devops02/devops02-devops-ui/version/devops02-devops-ui-version.jar |
| 版本： 分割release分支获取版本号 |
| 发布规范 | 用户输入版本，下载制品库，使用脚本启动服务。 |

**GitLab代码库：**

[📎devops02-devops-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626512395938-b17f192f-e133-435f-ace4-7ab477e05a78.zip)

[📎devops02-devops-ui-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626512400987-71c1b39a-a87e-4b4e-aaad-d59621656430.zip)



**Jenkins作业：**

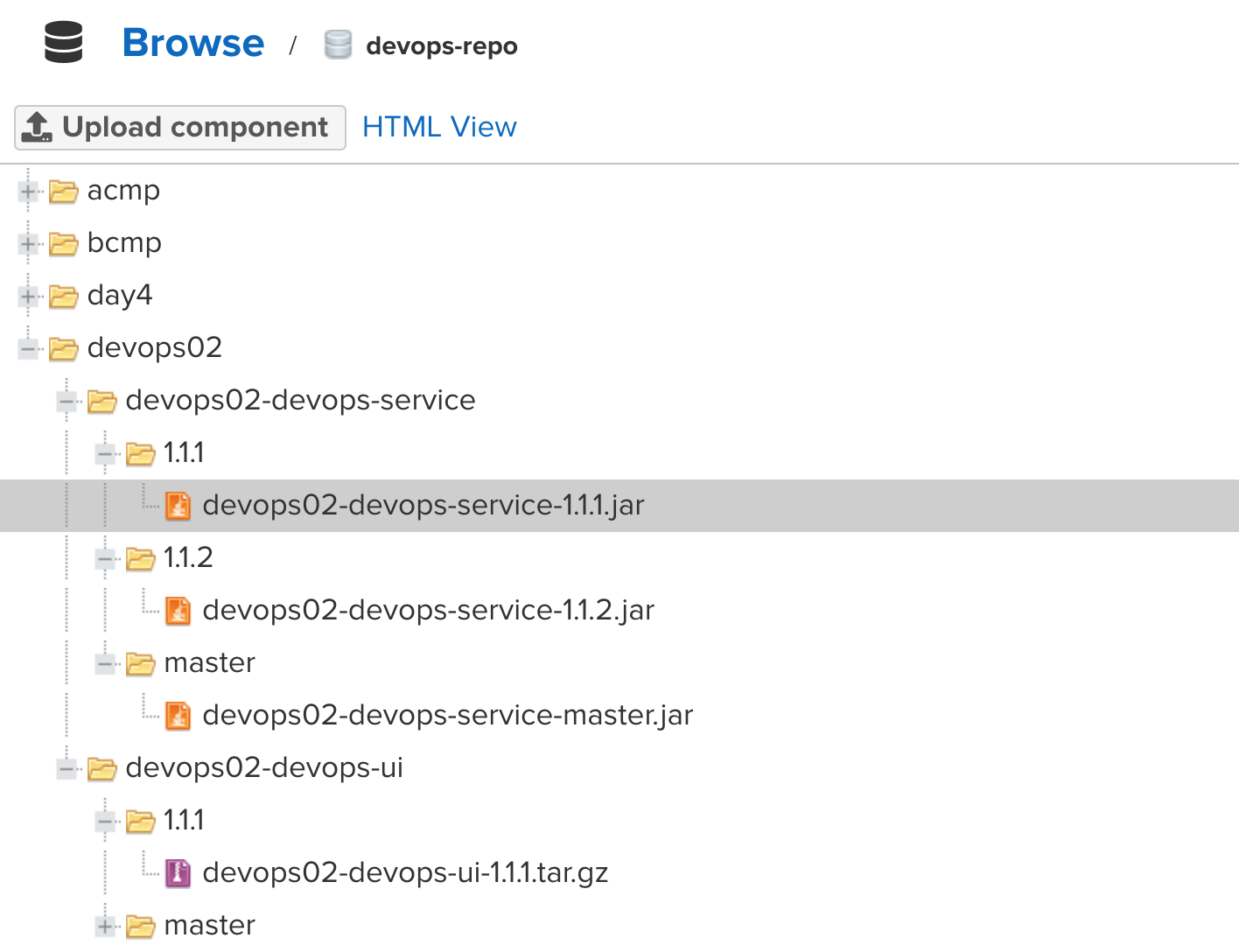


**Sonar代码报告：**

质量规则

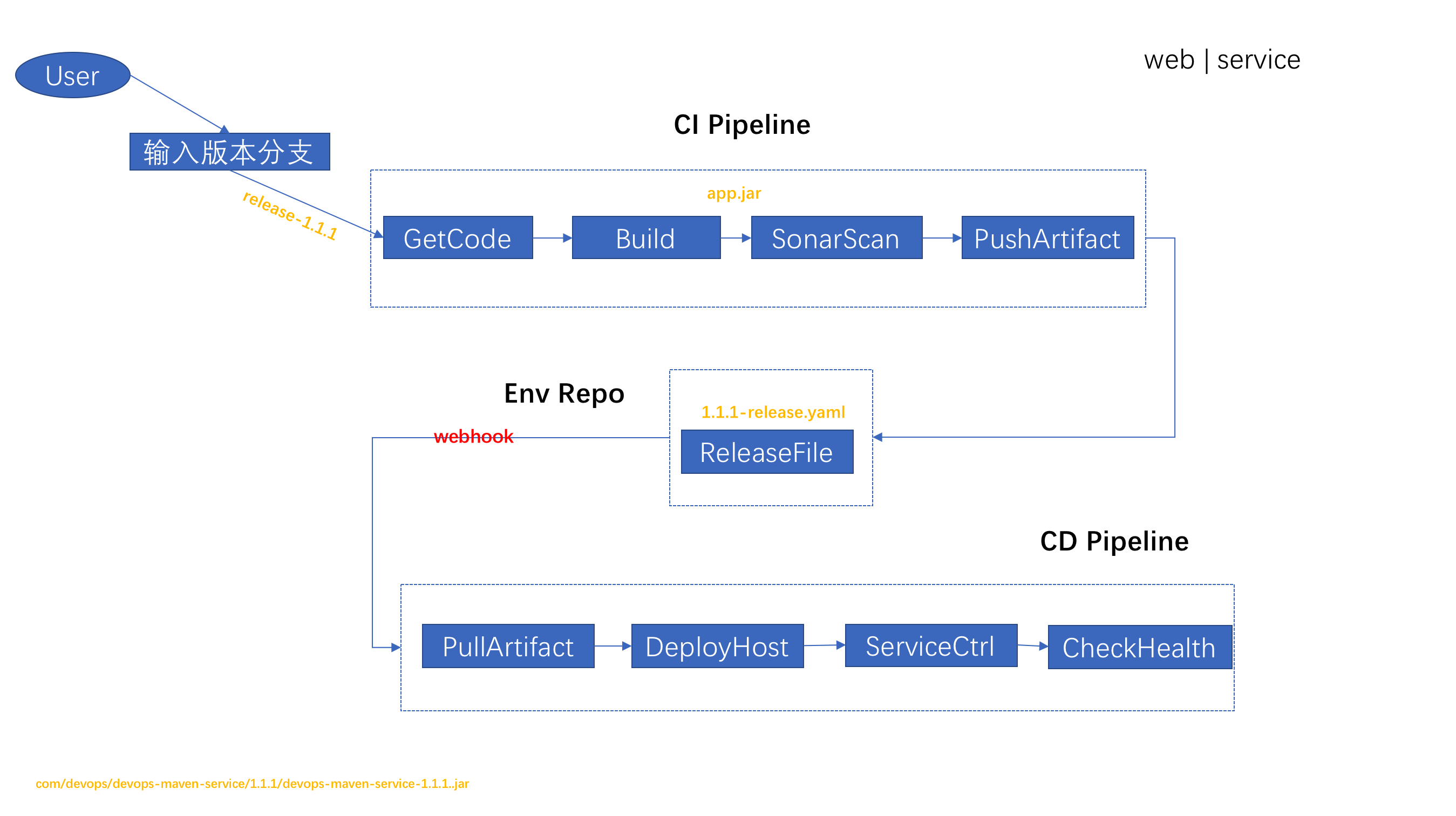


**Nexus制品库目录：(raw)**



**CI/CD流水线设计**

总体目标：

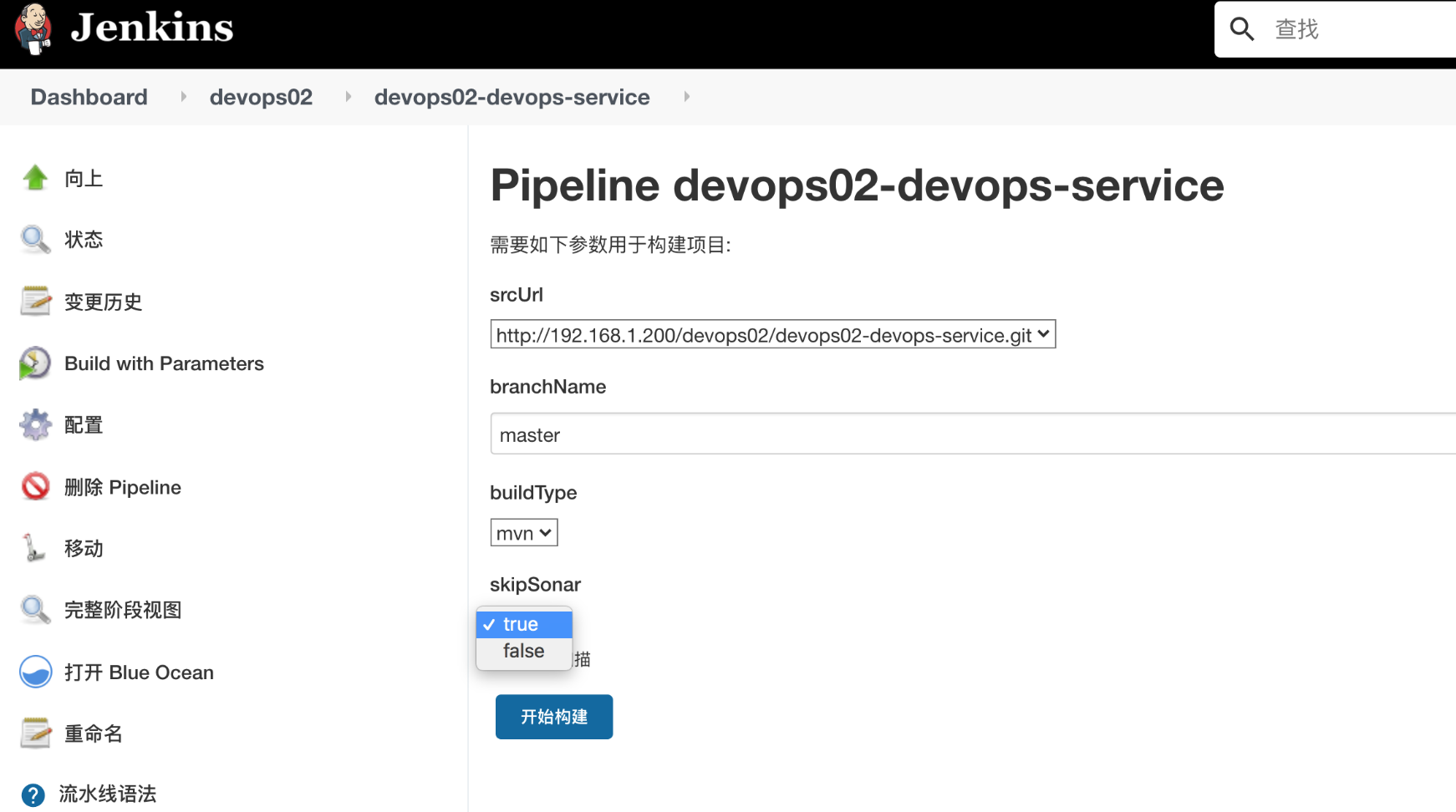


我们将CI和CD分成两条流水线作业。

* CI作业： 用户输入版本分支后下载代码，进行构建扫描最终将制品上传到制品仓库, **生成版本文件**。
* CD作业: 用户输入发布版本和选择要发布的主机IP后，下载制品，将制品和服务启动脚本cp到目标机器的发布目录， 远程执行启动脚本启动服务并进行健康检查。

