**Day5-07月10日-制品库平台实践**

|  |  |  |
| --- | --- | --- |
| **时间** | **标题** | **内容** |
|  | **制品库管理**  **（Nexus/Harbor）** | 1. 开源制品库管理软件-Nexus简介与基本概念 2. 使用Nexus搭建一个Maven私服 3. 使用Harbor搭建一个Docker私服 |
| 1. CI流水线上传制品阶段实现（二进制包） 2. CI流水线上传制品阶段实现（Docker镜像） |
|  | 课件资料 | SublimeText代码： [📎day5.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1625914640849-060d0f46-57c3-4414-97bb-72043e9a6491.zip) |

工作流定义：

* 集成流水线： 提交代码，构建，单测，代码扫描，上传制品【生成制品】
* 制品晋级流水线：输入版本，选择晋级策略。（将制品复制到对应环境的仓库）
* 发布流水线： 输入版本号， 输入部署环境.(从对应环境的制品库中取制品)

制品类型： 二进制制品， docker镜像

核心技能点：

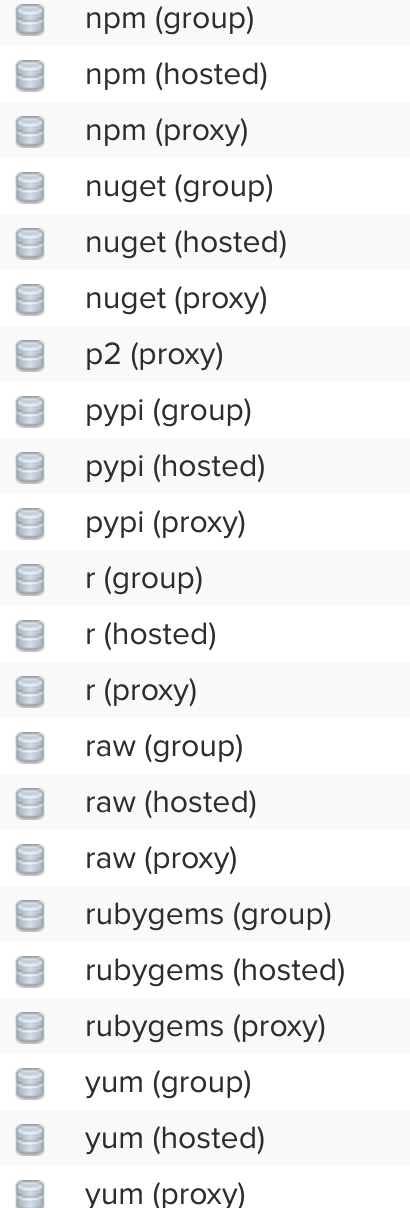
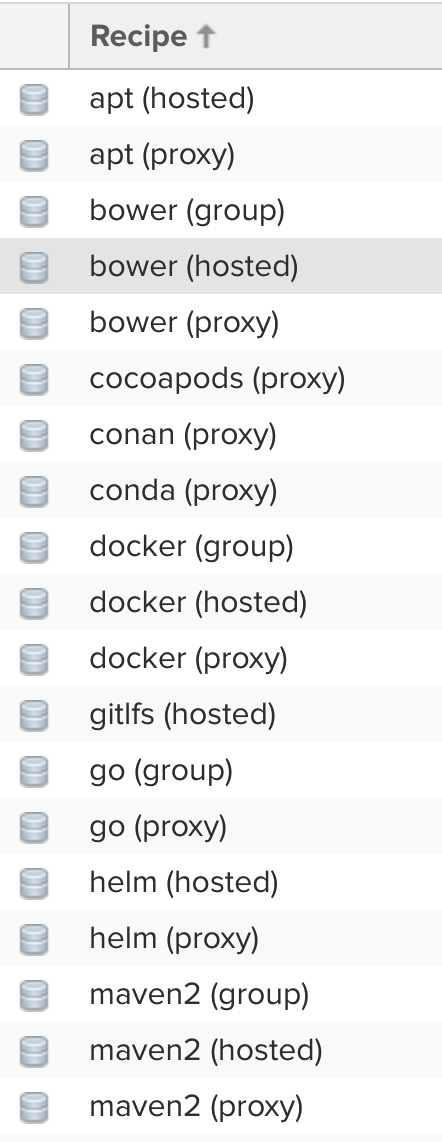
* 制品库管理规范（创建）
* 上传制品
* 晋级制品
* 下载制品

<https://help.sonatype.com/repomanager3/release-notes>

**1.Nexus 制品库实践**

**1.1 Nexus基本功能简介**

从3.x开始，它默认支持许多主流的软件包格式。Docker、Maven、Npm：



**仓库类型：**

* proxy 代理仓库。
* hosted 私有仓库。
* group 仓库组，将多个仓库组合在一起，通过同一个URL对外提供。

代理仓库 ： Maven、Npm等。用于存储外网公共仓库中的插件和依赖，不可进行修改和私自上传。

**1.2 Nexus3安装部署配置**

## 下载镜像

docker pull sonatype/nexus3:3.30.0

## 创建数据存储目录

mkdir -p /data/cicd2/nexus3/data

chmod 777 -R /data/cicd2/nexus3/

## 启动容器

docker run -itd \

--privileged=true --name=nexus3 \

-p 8081:8081 \

-v /data/cicd2/nexus3/data:/nexus-data \

sonatype/nexus3:3.30.0

通过查看日志来确定，Nexus是否已启动并准备就绪，请运行以下命令：

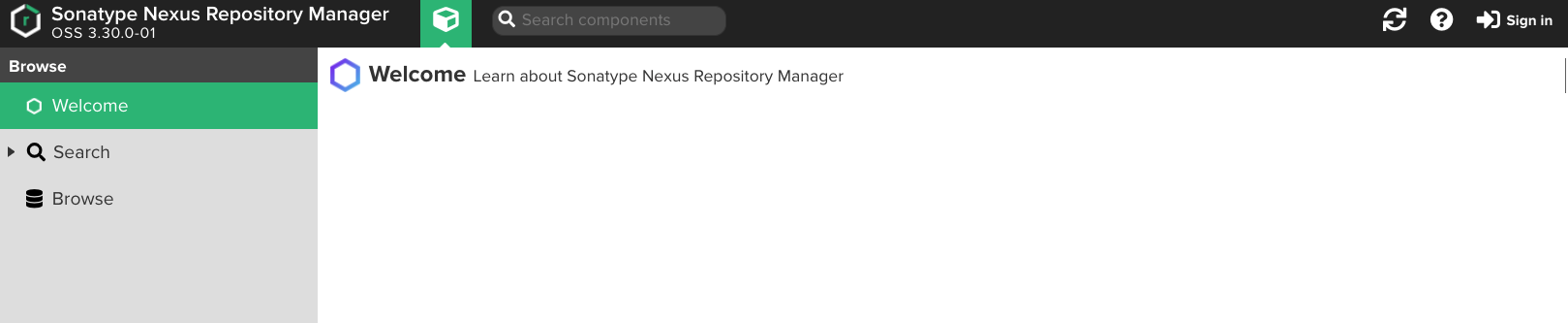
docker logs nexus3 -f

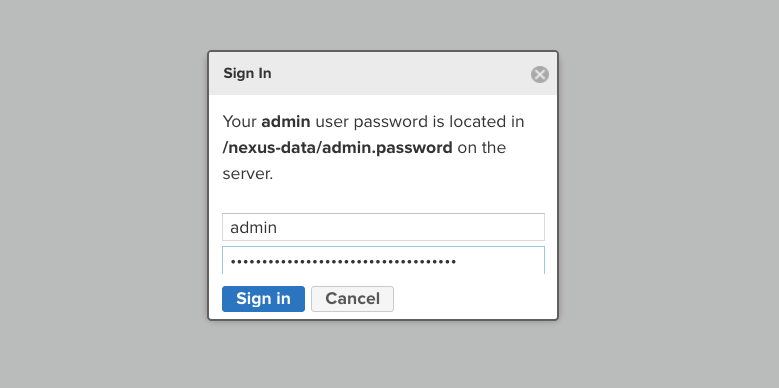
在日志中，看到消息：Started Sonatype Nexus OSS 3.30.0-01这意味着Nexus可以使用了。现在转到浏览器并打开:

http://your-ip-addr:8081

安装完成后, 默认的admin账号密码存储在了数据目录，获取初始化密码:

docker exec -i nexus3 cat /nexus-data/admin.password

登录后需要更新密码 admin/admin123



**1.3 Nexus制品库应用实践**

**搭建Maven私服（代理仓库）**

为什么要搭建maven私服呢？

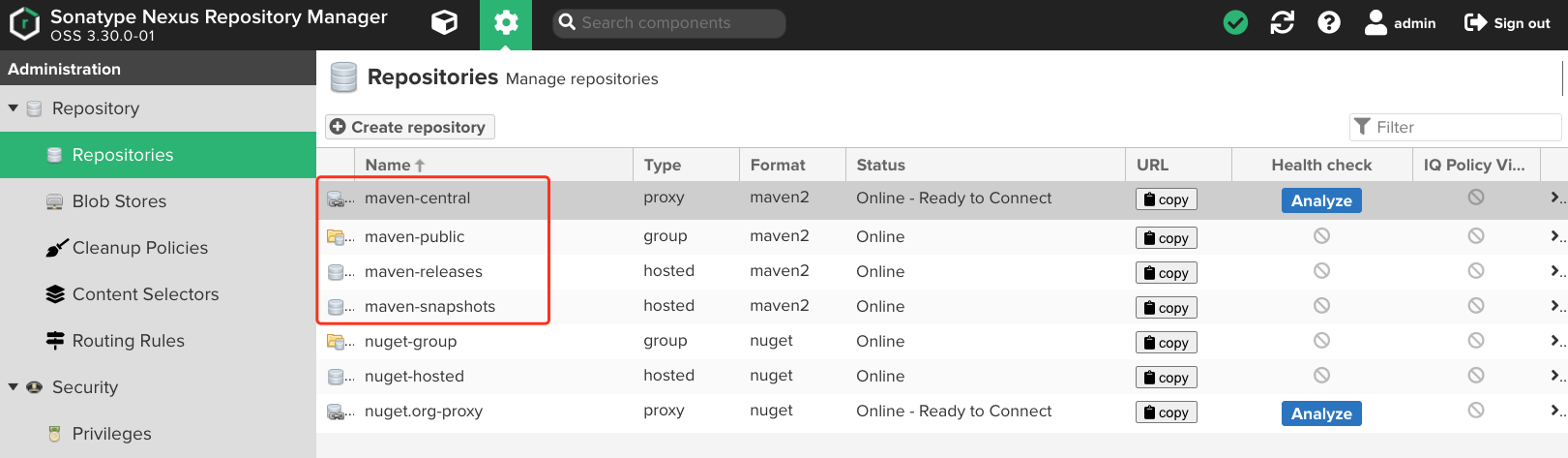
默认开发同学在进行开发的时候会使用一些包管理工具，例如：maven、ant、gradle这些都是常见项目编译构建工具 。这些工具可以理解为是一个命令行工具， 本身不会存储任何依赖包，而是通过公网官方的仓库中下载当前项目构建所需要的包。 （内网的速度要比公网快，这会直接影响管道的构建速度）

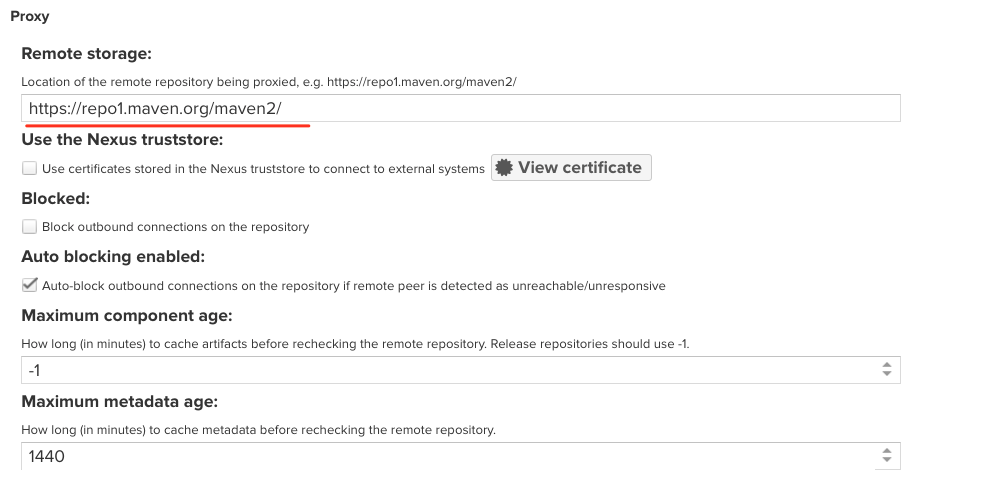
maven私服指的是什么？

使用私服，就是在企业内部建立单一的可信源， 例如：我们在公司通过nexus创建一个代理仓库， 将公网仓库中的maven包代理到内网仓库中。 这样整个公司的同学就可以直接访问内网的私服进行下载构建依赖包。（减少了引入不信任依赖的风险）

代理仓库不会一下子把公网仓库中的所有包下载到本地，而是按需缓存。 例如： 此时我需要使用aa这个包， 如果代理仓库中没有， 则请求外部服务器下载这个包并进行缓存。第二次访问的时候，就直接访问代理仓库了。

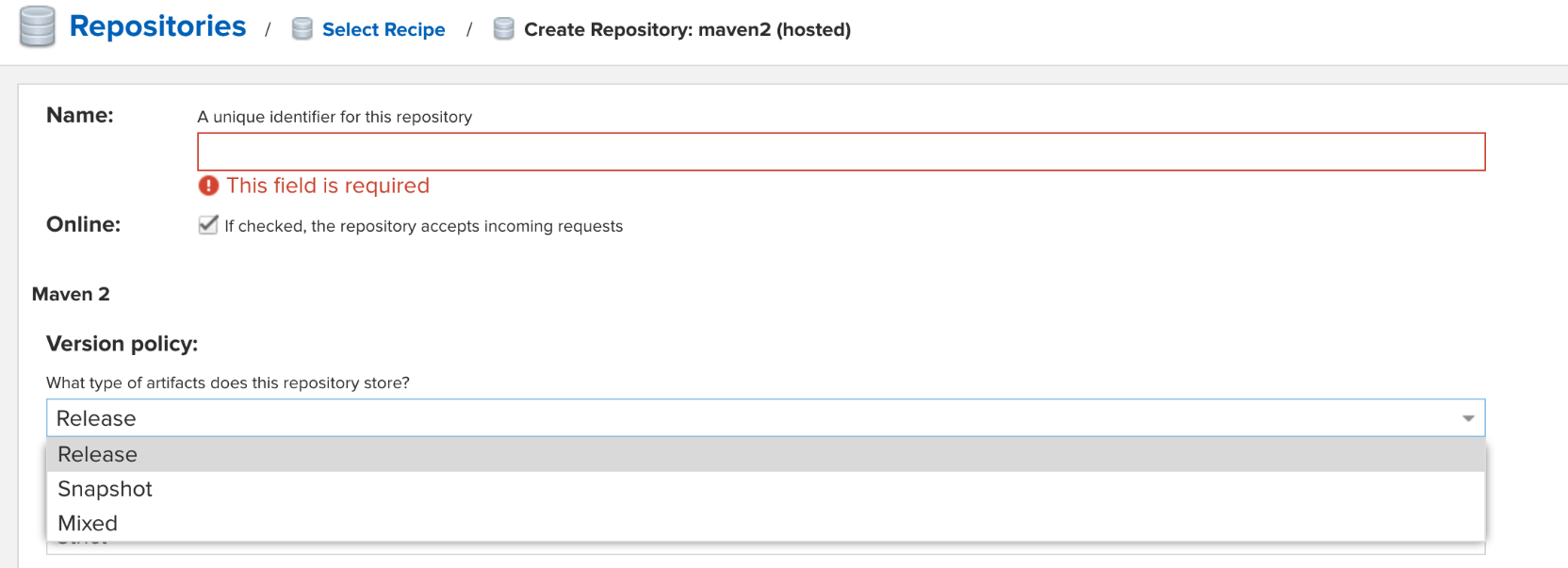
安装nexus后，默认存在以下图中的仓库， 这些仓库是官方默认配置好的maven私服。（可以直接使用）