[📎jenkinslib-day6.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1626512643085-eca7ed8c-d897-4b89-89cf-34ff9fcccb94.zip) **共享库代码**

**1. CI流水线的设计与实践**



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("mylib@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，构建）
* src/org/devops/sonar.groovy 代码扫描工具
* src/org/devops/nexus.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

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 gitHttpURL = "${env.gitHttpURL}" // 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 标准好， 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"]

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（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 ）

// sonar凭据列表

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

//sonar服务器列表

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

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

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

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

**流水线触发方式**

* gitlab提交自动触发
* 手动构建触发

//gitlab触发

try {

if ( "${runOpts}" == "gitlab"){

//自动

// 操作webhook数据

println("${WebHookData}")

webHookData = readJSON text: "${WebHookData}"

env.userName = webHookData["user\_username"]

env.userEmail = webHookData["user\_email"]

env.branchName = webHookData["ref"] - "refs/heads/"

env.srcUrl = webHookData["project"]["git\_http\_url"]

env.commitID = webHookData["checkout\_sha"]

currentBuild.description = "Trigger by user ${env.userName} \n branch: ${env.branchName}"

currentBuild.displayName = env.commitID

}

} catch(e){

println(e)

env.userEmail = "2560350642@qq.com"

currentBuild.description ="Trigger by admin \n branch: ${env.branchName}"

}

**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。

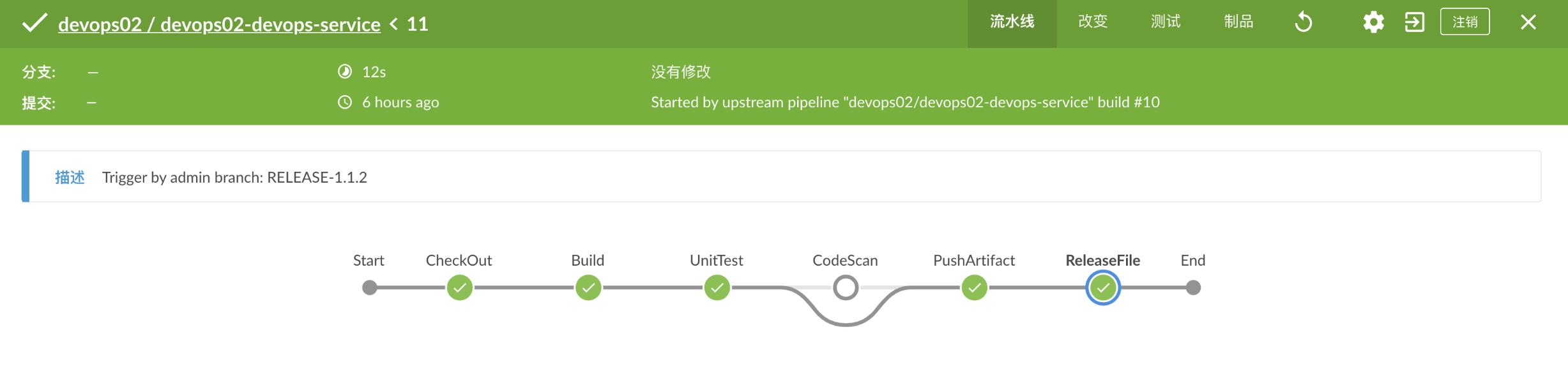
agent { label "build" }

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

// 构建代码

stage("Build"){

steps{

script{

mytools.CustomBuild(buildType,buildMap)

}

}

}

单元测试

//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)

}

}

}

上传制品

//上传制品

stage("PushArtifact"){

steps{

script{

env.artifactUrl = nexus.PushArtifacts(buildType)

println(env.artifactUrl)

}

}

}

上传版本文件

// 生成版本文件

stage("ReleaseFile"){

steps {

script {

//下载版本库文件 devops02-env/release.yaml

response = gitlab.GetRepoFile(23,"release.yaml", "master")

//println(response)

//替换文件中内容

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

println(yamlData)

yamlData.version = "${env.releaseVersion}"

yamlData.artifact = "${env.artifactUrl}"

yamlData.buname = "${buName}"

yamlData.appname = "${appName}"

println(yamlData.toString())

sh "rm -fr test.yaml"

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

newYaml = sh returnStdout: true, script: 'cat test.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(23,"${appName}%2f${branchName}.yaml",base64Content, "master")

} catch(e){

gitlab.CreateRepoFile(23,"${appName}%2f${branchName}.yaml",base64Content, "master")

}

}

}

}

构建后操作

post {

always{

script {

mytools.EmailUser("${env.userEmail}","success")

}

}

failure{

script{

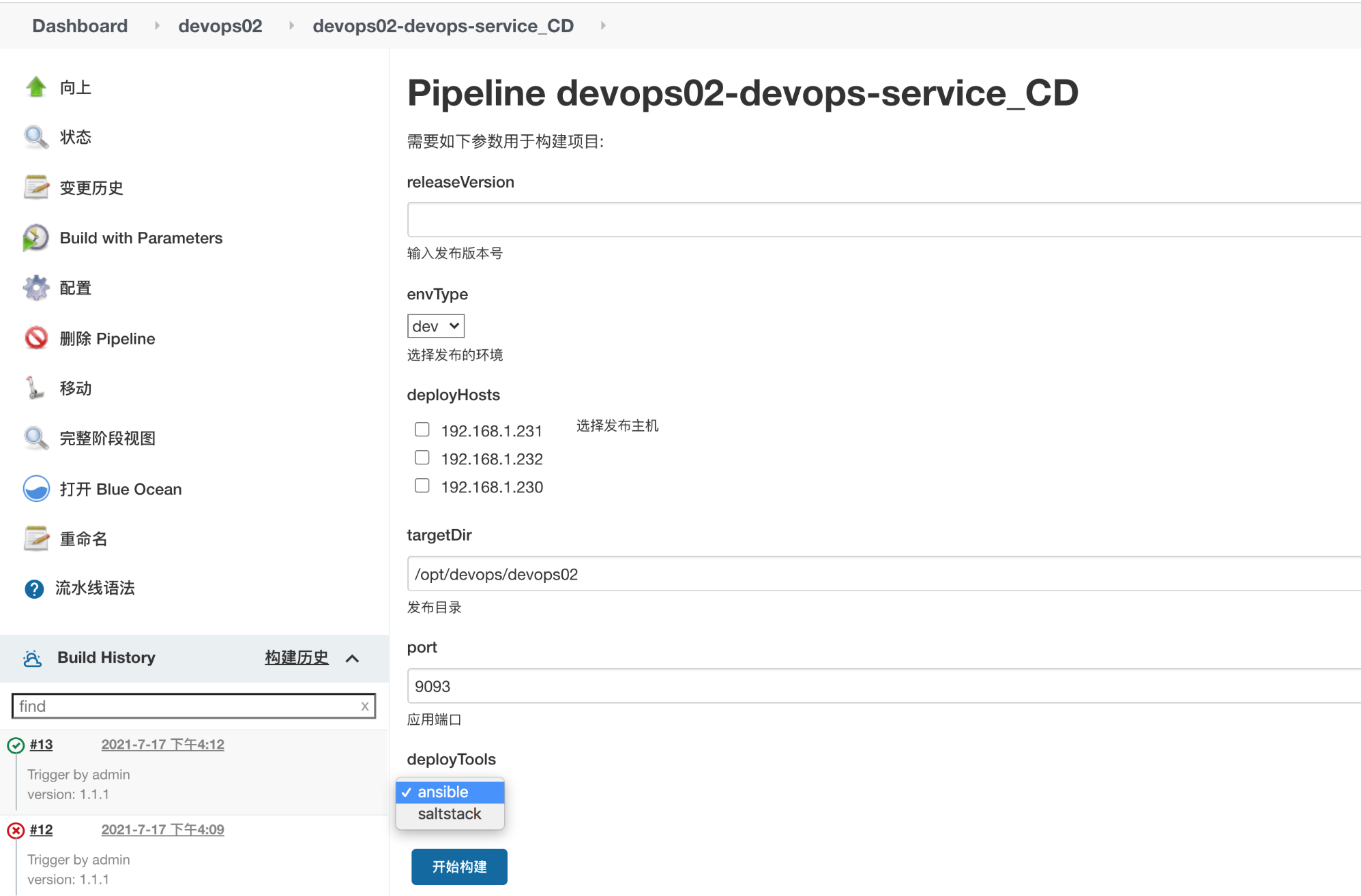
mytools.EmailUser("${env.userEmail}","failure")

}

}

}

**2. CD流水线的设计与实践**



此次CD实践项目我们使用了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("mylib@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
* src/org/devops/deploy.groovy 发布工具（ansible/saltstack）

// 使用共享库

@Library("mylib@master") \_

// 导入类方法

def gitlab = new org.devops.gitlab()

def mytools = new org.devops.mytools()

def deploy = new org.devops.deploy()

定义参数：

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

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

env.userEmail = "2560350642@qq.com"

currentBuild.description ="Trigger by admin \n version: ${env.releaseVersion}"