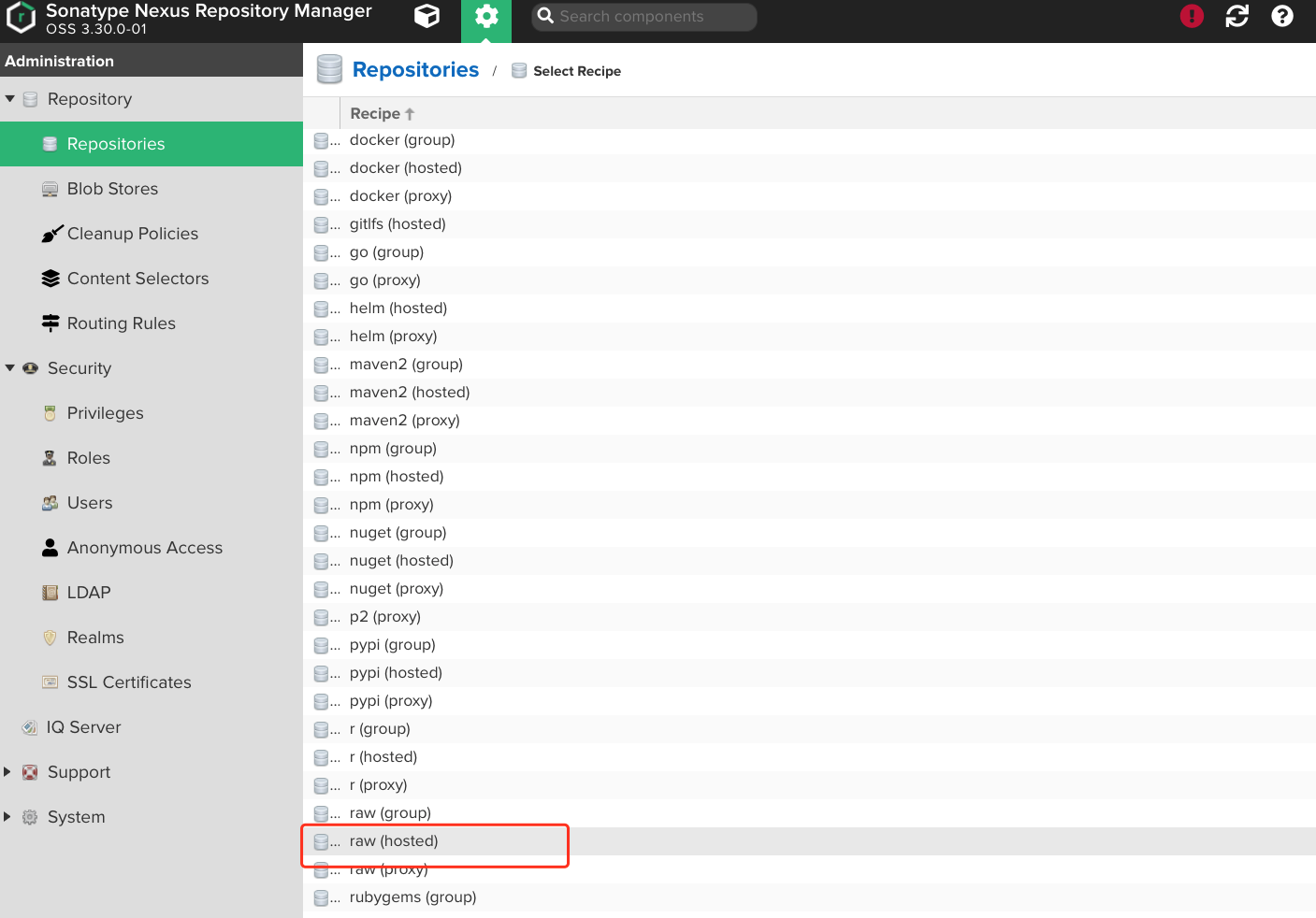
**搭建制品库（本地仓库）**

本地仓库：对于Maven为例， RELEASE类型仓库（存放制品稳定版）SNAPSHOT类型仓库（存放制品开发版）。用于存储公司内部业务开发所生成的制品。

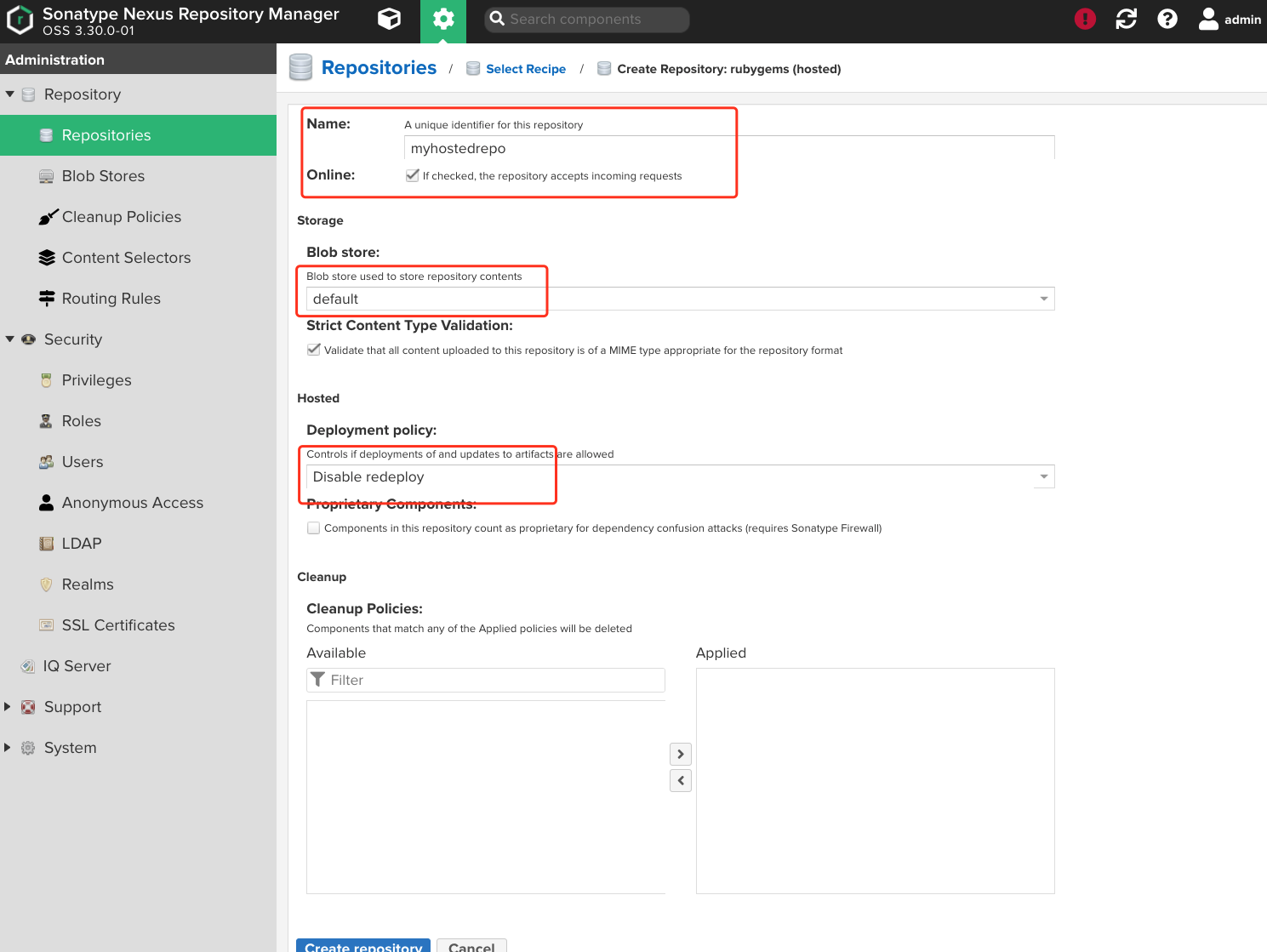


切记：release类型的仓库只能存放release版本的包。

新建raw类型的仓库



设置仓库的信息



**1.4 CI流水线中制品库集成**

在开始引入制品的时候，就应该根据公司的规模和团队特点制定制品库的管理和使用规范。 设置了标准化的规范之后就可以轻松的实现自动化了。

**制品规范定义**

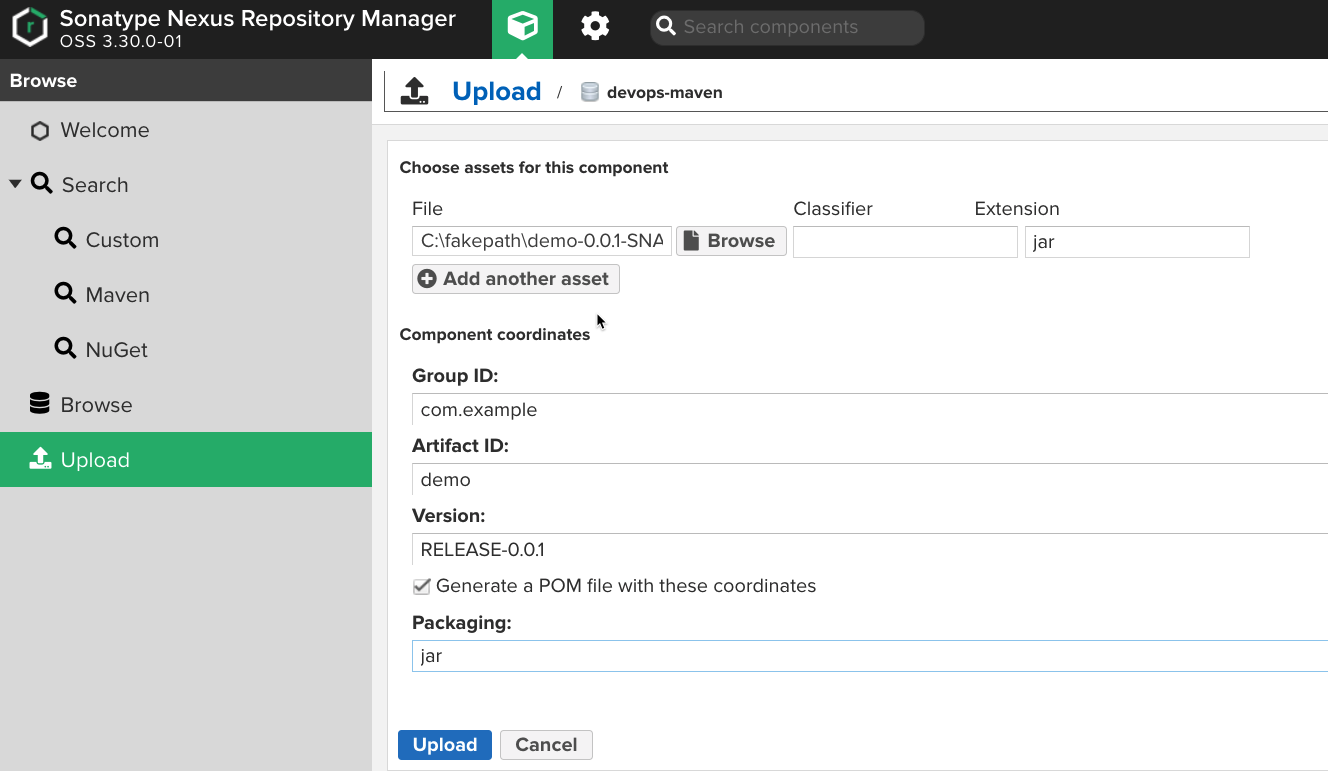
**版本号**

* 主版本号：表示项目的重大架构变更。
* 次版本号：表示较大范围的功能增加和变化。
* 修订版本号：表示重大Bug的修复。
* 里程碑版本：表示某一个版本的里程碑。

**仓库命名**

|  |  |  |
| --- | --- | --- |
| **类型** | **格式** | **示例** |
| 仓库组 | <技术>-group | maven-group |
| 仓库 | <业务简称>-<技术>-<类型> | devops-maven-RELEASE |
| 制品 | <应用名称>-<版本号> | demo-devops-service-1.1.0.jar |
|  |  |  |

**目录结构**： 按照 业务/服务/版本 层级。

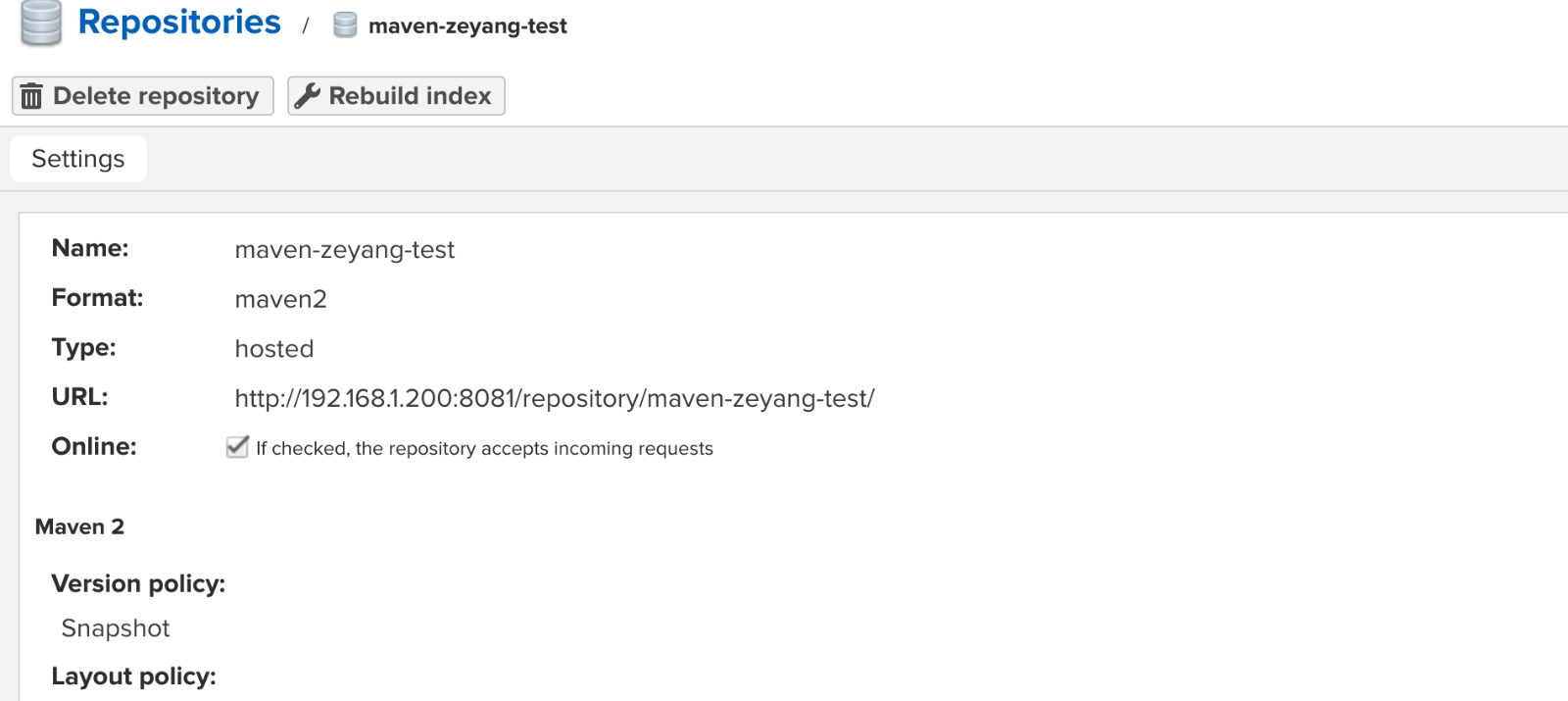
****



**使用maven指令上传制品**

参考：<https://support.sonatype.com/hc/en-us/articles/213465818-How-can-I-programmatically-upload-an-artifact-into-Nexus-2->

上传制品之前， 肯定得保证这仓库是存在的。 如果不存在我们可以新建一个hosted类型的maven仓库。



仓库已经有了， 此时我们需要配置maven的配置文件，在settings.xml中添加仓库的认证信息。如下：

<server>

<id>maven-hosted</id>

<username>admin</username>

<password>admin123</password>

</server>

注意使用mvn deploy 发布时，-DrepositoryId参数的值要与上面配置文件中的<server>中的<id>一致。不然会出现401，用户认证失败的问题。

mvn deploy:deploy-file

-DgroupId=xxxxxx pom中的groupId

-DartifactId=xxxxxx pom中的artifactId

-Dversion=xxxxxx pom中的版本号version

-Dpackaging=xxxxxx pom中打包方式

-Dfile=xxxxxx 本地文件

-Durl=xxxxxx 仓库url

-DrepositoryId=xxxxxx 对应的是setting.xml（认证）

Maven上传报错， 401 可以确定是认证的错误。 需要检查认证信息。

[ERROR] Failed to execute goal org.apache.maven.plugins:maven-deploy-plugin:2.8.2:deploy-file (default-cli) on project demo: Failed to deploy artifacts: Could not transfer artifact com.devops:zeyang:jar:1.1.1 from/to remote-repository (http://192.168.1.200:8081/repository/devops-maven/): authentication failed for http://192.168.1.200:8081/repository/devops-maven/com/devops/zeyang/1.1.1/zeyang-1.1.1.jar, status: 401 Unauthorized -> [Help 1]

替换参数， 执行命令开始上传制品。

mvn deploy:deploy-file \

-DgroupId=com.devops \

-DartifactId=zeyang \

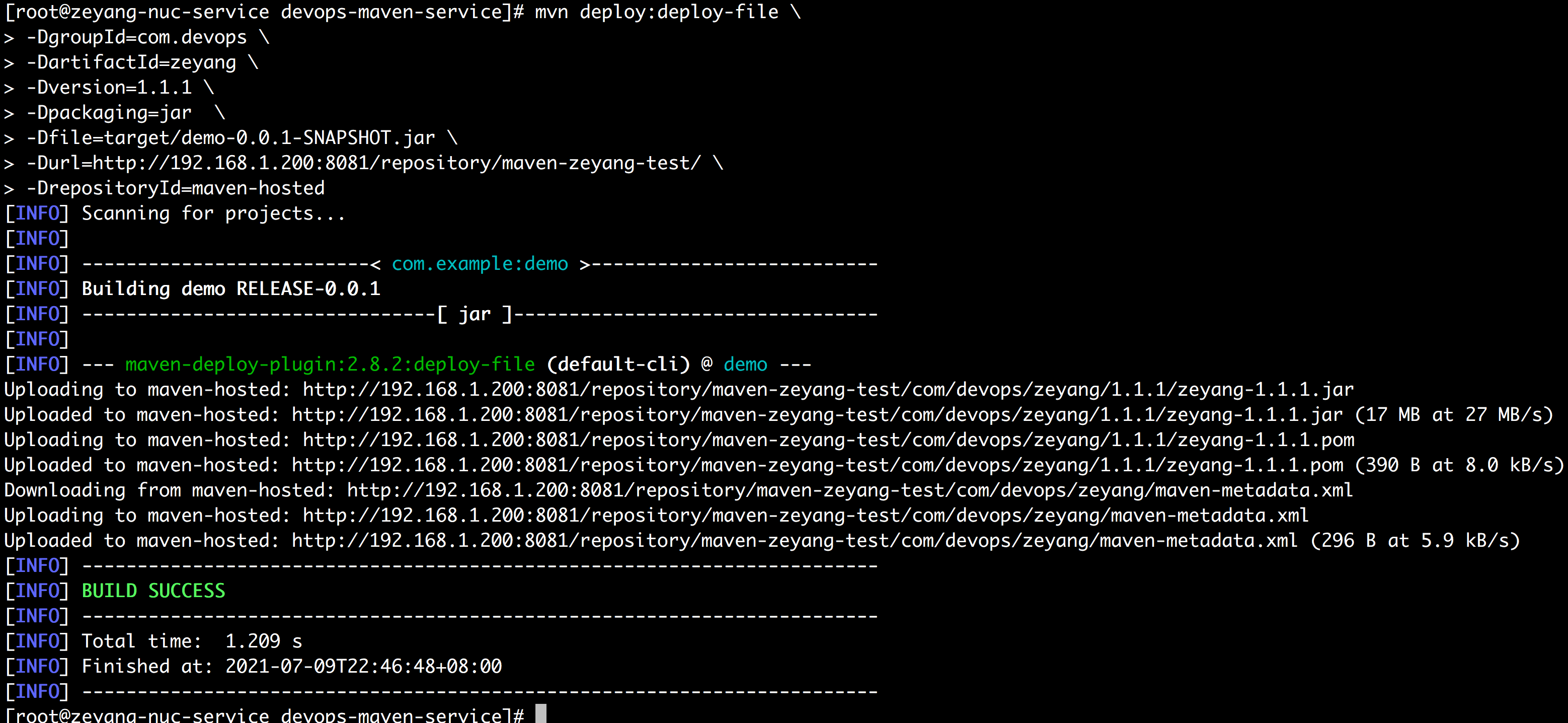
-Dversion=1.1.1 \

-Dpackaging=jar \

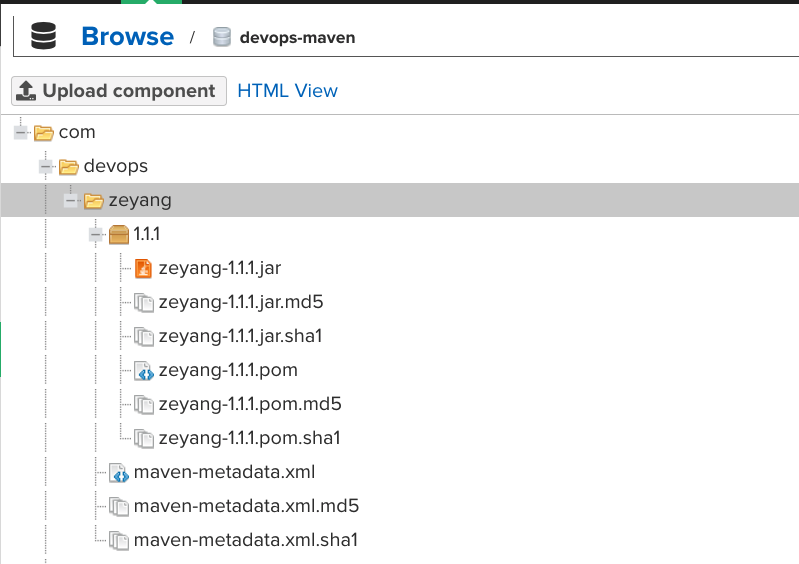
-Dfile=target/demo-0.0.1-SNAPSHOT.jar \

-Durl=http://192.168.1.200:8081/repository/devops-maven/ \

-DrepositoryId=maven-hosted



验证：制品已经上传成功了。



直接读取pom文件(方便)

mvn deploy:deploy-file \

-DgeneratePom=false \

-DrepositoryId=maven-hosted \

-Durl=http://192.168.1.200:8081/repository/devops-maven/ \

-DpomFile=pom.xml \

-Dfile=target/demo-0.0.1-SNAPSHOT.jar

自定义pom信息(灵活)

mvn deploy:deploy-file -Dmaven.test.skip=true \

-Dfile=target/demo-0.0.1-SNAPSHOT.jar \

-DgroupId=com.example \

-DartifactId=demo \

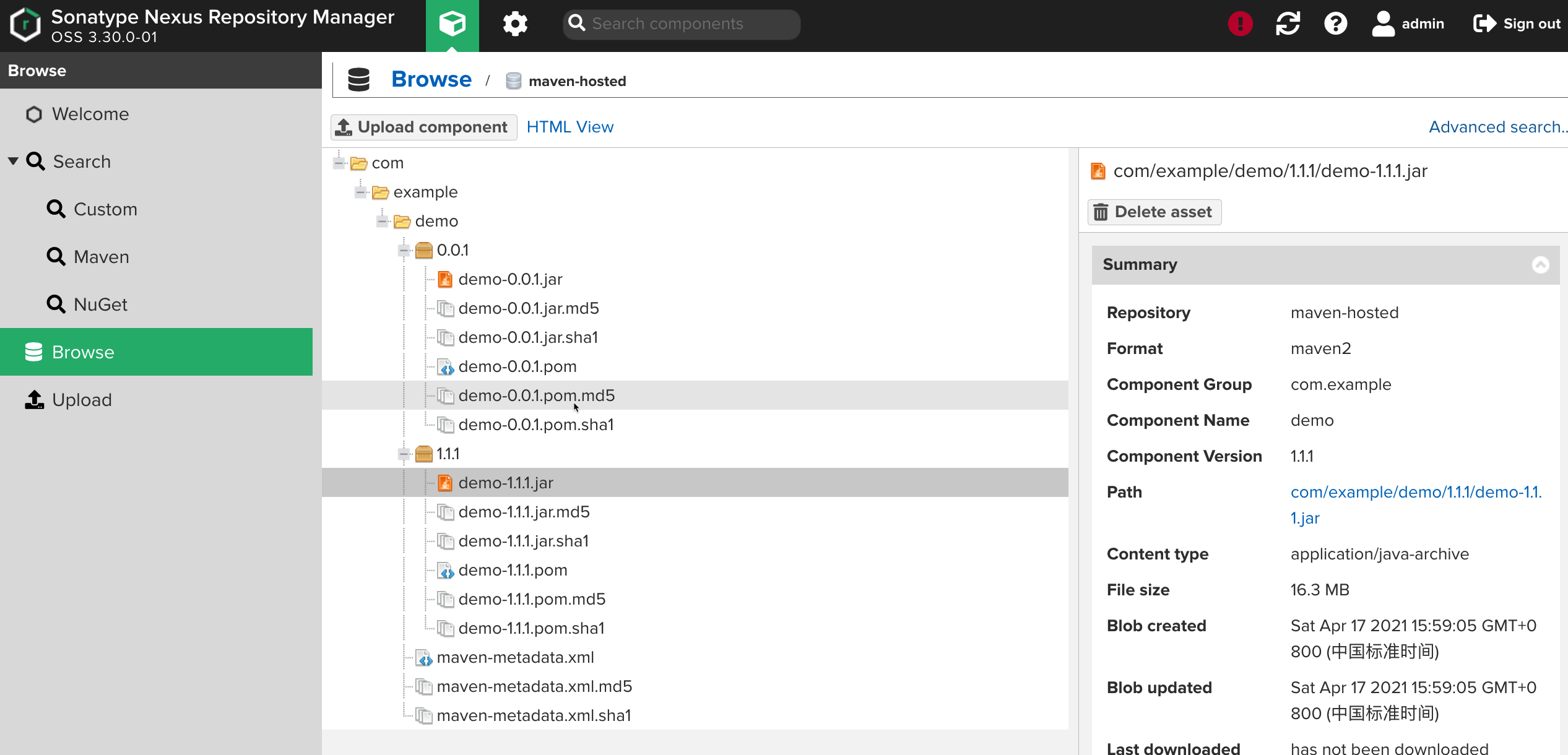
-Dversion=1.1.1 \

-Dpackaging=jar \

-DrepositoryId=maven-hosted \

-Durl=http://192.168.1.200:8081/repository/maven-hosted/

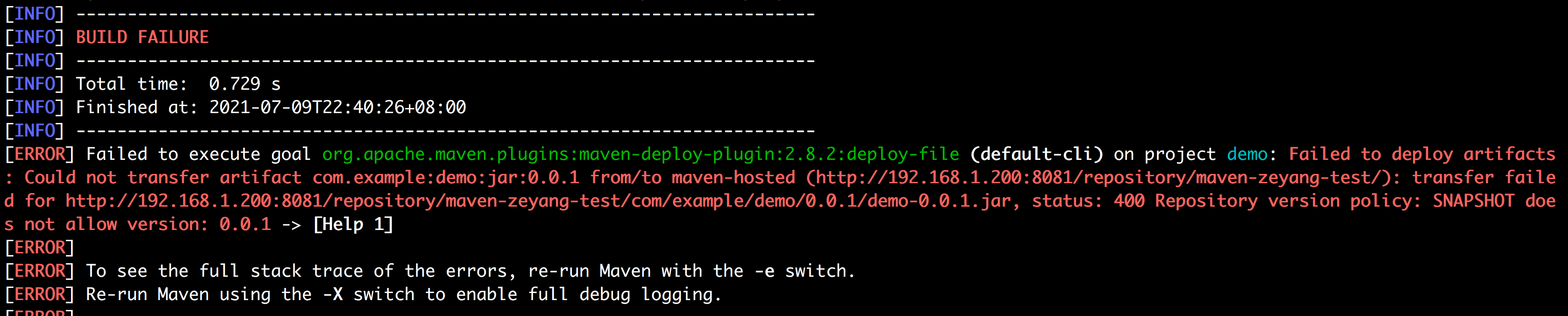




FAQ:

release类型的仓库只能上传release版本的包。如果你尝试用snapshot包上传到release类型的仓库时会遇到这些错误的。

[ERROR] Failed to execute goal org.apache.maven.plugins:maven-deploy-plugin:2.8.2:deploy-file (default-cli) on project demo: Failed to deploy artifacts: Could not transfer artifact com.example:demo:jar:0.0.1 from/to maven-hosted (http://192.168.1.200:8081/repository/maven-zeyang-test/): transfer failed for http://192.168.1.200:8081/repository/maven-zeyang-test/com/example/demo/0.0.1/demo-0.0.1.jar, status: 400 Repository version policy: SNAPSHOT does not allow version: 0.0.1 -> [Help 1]