**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。

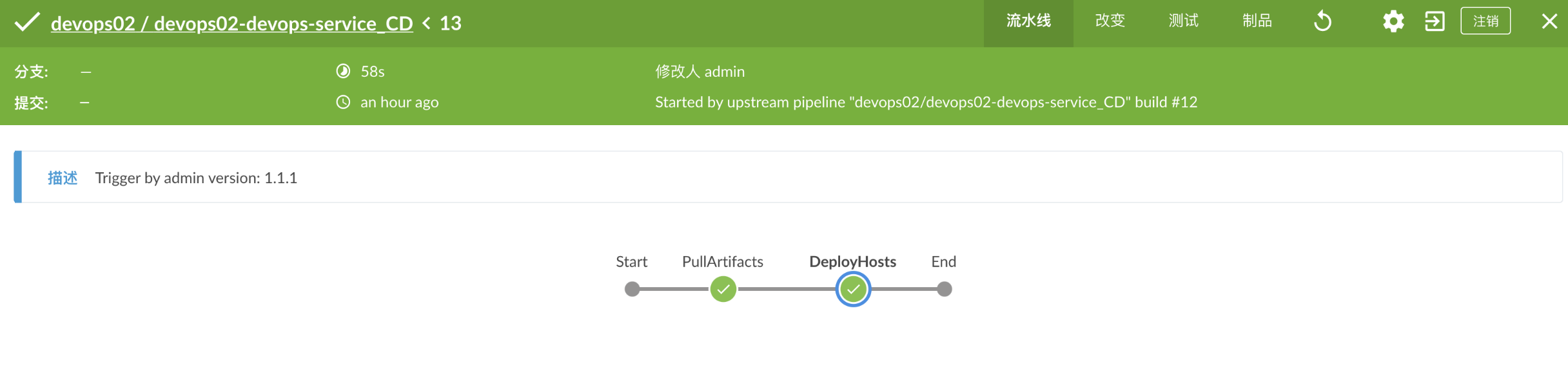
agent { label "build" }

options {

skipDefaultCheckout true

}

**阶段与节点**



下载制品和发布脚本

// 生成版本文件

stage("PullArtifacts"){

steps {

script {

//下载版本库文件 devops02-env/release.yaml

response = gitlab.GetRepoFile(23,"${appName}%2fRELEASE-${env.releaseVersion}.yaml", "master")

//println(response)

//替换文件中内容

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

println(yamlData.artifact.toString())

artifactUrl = yamlData.artifact.toString()

withCredentials([usernamePassword(credentialsId: '111b337e-89a0-4761-9ca3-6ee626ccd7a1', passwordVariable: 'TOKEN', usernameVariable: 'USER')]) {

sh """

wget --http-user=${USER} \

--http-passwd=${TOKEN} \

${artifactUrl} -q

ls -l

"""

}

}

}

}

发布主机

// 发布

stage("DeployHosts"){

steps{

script{

//下载服务脚本

response = gitlab.GetRepoFile(23,"service.sh", "master")

println(response)

writeFile file: 'service.sh', text: "${response}"

sh "ls -a "

if ("${env.deployTool}" == "ansible") {

//ansible(ip)

deploy.DeployByAnsible("${env.deployHosts}", "${env.targetDir}", appName, releaseVersion, "${env.port}" )

}

if ("${env.deployTool}" == "saltstack") {

//saltstack(hostname)

deploy.DeployBySaltStack("${env.deployHosts}", "${env.targetDir}", appName, releaseVersion, "${env.port}")

}

}

}

}

post部分

post {

always{

script {

mytools.EmailUser("${env.userEmail}","success")

cleanWs()

}

}

failure{

script{

mytools.EmailUser("${env.userEmail}","failure")

}

}

}

**3.流水线调试中出现的问题**

## SoanrQube的项目名称不能带有特殊字符'/'

{"errors":[{"msg":"Malformed key for Project: 'anyops/anyops-devops-service'. Allowed characters are alphanumeric, '-', '\_', '.' and ':', with at least one non-digit."}]}

## 设置质量规则时，前端项目的language应该是js或者ts， 而不是npm。

{"errors":[{"msg":"Value of parameter 'language' (npm) must be one of: [java, go, js, ts]"}]}

## 在Sonarqube中找不到buildTools方法， 最后发现没有传递进去。

No such property: buildTools for class: org.devops.sonarqube

扩展： 如何清除工作目录？ 安装Workspace Cleanup插件。在Pipeline 的Post中的always添加CleanWs()