解决方法： 1. 更新pom中的版本号 2. 对号入座，上传到对应类型的仓库。

<groupId>com.example</groupId>

<artifactId>myapp</artifactId>

<version>0.0.2-SNAPSHOT</version> //改成0.0.2-RELEASE

将包上传到snapshot类型的仓库：

mvn deploy:deploy-file \

-DgeneratePom=false \

-DrepositoryId=maven-hosted \

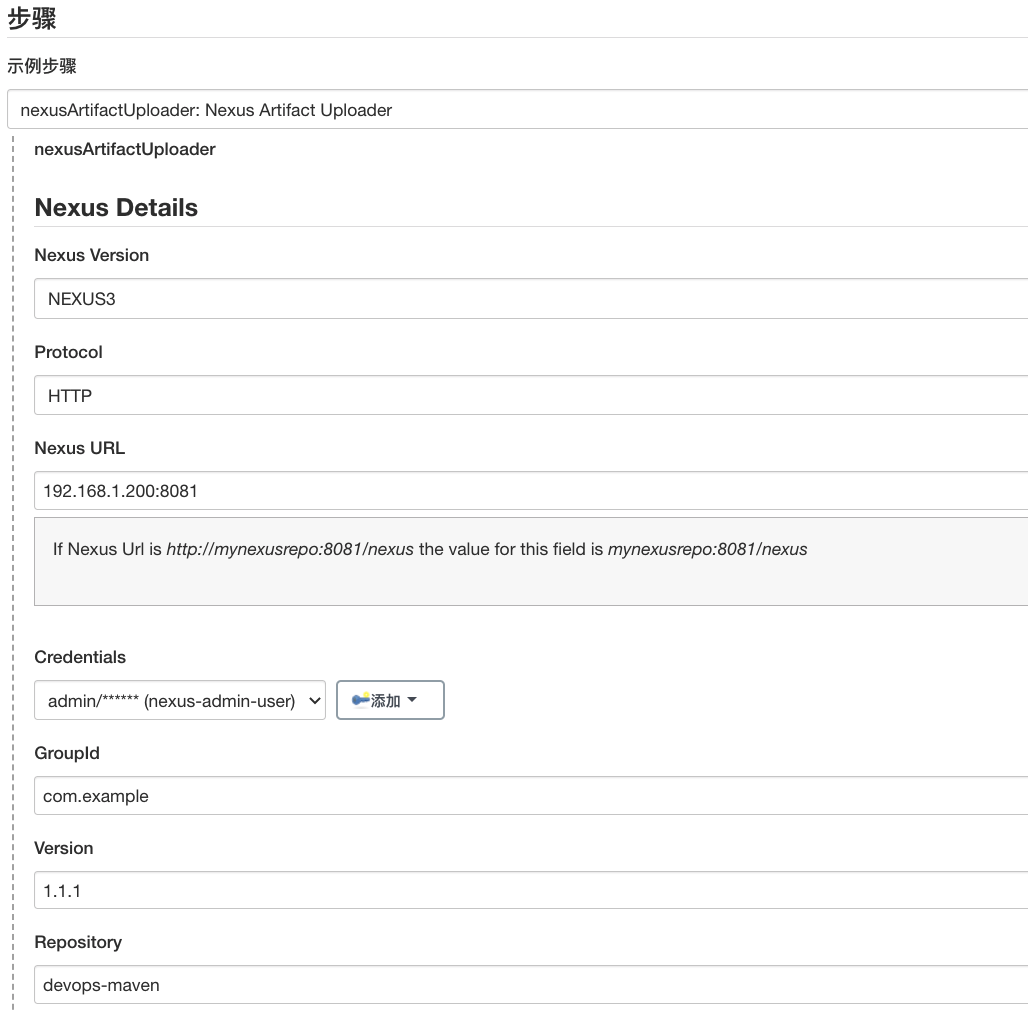
-Durl=http://192.168.1.200:8081/repository/devops-maven-snapshot/ \

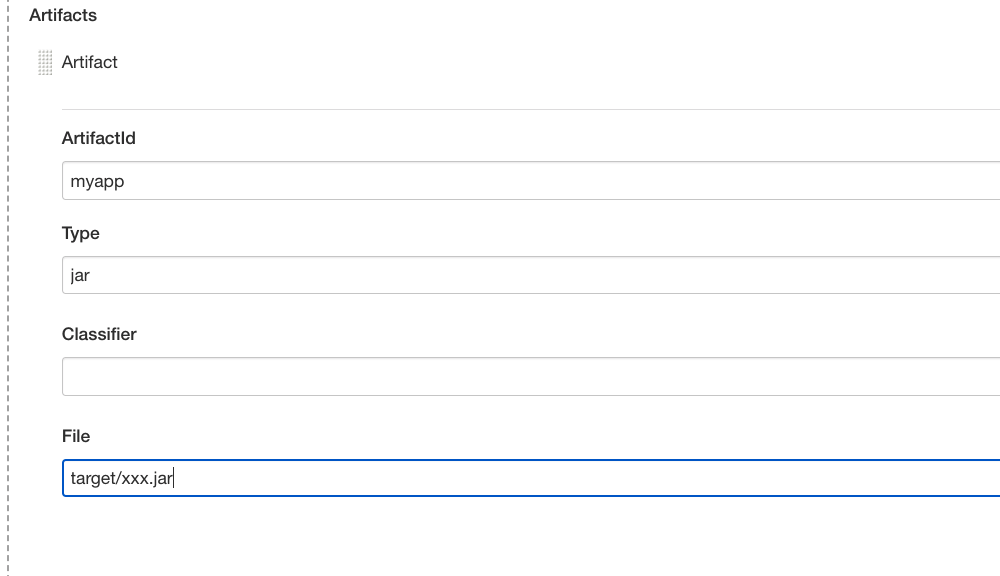
-DpomFile=pom.xml \

-Dfile=target/demo-0.0.1-SNAPSHOT.jar

**使用Jenkins插件上传制品**

安装[Nexus Artifact Uploader](https://plugins.jenkins.io/nexus-artifact-uploader)插件、使用片段生成器生成DSL。





如下面代码：

nexusArtifactUploader artifacts: [[artifactId: 'devops-lib',

classifier: '',

file: 'target/demo-0.0.1-SNAPSHOT.jar',

type: 'jar']],

credentialsId: '111b337e-89a0-4761-9ca3-6ee626ccd7a1',

groupId: 'org.devops',

nexusUrl: '192.168.1.200:8081',

nexusVersion: 'nexus3',

protocol: 'http',

repository: 'devops-maven',

version: '1.1.1'

进行优化， 将参数以变量的方式传递给函数。

//上传制品

def PushArtifacts(repoName,groupId, artifactId, version, type, filePath ){

nexusArtifactUploader artifacts: [[artifactId: "${artifactId}",

classifier: '',

file: "${filePath}",

type: "${type}"]],

credentialsId: '111b337e-89a0-4761-9ca3-6ee626ccd7a1',

groupId: "${groupId}",

nexusUrl: '192.168.1.200:8081',

nexusVersion: 'nexus3',

protocol: 'http',

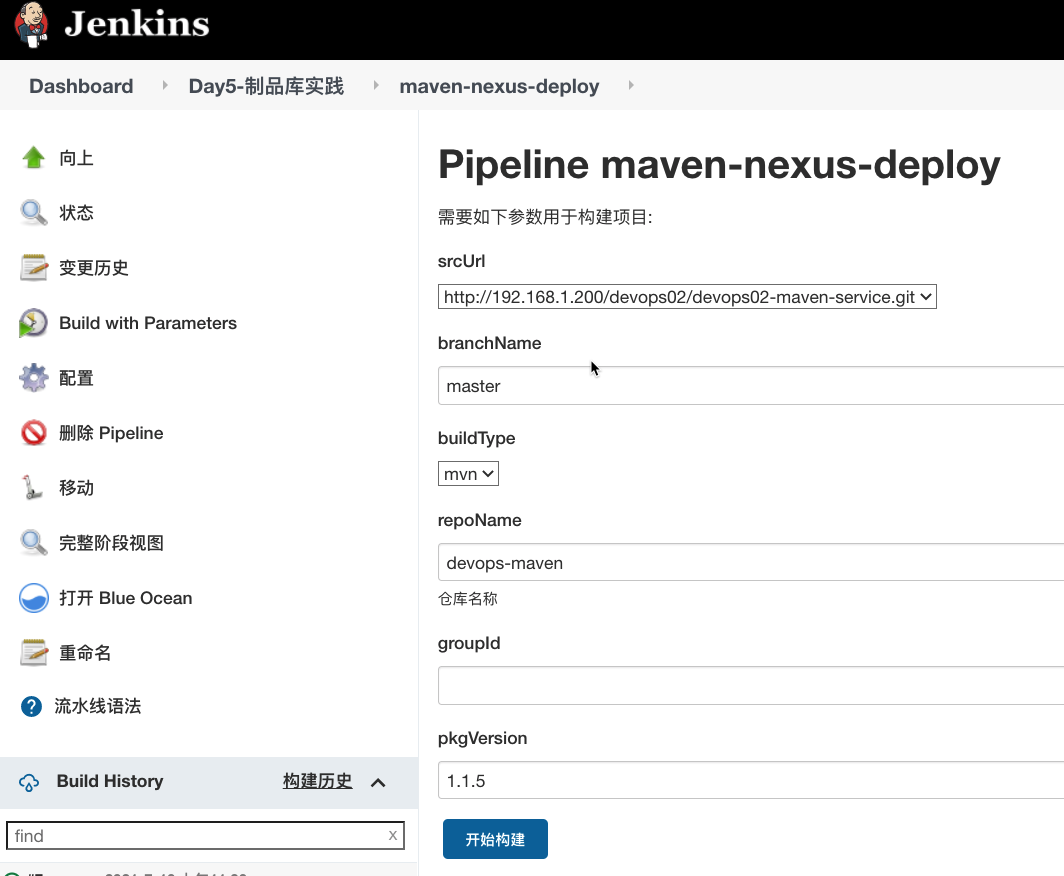
repository: "${repoName}",

version: "${version}"

}

//PushArtifacts('devops-maven','org.devops', 'devops-lib', '1.1.2', 'jar', 'target/demo-0.0.1-SNAPSHOT.jar' )

扩展： 可以在Jenkins页面添加参数， 让用户输入后进行发布。



pkg = sh returnStdout: true, script: """ ls target/ | grep -E "jar\$" """

pkg = pkg.trim()

println(pkg)

// //demo-0.0.1-SNAPSHOT.jar

// pkgName = pkg.split("-")[0]

// pkgType = pkg.split("\\.")[-1]

// println(pkgName + "| " + pkgType)

PushArtifacts( "${params.repoName}",

"${params.groupId}",

pkgName,

"${params.pkgVersion}",

pkgType,

"target/${pkg}"

)

如果是maven类型的具有源码的项目， 可以直接使用mvn命令上传，更加方便。

//上传制品使用maven命令

def PushArtifactsByMvn(repoName,filePath ){

sh """

mvn deploy:deploy-file \

-DgeneratePom=false \

-DrepositoryId="maven-hosted" \

-Durl=http://192.168.1.200:8081/repository/"${repoName}" \

-DpomFile=pom.xml \

-Dfile="${filePath}"

"""

}

PushArtifactsByMvn("${params.repoName}","target/${pkg}")

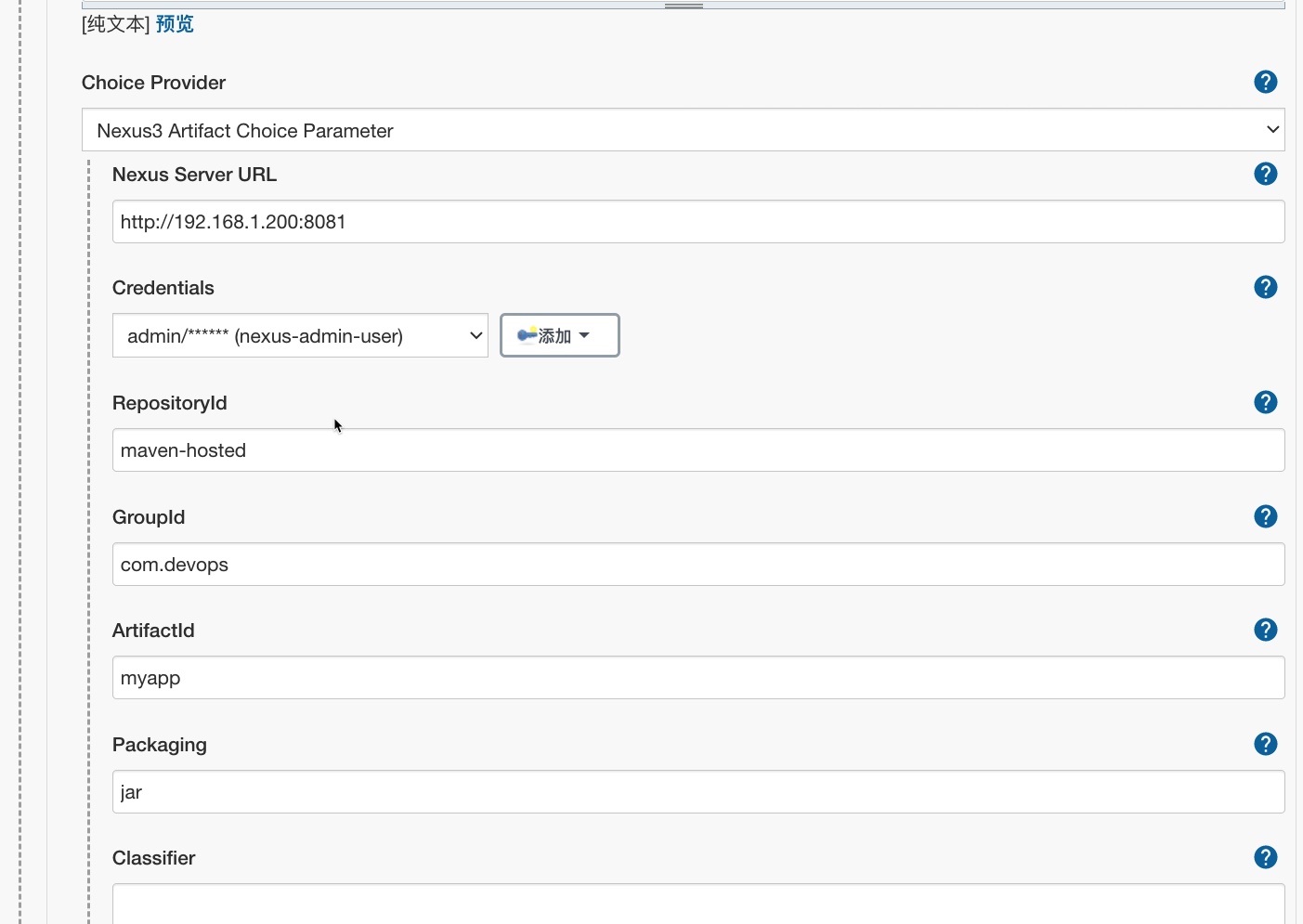
**使用Jenkins插件发布制品**

发布其实就是下载制品，然后将制品发送到目标主机，最后通过脚本或者指令启动程序。

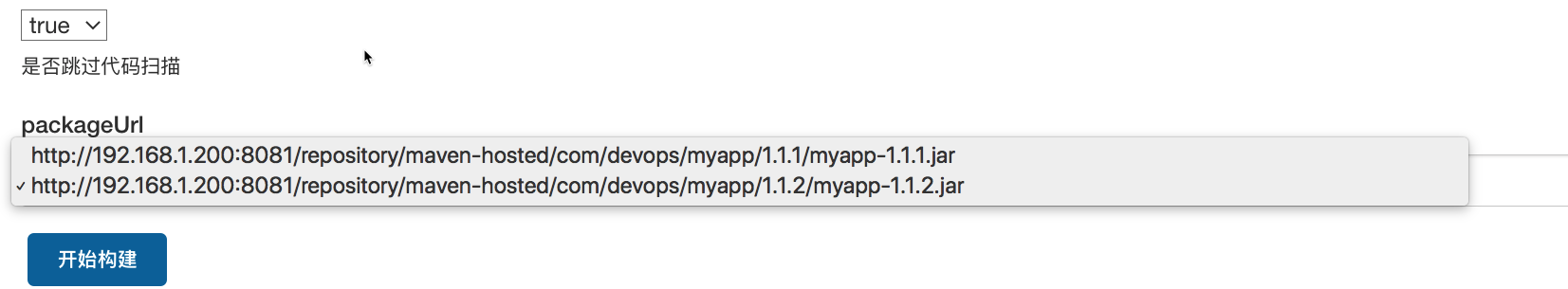
下面是下载制品的示例：

curl http://192.168.1.200:8081/repository/devops-maven/com/example/demo/1.1.10/demo-1.1.10.jar -o app.jar -uadmin:admin123