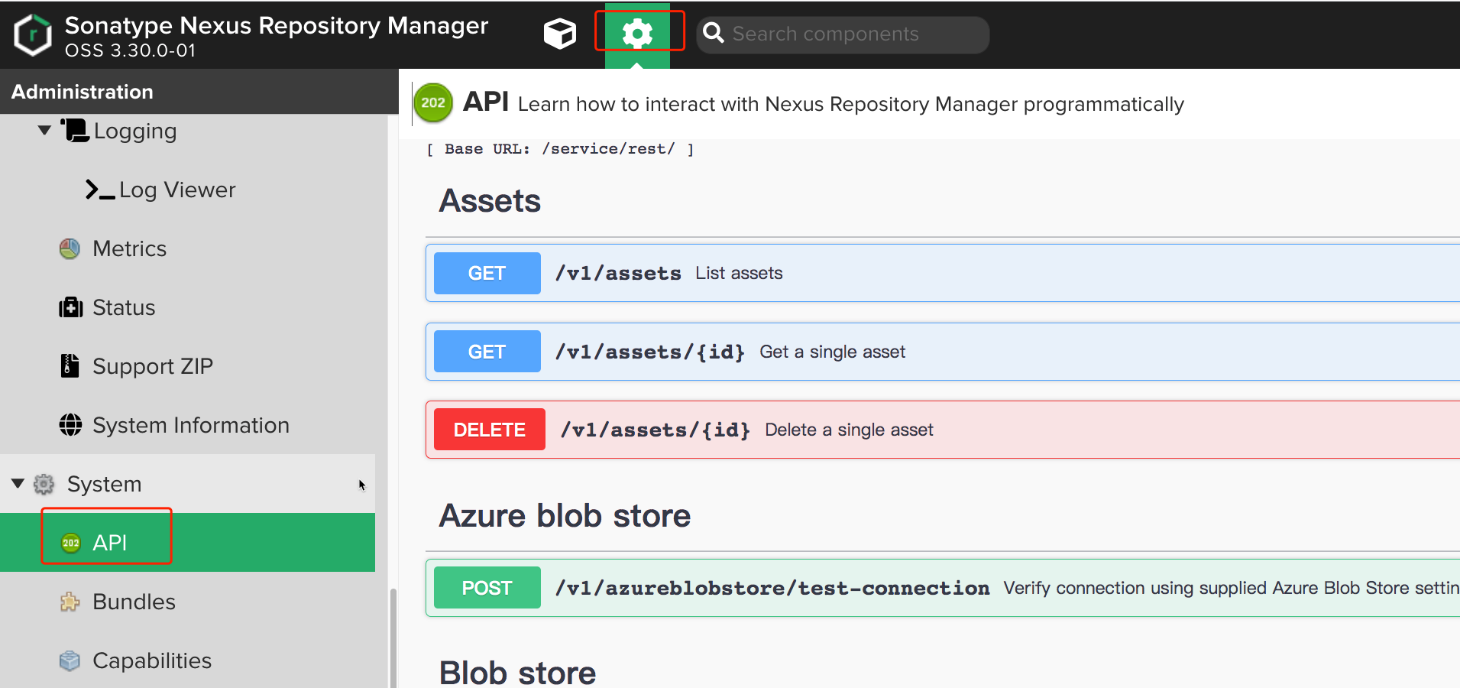


**4. 工程关键技术点讲解**

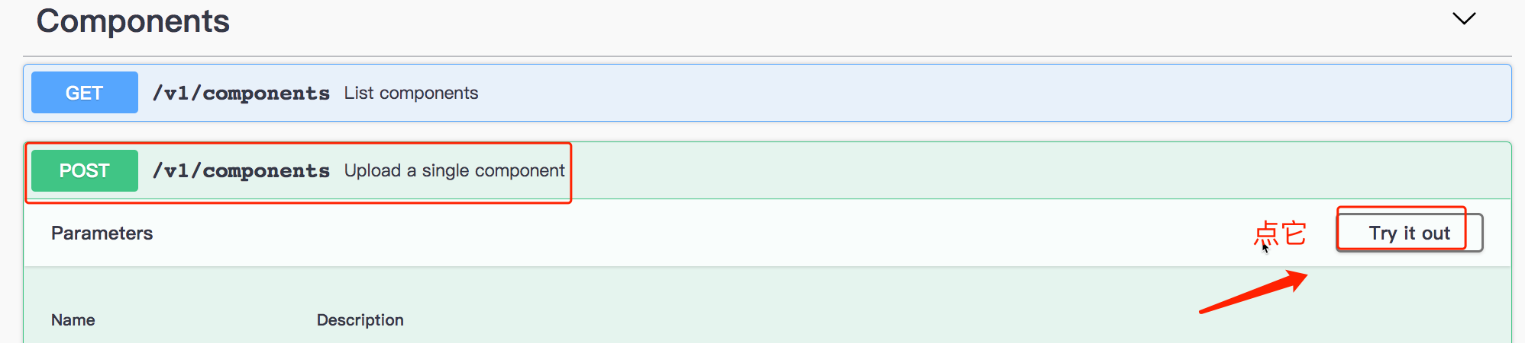
**4.1 Nexus制品上传和下载**

**NexusAPI调试方法**

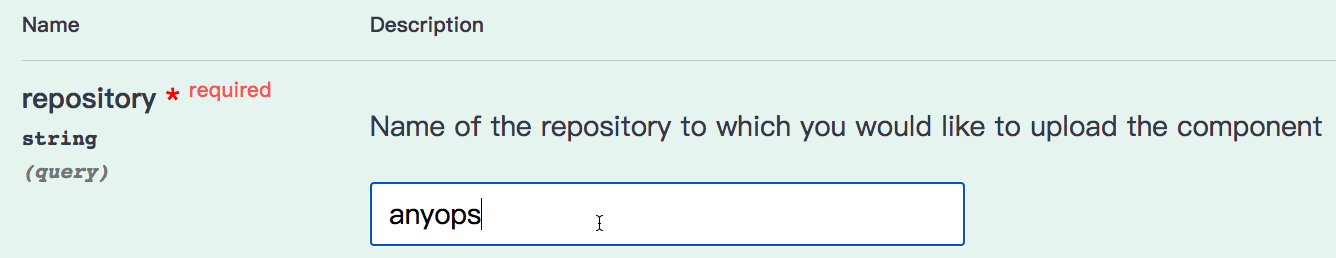
进入设置页面， 找到System > API , 即可进入API调试页面。

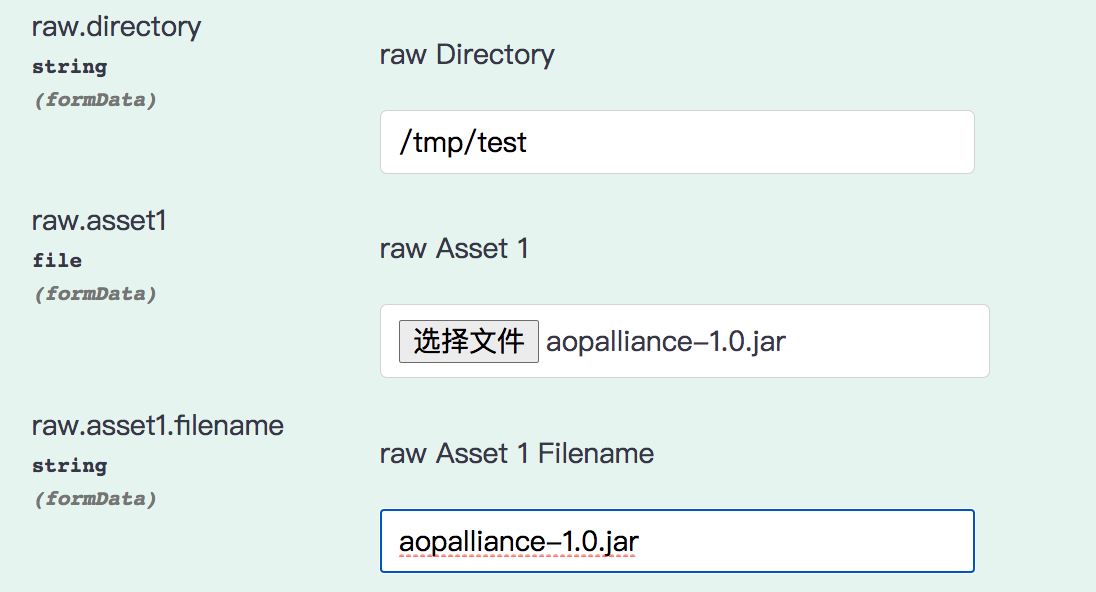


调试API /v1/components, 点击Try it out才能填写信息。

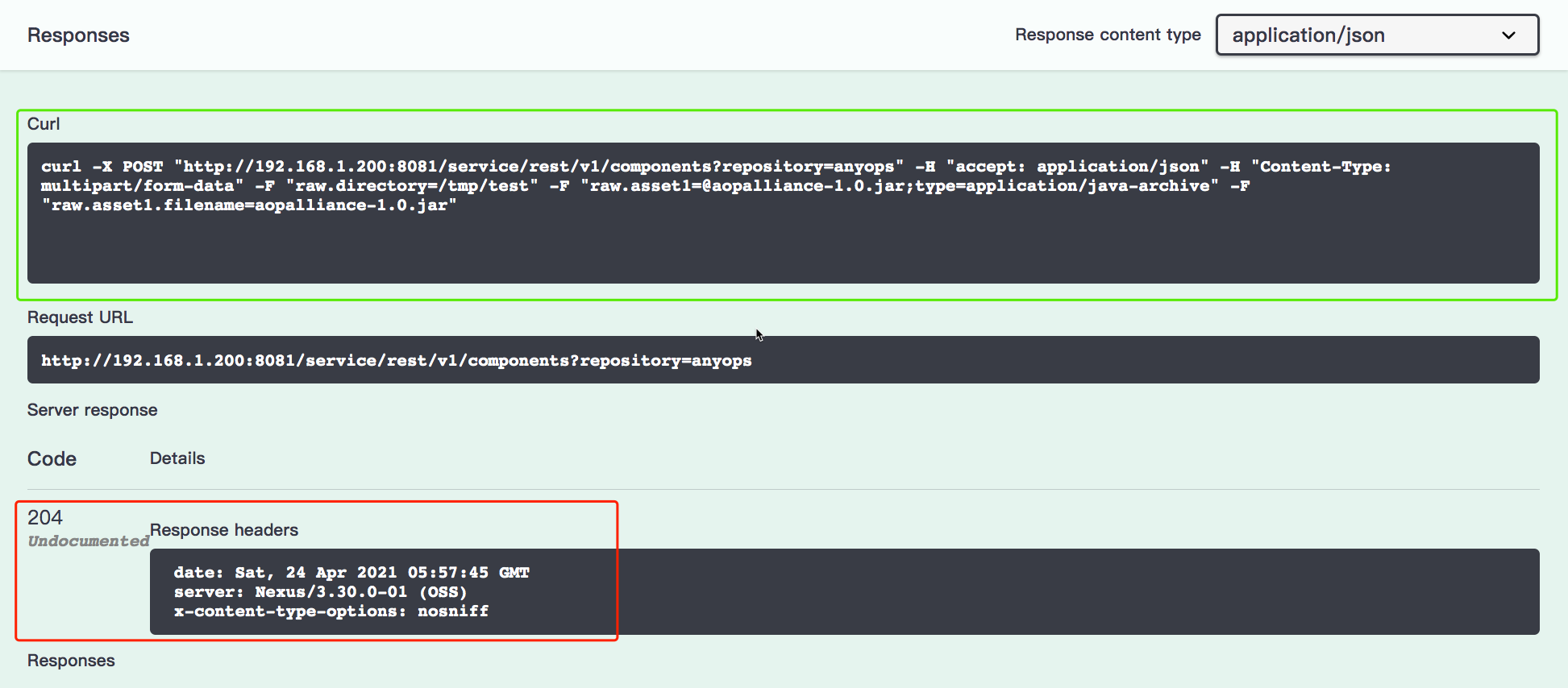


填写参数信息





点击https://cdn.nlark.com/yuque/0/2021/png/2584012/1619243848288-6b26a06b-f722-48bf-8c0f-06afe29db325.png执行操作， 204表示成功。 我们可以复用这里的CURL指令， 最后封装到Jenkins流水线当中。



**上传制品**

curl -u admin:admin123 如果Nexus开启了认证需要配置认证信息才能正常访问。

##PNG

curl -X POST "http://192.168.1.200:8081/service/rest/v1/components?repository=myrepo" -H "accept: application/json" -H "Content-Type: multipart/form-data" -F "raw.directory=/tmp" -F "raw.asset1=@默认标题\_自定义px\_2020-10-01-0.png;type=image/png" -F "raw.asset1.filename=默认标题\_自定义px\_2020-10-01-0.png"

## tar.gz & ZIP

curl -X POST "http://192.168.1.200:8081/service/rest/v1/components?repository=myrepo" -H "accept: application/json" -H "Content-Type: multipart/form-data" -F "raw.directory=/tmp" -F "raw.asset1=@nexus-3.30.0-01-unix.tar.gz;type=application/x-gzip" -F "raw.asset1.filename=aaa.tar.gz"

curl -X POST "http://192.168.1.200:8081/service/rest/v1/components?repository=myrepo" -H "accept: application/json" -H "Content-Type: multipart/form-data" -F "raw.directory=/tmp" -F "raw.asset1=@waypoint\_0.1.5\_linux\_amd64.zip;type=application/x-gzip" -F "raw.asset1.filename=waypoint\_0.1.5\_linux\_amd64.zip"

## Jar file

curl -X POST "http://192.168.1.200:8081/service/rest/v1/components?repository=myrepo" -H "accept: application/json" -H "Content-Type: multipart/form-data" -F "raw.directory=/tmp" -F "raw.asset1=@aopalliance-1.0.jar;type=application/java-archive" -F "raw.asset1.filename=aopalliance-1.0.jar"

**下载制品**

cURL

curl -u admin:admin123 http://192.168.1.200:8081/repository/anyops/com/anyops/a

nyops-devops-service/1.1.1/anyops-devops-service-1.1.1.jar -o anyops-devops-service-1.1.1.jar

Wget

wget --http-user=admin --http-passwd=admin123 http://192.168.1.200:8081/repos

itory/anyops/com/anyops/anyops-devops-service/1.1.1/anyops-devops-service-1.1.1.jar

**4.2 前端后端项目发布**

[Extended Choice Parameter](https://plugins.jenkins.io/extended-choice-parameter)

**前端项目**

* 复制静态文件到nginx站点目录，nginx -s reload

## 进入Web服务器的站点目录下

## 下载包

[root@master html]# curl -u admin:admin123 http://192.168.1.200:8081/repository/anyops/com/anyops/anyops-devops-ui/1.1.1/anyops-devops-ui-1.1.1.tar.gz -o anyops-devops-ui-1.1.1.tar.gz

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 196k 100 196k 0 0 24.0M 0 --:--:-- --:--:-- --:--:-- 24.0M

## 解压包

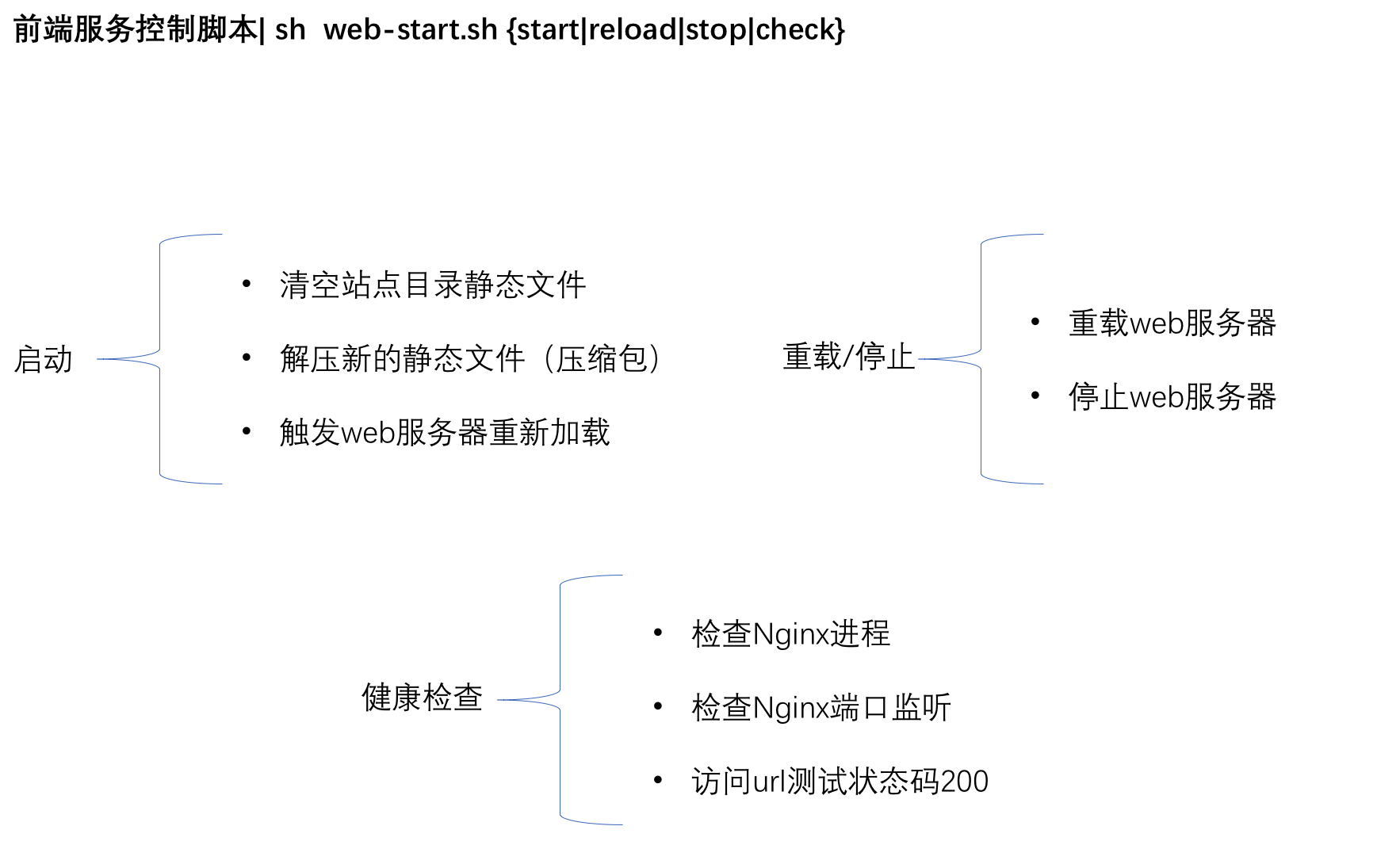
[root@master html]# tar zxf anyops-devops-ui-1.1.1.tar.gz

[root@master html]# ls

anyops-devops-ui-1.1.1.tar.gz index.html static

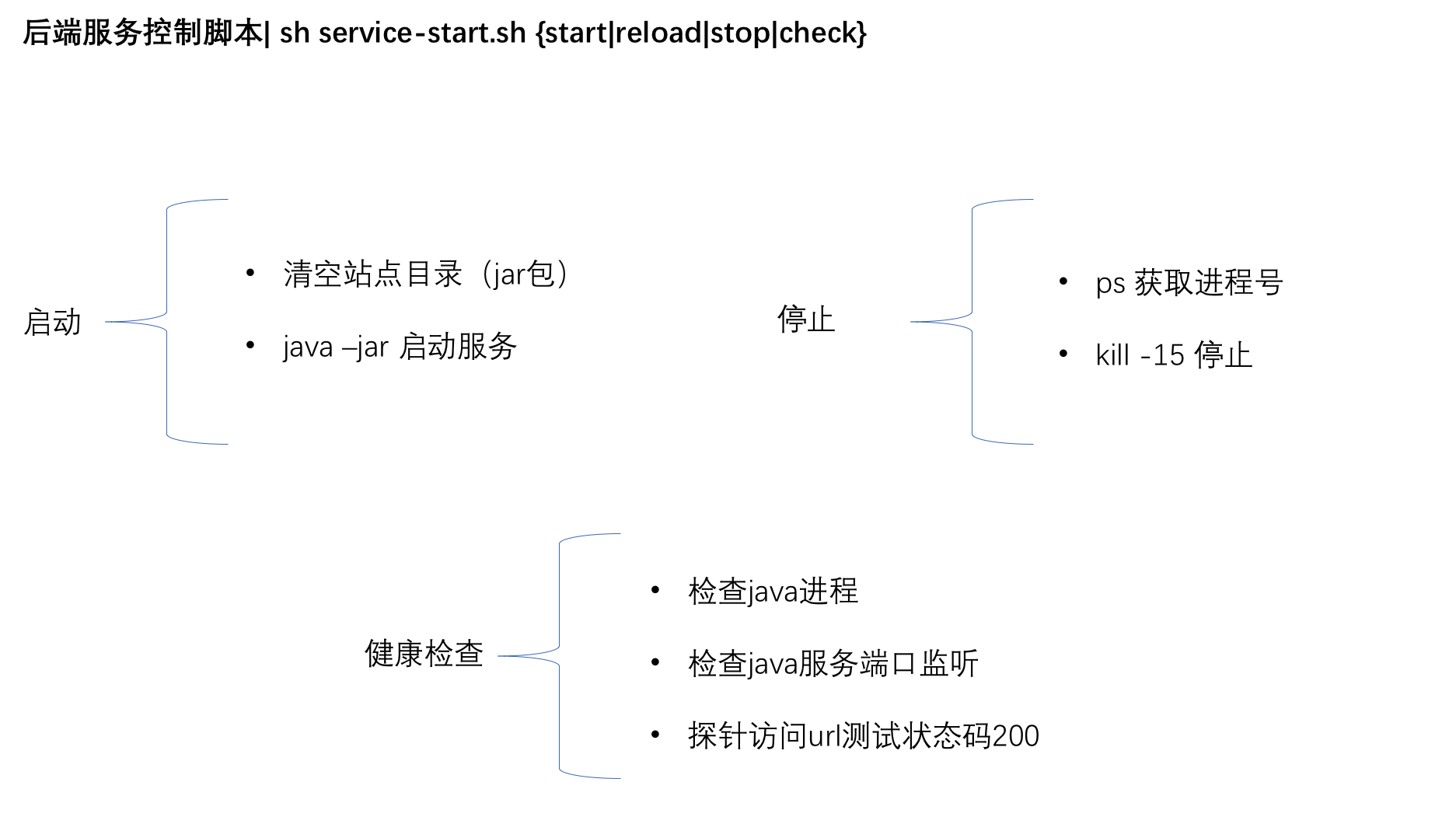
## 触发nginx重载

[root@master html]# nginx -s reload



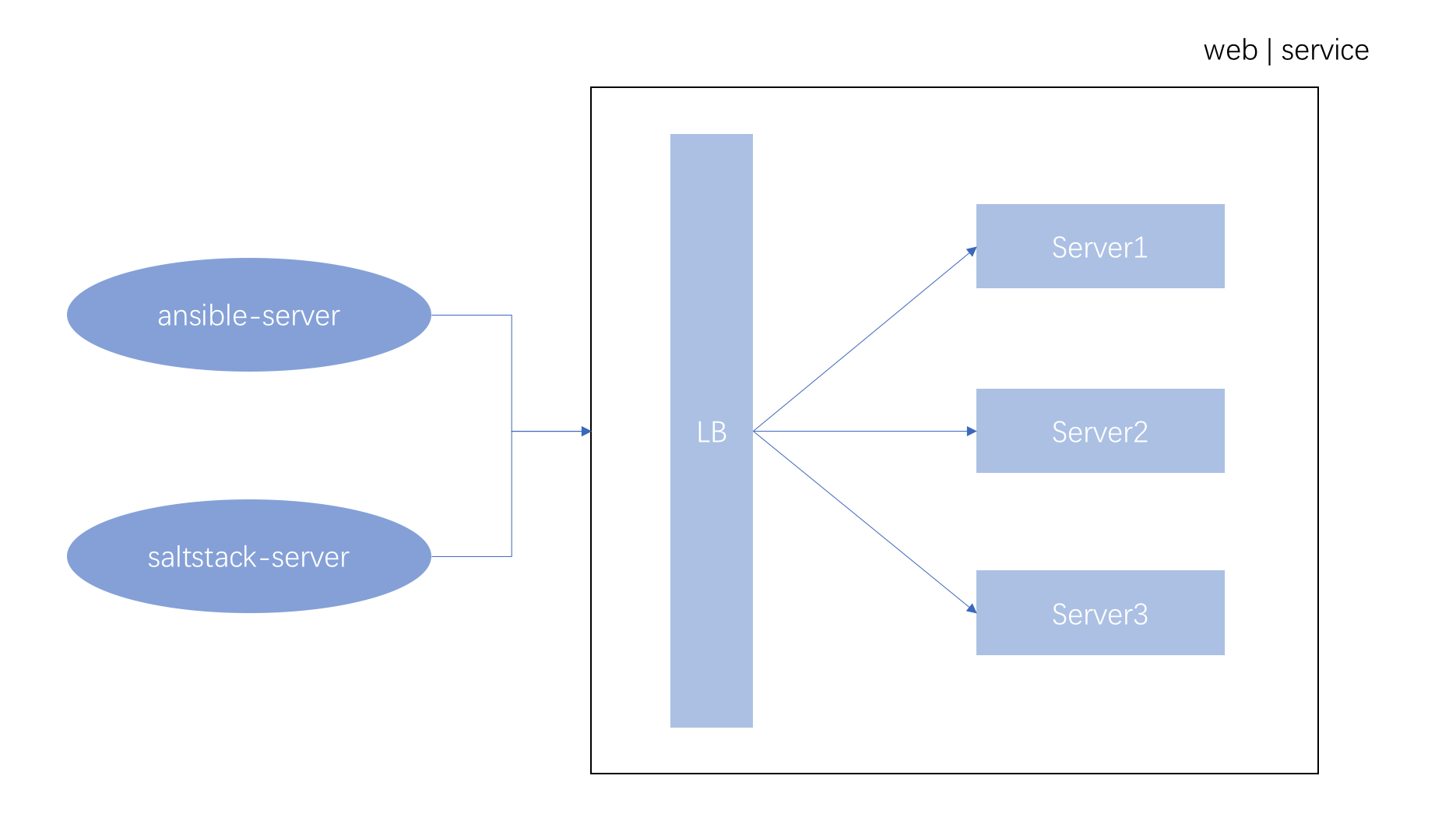
**后端项目**

* 复制jar包到目标目录， 使用nohup java -jar 启动服务。
* nohup java -jar app.jar >output 2>&1 &



Ansible和SaltStack都是基于**Python**开发，Ansible只需要在一台普通的服务器上运行即可，不需要在客户端服务器上安装客户端。Ansible安装使用都很简单，而且基于上千个插件和模块，实现各种软件、平台、版本的管理，支持虚拟容器多层级的部署。

有时候会觉得Ansible比SaltStack执行效率慢，其实并不是软件本身的问题，而是由于SSH服务慢，可以通过优化SSH连接速度和使用Ansible加速模块提高效率。



**4.3 Ansible工具应用**

Ansible是一个IT自动化工具，简单易用。使用OpenSSH进行传输，可以配置系统，部署软件以及编排更高级的IT任务，例如持续部署。

官网：http://www.ansible.com/home

官网文档：http://docs.ansible.com/ansible/index.html

Github地址：https://github.com/ansible

**安装环境**

## yum

yum –y install ansbile

## rpm

$ git clone git://github.com/ansible/ansible.git

$ cd ./ansible

$ make rpm

$ sudo rpm -Uvh ~/rpmbuild/ansible-\*.noarch.rpm

Ansible如何管理主机呢？清单文件（Host Inventory）Ansible的配置文件，对主机进行分类，用来告诉ansible需要管理哪些主机。默认文件为： /etc/ansible/hosts