安装Maven Artifact ChoiceListProvider (Nexus)插件， 可以使用该插件列出包列表。

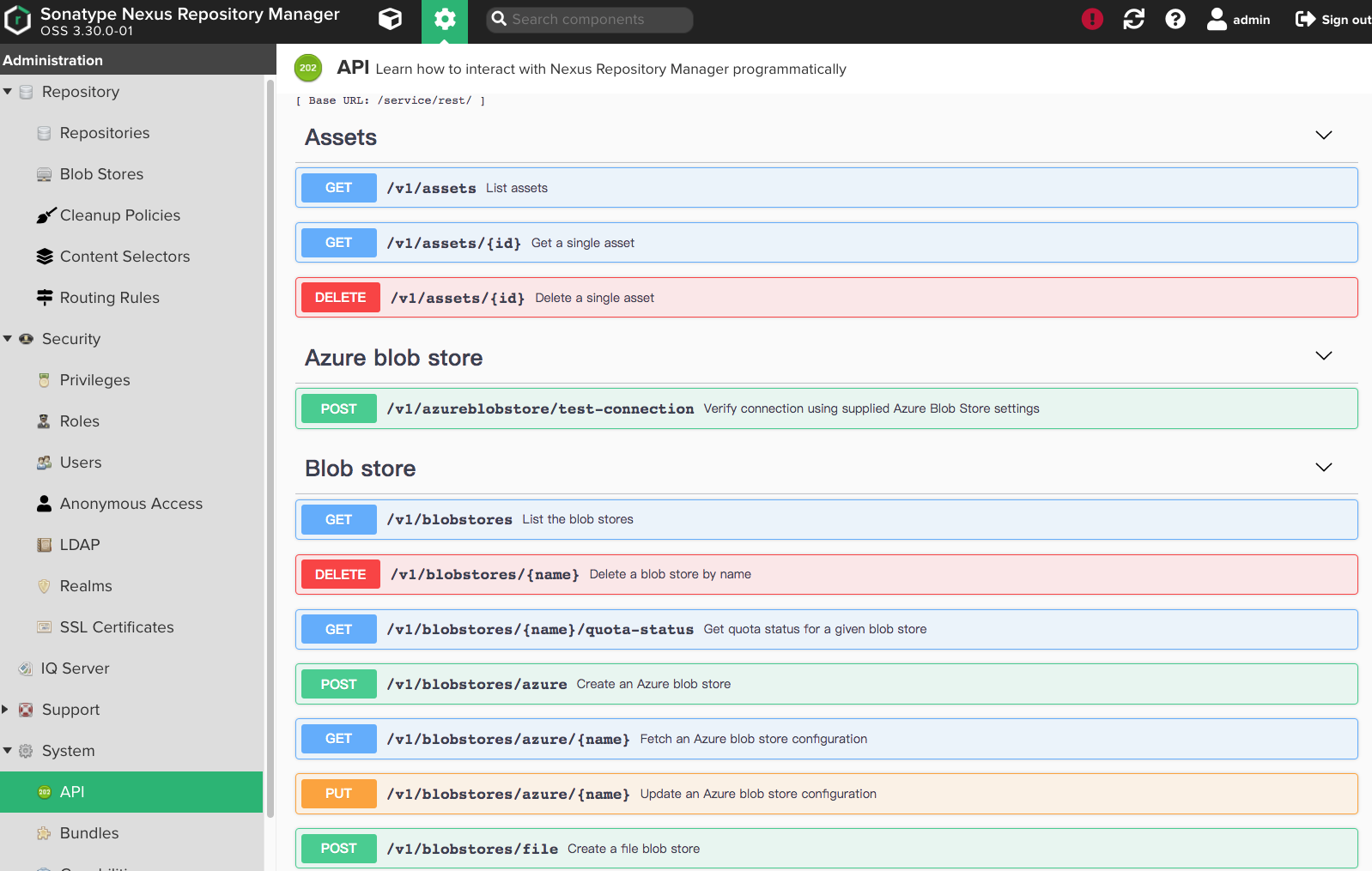


用户选择制品后， 点击构建。此时可以想象，Jenkins下载这个包， 然后通过salt、ansible进行发布部署。



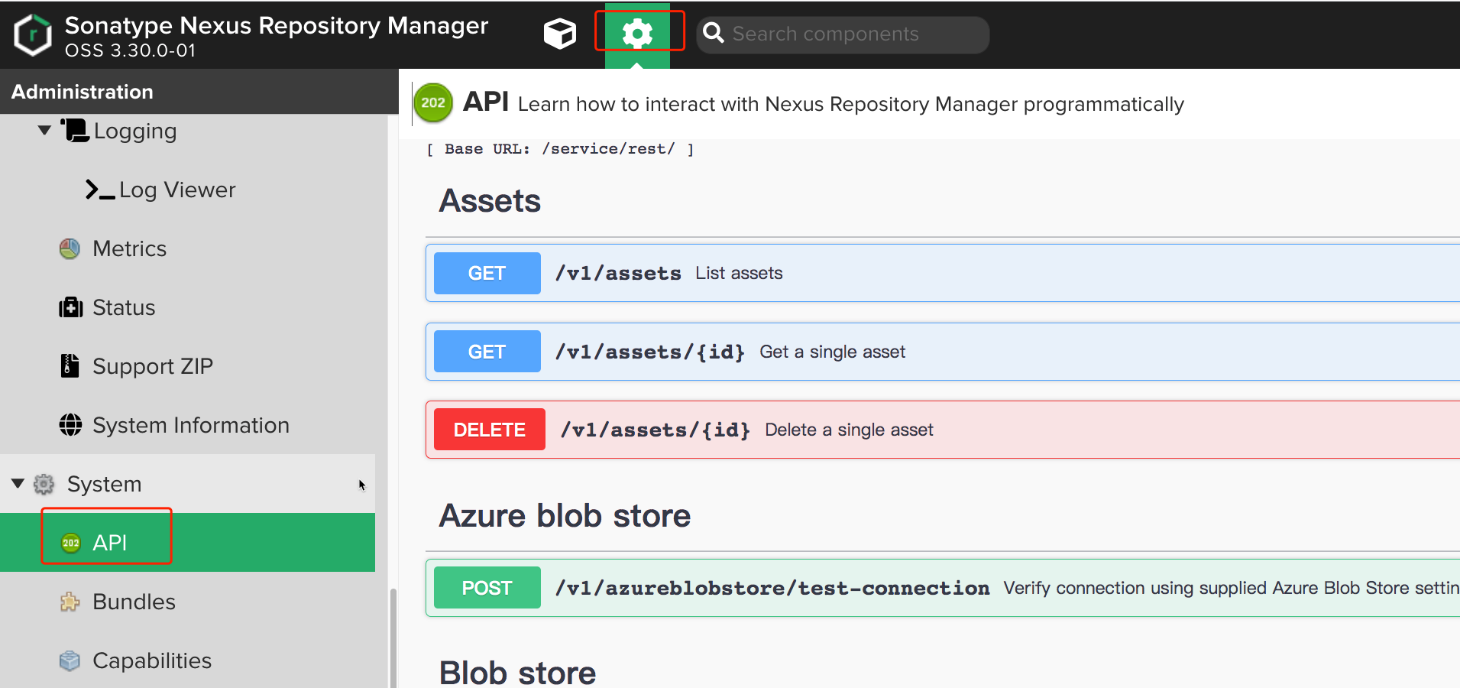
**1.5 Nexus REST API**

<http://192.168.1.200:8081/#admin/system/api>

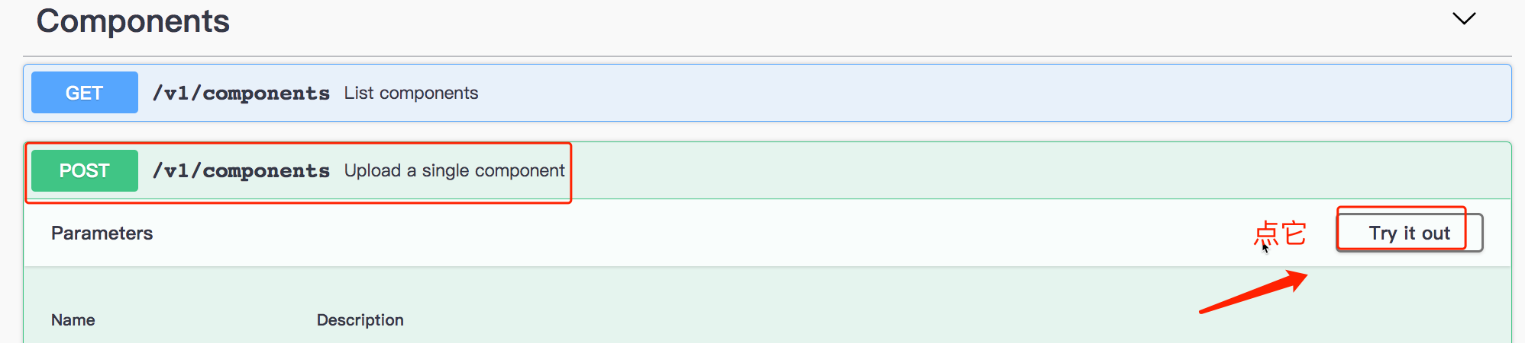


**NexusAPI调试方法**

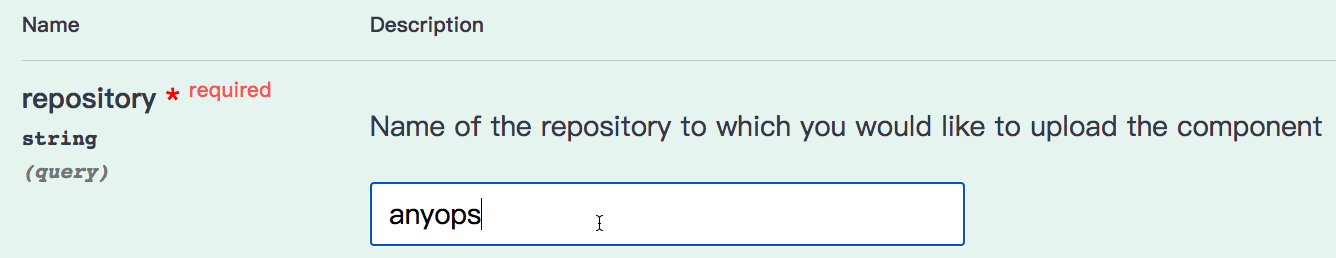
进入设置页面， 找到System > API , 即可进入API调试页面。

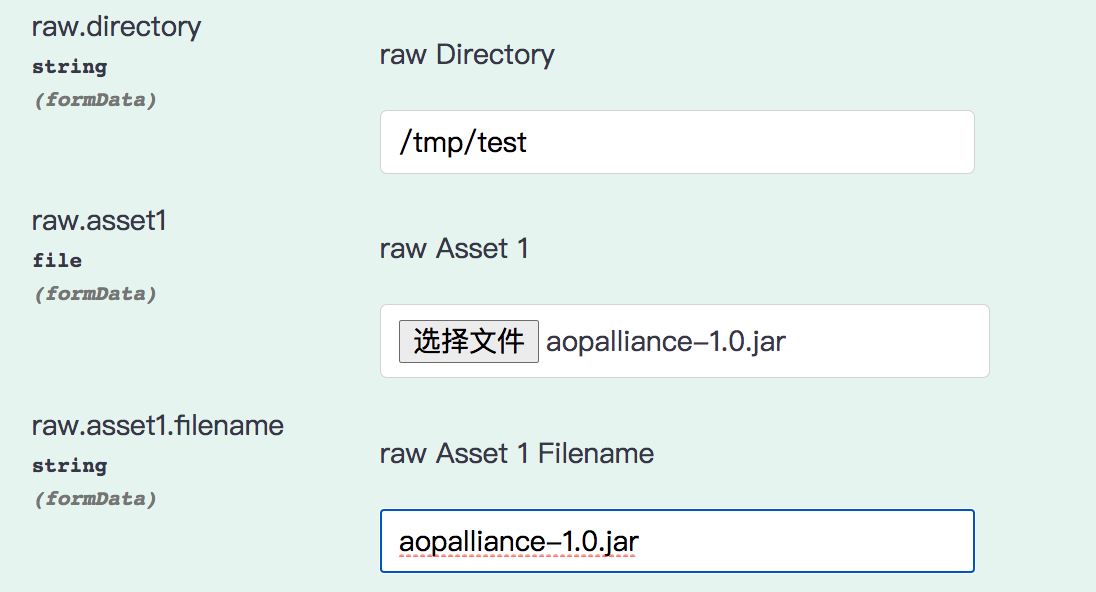


调试API /v1/components, 点击Try it out才能填写信息。

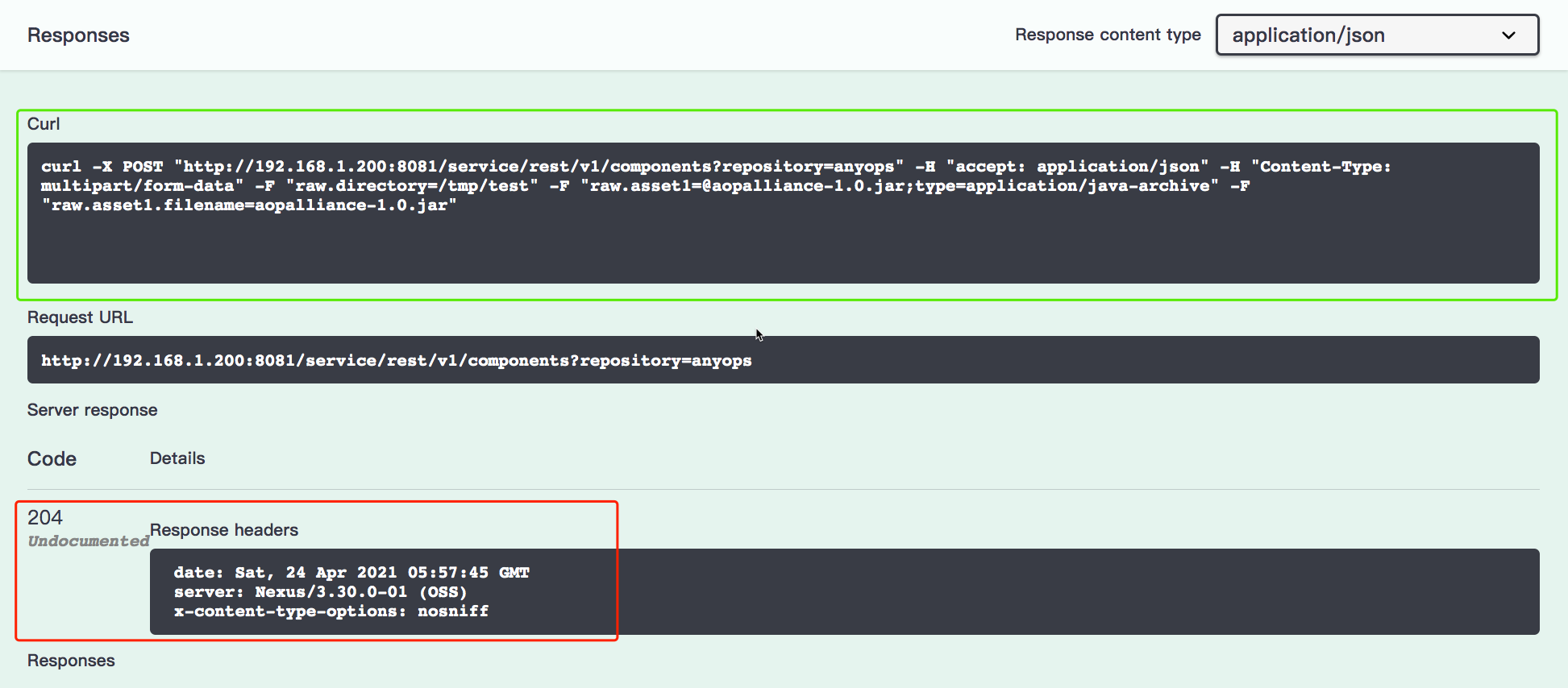


填写参数信息





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"