## 定义主机

192.168.0.22

devops.server.com

jenkins.test.com

## 定义主机组

[webservers]

test.server.com

test2.server.com

**命令与指令**

检查主机连通性

Ansible命令格式

ansible <hosts> [options]

检查ansible安装环境： ansible all -m ping -u root

执行命令： ansible all -a "/bin/echo hello world"

复制文件

ansible webserver -m copy -a "src=/etc/passwd dest=/opt/passwd"

yum安装软件

ansible webserver -m yum -a "name=lrzsz"

添加系统用户

ansible webserver -m user -a "name=zhangsan password=123456"

服务控制

ansible webserver -m service -a "name=sshd state=started"

**4.4 SaltStack工具应用(扩展)**

官方文档：<https://docs.saltproject.io/en/latest/contents.html>

SaltStack基于Python开发的一套C/S架构配置管理工具。使用SSL证书签发的方式进行认证管理。

可以对主机进行集中管理、文件发布、数据采集、软件包管理等配置管理操作。有利于运维人员提高工作效率，规范业务配置和操作。是常见的自动化运维利器。

Saltstack组成：

* Master是服务端，用于操作调度Minion。
* Minion是客户端,接收来自Master的指令并执行。

服务端口：

* 4505 Master和Minion的认证通信端口。（当客户端启动后,会主动向Master端注册）
* 4506 Master与Minion指令交互端口。

**配置环境**

一台salt-master， 多台salt-minion。

sudo rpm --import https://repo.saltproject.io/py3/redhat/8/x86\_64/latest/SALTSTACK-GPG-KEY.pub

curl -fsSL https://repo.saltproject.io/py3/redhat/8/x86\_64/latest.repo | sudo tee /etc/yum.repos.d/salt.repo

yum install salt-master

yum install salt-minion

## master start

[root@zeyang-nuc-service yum.repos.d]# systemctl start salt-master

[root@zeyang-nuc-service yum.repos.d]# systemctl status salt-master

● salt-master.service - The Salt Master Server

Loaded: loaded (/usr/lib/systemd/system/salt-master.service; disabled; vendor preset: disabled)

Active: active (running) since Fri 2021-04-23 21:22:29 HKT; 5s ago

Docs: man:salt-master(1)

file:///usr/share/doc/salt/html/contents.html

https://docs.saltstack.com/en/latest/contents.html

Main PID: 169600 (salt-master)

Tasks: 32 (limit: 203502)

Memory: 252.0M

CGroup: /system.slice/salt-master.service

├─169600 /usr/bin/python3.6 /usr/bin/salt-master

├─169614 /usr/bin/python3.6 /usr/bin/salt-master

├─169619 /usr/bin/python3.6 /usr/bin/salt-master

├─169622 /usr/bin/python3.6 /usr/bin/salt-master

├─169623 /usr/bin/python3.6 /usr/bin/salt-master

├─169624 /usr/bin/python3.6 /usr/bin/salt-master

├─169625 /usr/bin/python3.6 /usr/bin/salt-master

├─169632 /usr/bin/python3.6 /usr/bin/salt-master

├─169633 /usr/bin/python3.6 /usr/bin/salt-master

├─169634 /usr/bin/python3.6 /usr/bin/salt-master

├─169635 /usr/bin/python3.6 /usr/bin/salt-master

├─169636 /usr/bin/python3.6 /usr/bin/salt-master

└─169637 /usr/bin/python3.6 /usr/bin/salt-master

Apr 23 21:22:28 zeyang-nuc-service systemd[1]: Starting The Salt Master Server...

Apr 23 21:22:29 zeyang-nuc-service systemd[1]: Started The Salt Master Server.

[root@zeyang-nuc-service yum.repos.d]# netstat -anlpt | grep 450

tcp 0 0 0.0.0.0:4505 0.0.0.0:\* LISTEN 169619/python3.6

tcp 0 0 0.0.0.0:4506 0.0.0.0:\* LISTEN 169625/python3.6

salt-minion

[root@master ~]# vi /etc/salt/minion

master: 192.168.1.200

[root@master ~]# systemctl start salt-minion

[root@master ~]# systemctl status salt-minion

● salt-minion.service - The Salt Minion

Loaded: loaded (/usr/lib/systemd/system/salt-minion.service; disabled; vendor preset: disabled)

Active: active (running) since Fri 2021-04-23 09:38:59 EDT; 6s ago

Docs: man:salt-minion(1)

file:///usr/share/doc/salt/html/contents.html

https://docs.saltstack.com/en/latest/contents.html

Main PID: 33505 (salt-minion)

Tasks: 8 (limit: 24986)

Memory: 75.6M

CGroup: /system.slice/salt-minion.service

├─33505 /usr/bin/python3.6 /usr/bin/salt-minion

├─33510 /usr/bin/python3.6 /usr/bin/salt-minion

└─33514 /usr/bin/python3.6 /usr/bin/salt-minion

4月 23 09:38:58 master.zy.com systemd[1]: Starting The Salt Minion...

4月 23 09:38:59 master.zy.com systemd[1]: Started The Salt Minion.

4月 23 09:39:00 master.zy.com salt-minion[33505]: [ERROR ] The Salt Master has cached the public key for t>

salt-master 认证minion

[root@zeyang-nuc-service yum.repos.d]# salt-key -L

Accepted Keys:

zeyang-nuc-service

Denied Keys:

Unaccepted Keys:

master.zy.com

Rejected Keys:

[root@zeyang-nuc-service yum.repos.d]# salt-key -a master.zy.com

The following keys are going to be accepted:

Unaccepted Keys:

master.zy.com

Proceed? [n/Y] y

Key for minion master.zy.com accepted.

[root@zeyang-nuc-service yum.repos.d]# salt-key -L

Accepted Keys:

master.zy.com

zeyang-nuc-service

Denied Keys:

Unaccepted Keys:

Rejected Keys:

salt-api配置(扩展)

useradd saltapi

passwd saltapi ## 设置密码 123456

yum -y install salt-api

vi /etc/salt/master.d/api.conf

external\_auth:

pam:

saltapi:

- .\*

- '@wheel'

- '@runner'

rest\_cherrypy:

port: 8000

disable\_ssl: true

host: 0.0.0.0

systemctl restart salt-master

systemctl restart salt-api

## API(success)

[root@zeyang-nuc-service master.d]# curl -k http://127.0.0.1:8000/login -H "Accept: application/x-yaml" -d username='saltapi' -d password='123456' -d eauth='pam'

return:

- eauth: pam

expire: 1619230016.3118818

perms:

- .\*

- '@wheel'

- '@runner'

start: 1619186816.3118815

token: 01049ff981bc7dae25fdd27875e09afd6cd34989

user: saltapi

**操作指令**

## 模块使用

salt '\*' sys.doc cmd.run

## 分发文件

salt-cp 'node01.zy.com' /etc/hosts /tmp/hosts

salt-cp -L "node01.zy.com,node02.zy.com" /etc/hosts /tmp/hosts

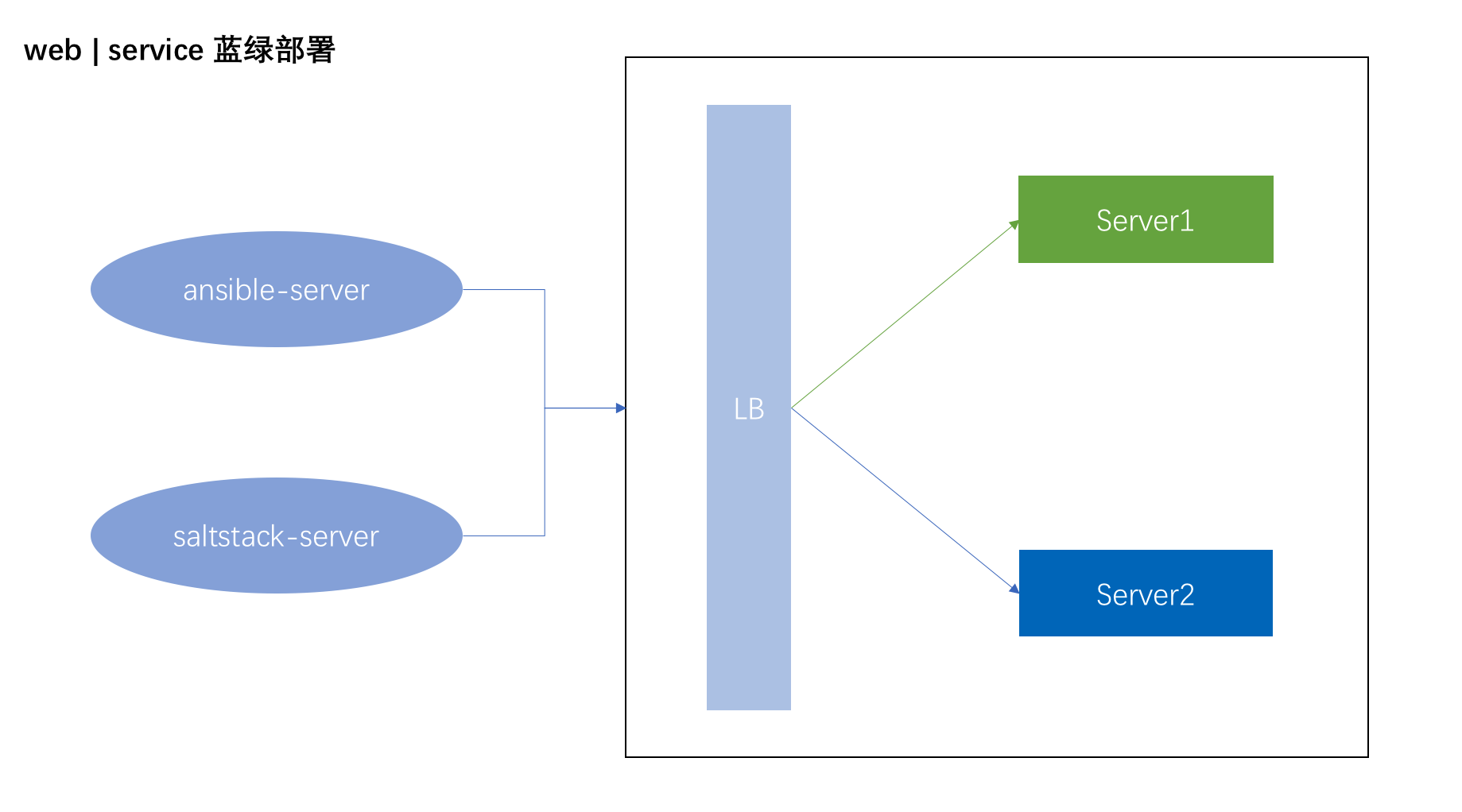
## 执行命令

salt '\*' cmd.run "ls -l | awk '/foo/{print \\$2}'"

**4.5 应用发布与回滚策略**

**蓝绿发布**

环境存在两个版本，蓝版本和绿版本同时存在，部署新版本然后进行测试，将流量切到新版本，最终实际运行的只有一个版本（蓝/绿）。好处是无需停机，并且发布风险较小。



nginx upstream模块实现:

upstream webservers {

server 192.168.1.253:8099 weight=100;

server 192.168.1.252:8099 down;

}

server {

listen 8017;

location / {

proxy\_pass http://webservers;

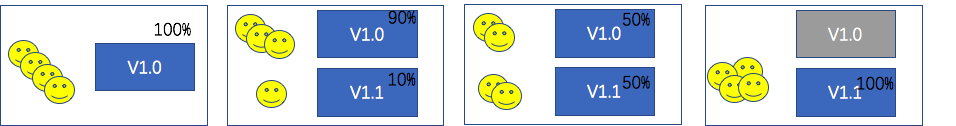
}

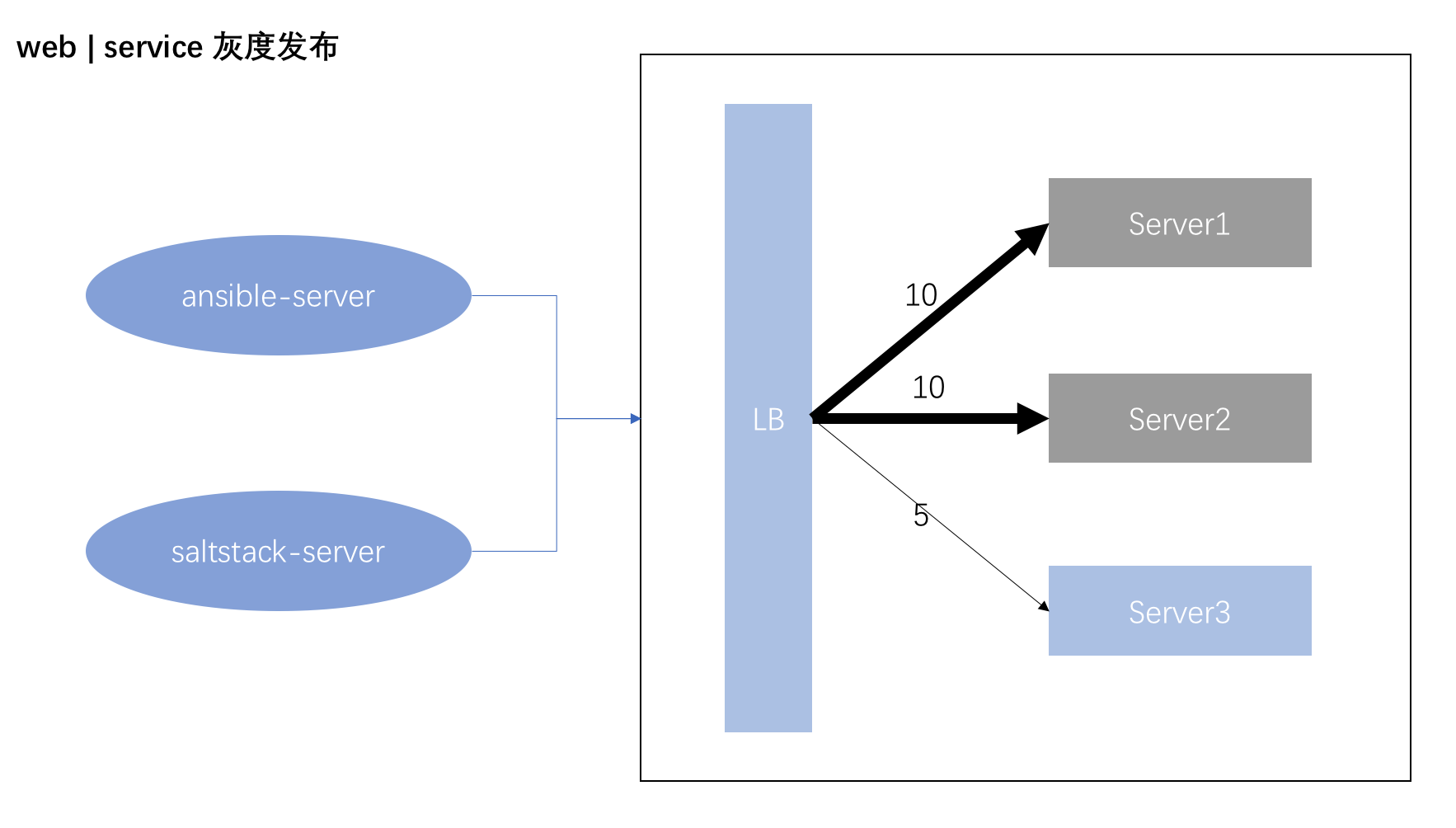
}

nginx -s reload

**灰度发布**

将发行版发布到一部分用户或服务器的一种模式。这个想法是首先将更改部署到一小部分服务器，进行测试，然后将更改推广到其余服务器。一旦通过所有运行状况检查，当没有问题时，所有的客户将被路由到该应用程序的新版本，而旧版本将被删除。





nginx 权重模拟:

upstream webservers {

server 192.168.1.223:8099 weight=100;

server 192.168.1.222:8099 weight=100;

server 192.168.1.221:8099 weight=100;

}

server {

listen 8017;

location / {

proxy\_pass http://webservers;

}

}

nginx -s reload

**版本回滚**

* 版本一直升级，则无需回滚。
* 选择旧版本文件，进行发布。

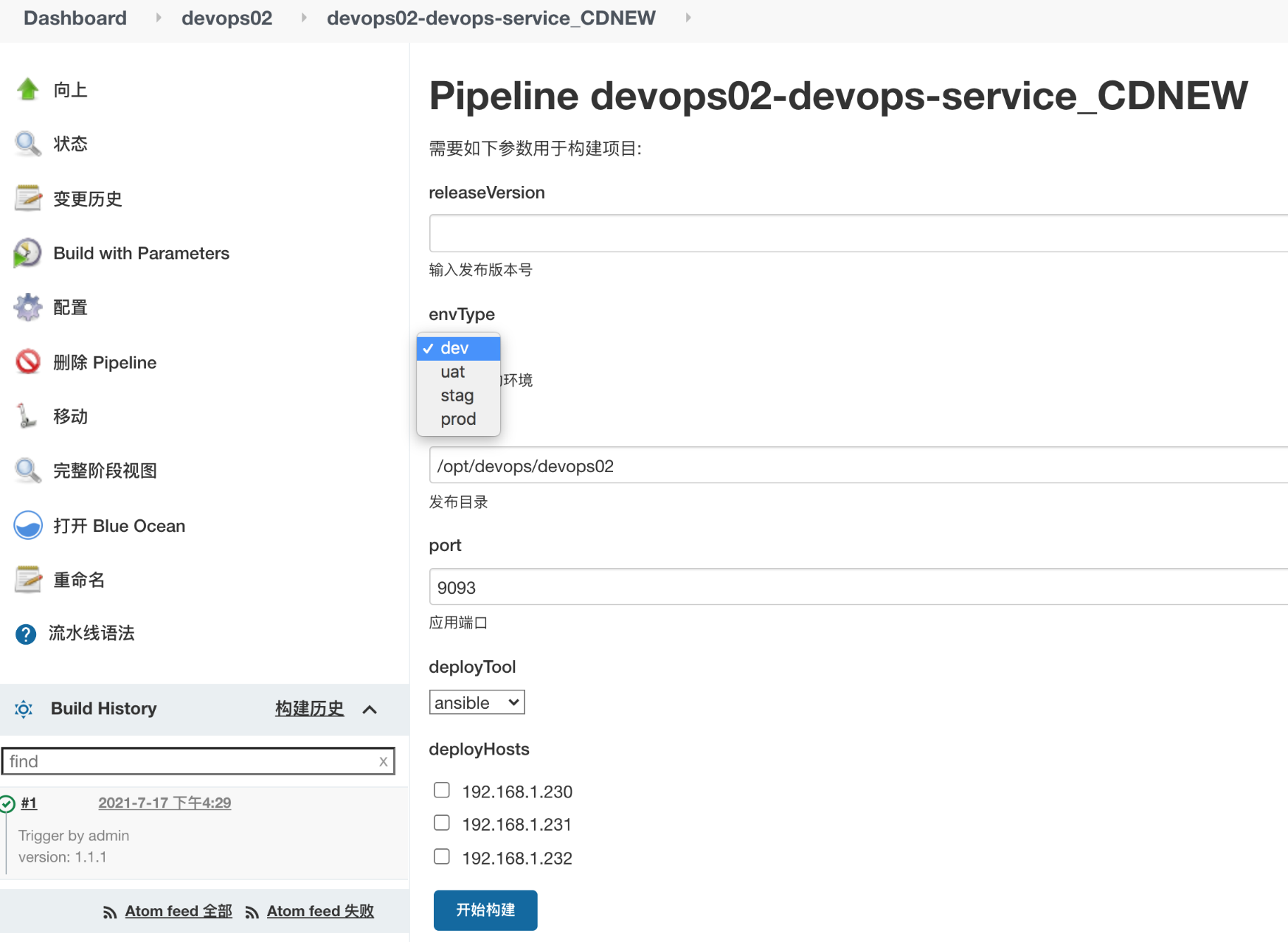
**4.6 版本文件的生成**

Scripts not permitted to use staticMethod org.codehaus.groovy.runtime.EncodingGroovyMethods encodeBase64 byte[]. Administrators can decide whether to approve or reject this signature.

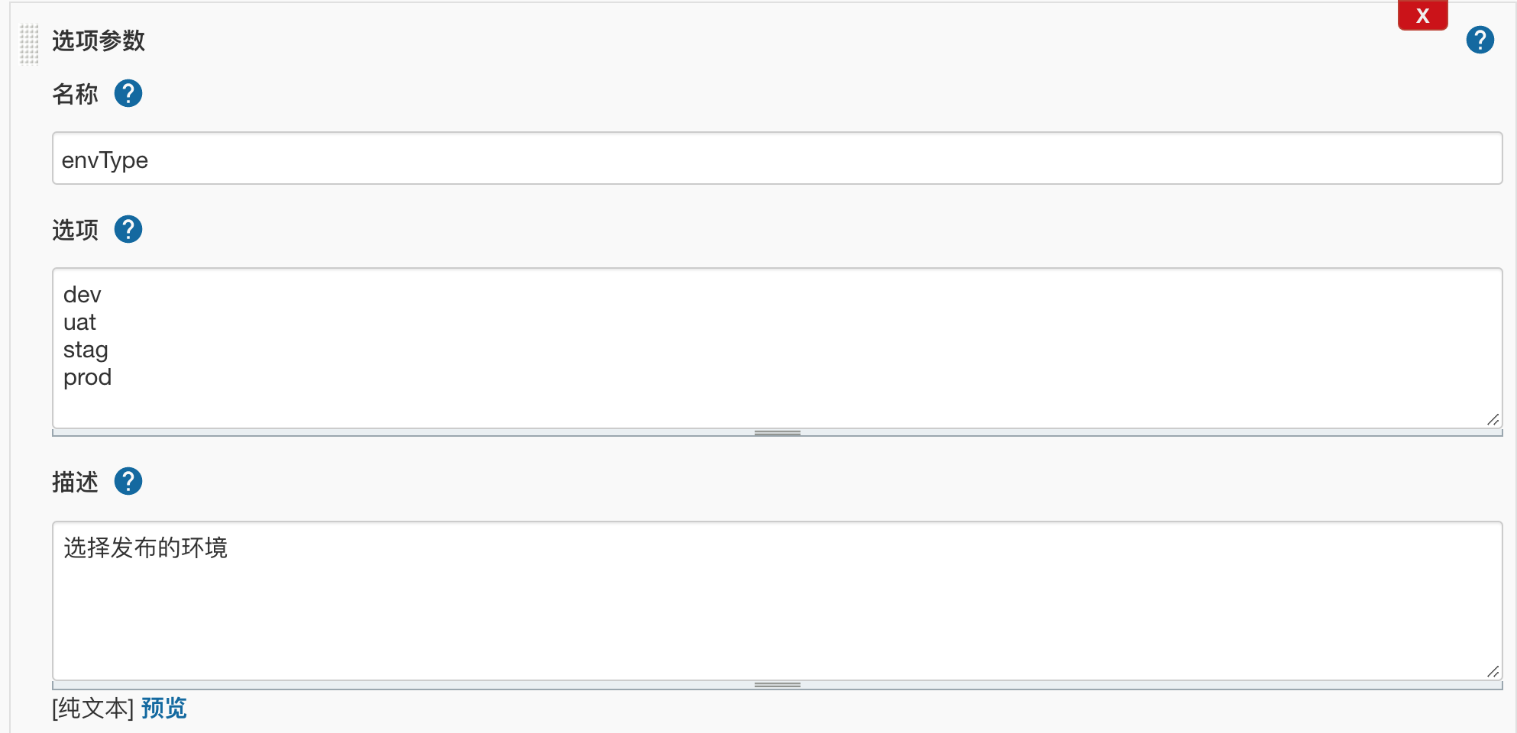


**5. 扩展ActiveChoice参数实践**

* 根据不同的环境和发布工具，动态展示主机参数。

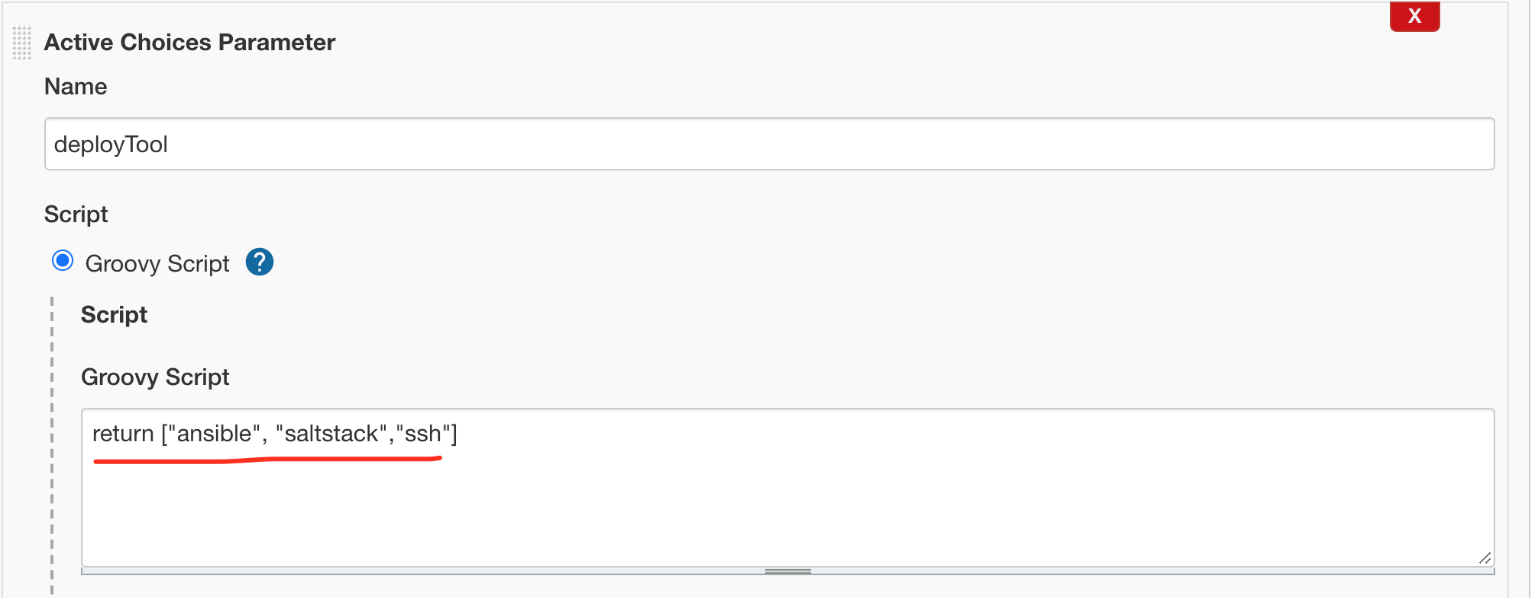


定义发布环境参数



定义发布工具参数

return ["ansible", "saltstack","ssh"]



单选类型



定义发布主机

if (envType.equals("dev")){

if (deployTool.equals("ansible")){

return ["192.168.1.230", "192.168.1.231", "192.168.1.232"]

}

if (deployTool.equals("saltstack")){

return ["node01.zy.com", "node02.zy.com"]

}

}

if (envType.equals("uat")){

if (deployTool.equals("ansible")){

return ["192.168.1.130", "192.168.1.131", "192.168.1.132"]

}

if (deployTool.equals("saltstack")){

return ["node101.zy.com", "node102.zy.com"]

}

}

if (envType.equals("stag")){

if (deployTool.equals("ansible")){

return ["192.168.1.30", "192.168.1.31", "192.168.1.32"]

}

if (deployTool.equals("saltstack")){

return ["node1.zy.com", "node2.zy.com"]

}

}

if (envType.equals("prod")){

if (deployTool.equals("ansible")){

return ["192.168.1.3", "192.168.1.5", "192.168.1.2"]

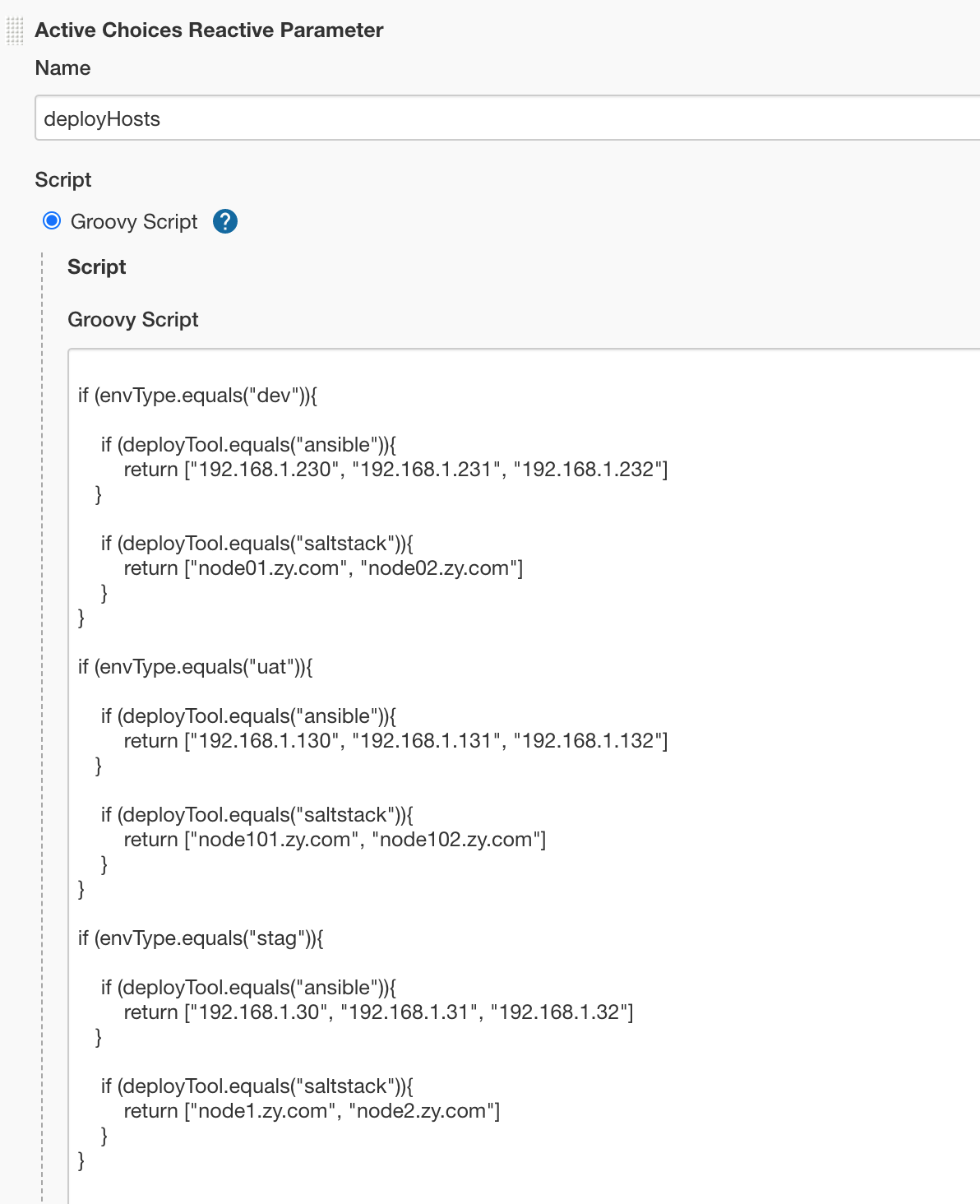
}

if (deployTool.equals("saltstack")){

return ["nodea.zy.com", "nodeb.zy.com"]

}

}



选择关联的参数，多个参数用逗号分割