上传制品（maven类型的制品）：

curl -X POST "http://192.168.1.200:8081/service/rest/v1/components?repository=devops-maven" \

-H "accept: application/json" \

-H "Content-Type: multipart/form-data" \

-F "maven2.groupId=com.newdevops" \

-F "maven2.artifactId=devopsapp" \

-F "maven2.version=1.1.5" \

-F "maven2.packaging=jar" \

-F "maven2.asset1=@demo-0.0.1-SNAPSHOT.jar;type=application/java-archive" \

-F "maven2.asset1.extension=demo-0.0.1-SNAPSHOT.jar" \

-u admin:admin123

**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

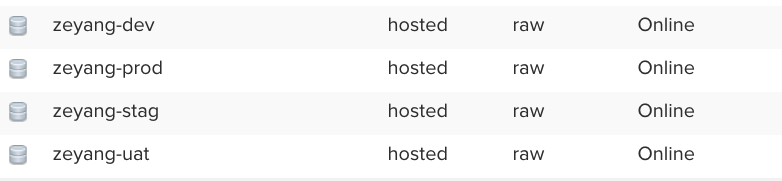
Wget

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

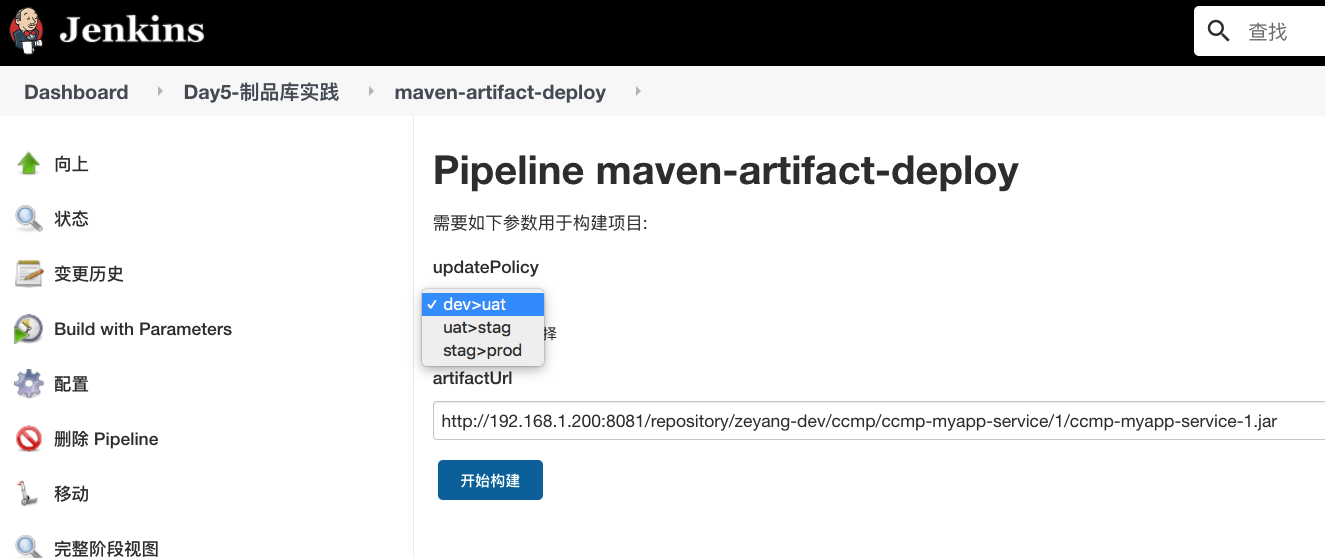
**制品晋级实践【New】**

准备仓库： 按照环境不同，创建四个仓库分别存储不同环境的包。



用户选择：

* 晋级策略
* 包信息



Jenkinsfile内容

currentBuild.description = "trigger by ${params.updatePolicy}"

pipeline {

agent {

label "build"

}

stages{

stage("PullArtifact"){

steps{

script{

env.version = "${params.artifactUrl}".split('/')[-2]

env.appName = "${params.artifactUrl}".split('/')[-3] // Day5/acmp-myapp-service

env.buName = "${params.artifactUrl}".split('/')[-4]

env.pkgName = env.appName + "-${version}.jar"

sh """

curl ${params.artifactUrl} -o ${env.pkgName} -u admin:admin123

"""

}

}

}

stage("UpdateArtifact"){

steps{

script{

switch("${params.updatePolicy}") {

case "dev>uat":

repoName = "zeyang-uat"

targetDir = "${env.buName}/${env.appName}/${env.version}/"

PushRawArtifacts(repoName,targetDir, "./", "${env.pkgName}" )

break;

case "uat>stag":

repoName = "zeyang-stag"

targetDir = "${env.buName}/${env.appName}/${env.version}/"

PushRawArtifacts(repoName,targetDir, "./", "${env.pkgName}" )

break;

case "stag>prod":

repoName = "zeyang-prod"

targetDir = "${env.buName}/${env.appName}/${env.version}/"

PushRawArtifacts(repoName,targetDir, "./", "${env.pkgName}" )

break;

default:

println("xxxxxxxxx")

}

}

}

}

}

}

//上传制品

def PushRawArtifacts(repoName,targetDir, filePath, pkgName ){

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

sh """

curl -X POST "http://192.168.1.200:8081/service/rest/v1/components?repository=${repoName}" \

-H "accept: application/json" \

-H "Content-Type: multipart/form-data" \

-F "raw.directory=${targetDir}" \

-F "raw.asset1=@${filePath}/${pkgName};type=application/java-archive" \

-F "raw.asset1.filename=${pkgName}" \

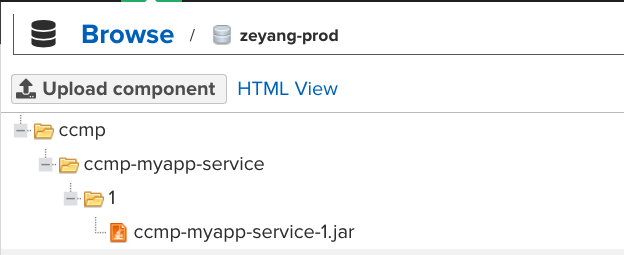
-u "${USER}":"${TOKEN}"

"""

}

}

效果：



**2. Harbor镜像库实践**

[官网： https://goharbor.io/](https://goharbor.io/)

**2.1 Harbor基本功能简介**

Harbor是一个用于存储和分发Docker镜像的企业级Registry服务器，可以用来构建企业内部的Docker镜像仓库。具有一些企业需要的功能特性，如镜像同步复制、漏洞扫描和权限管理等。

**2.2 Harbor安装部署配置**

安装docker-compose

参考文档：<https://docs.docker.com/compose/install/#install-compose-on-linux-systems>

curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

cp docker-compose-Linux-x86\_64 /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

## 测试命令

docker-compose

Define and run multi-container applications with Docker.

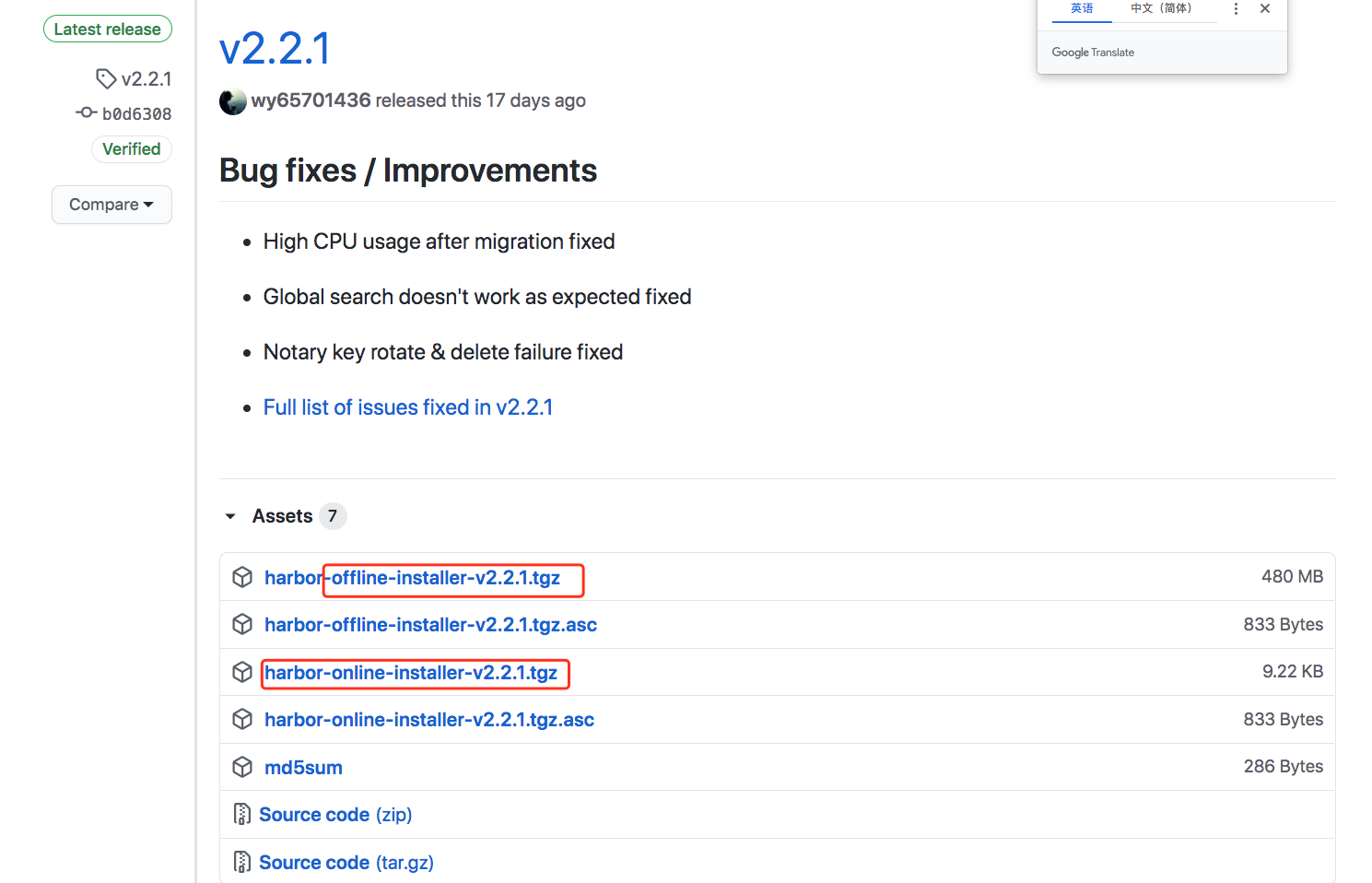
Usage:

docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]

docker-compose -h|--help

下载release包： <https://goharbor.io/docs/2.2.0/install-config/download-installer/>

<https://github.com/goharbor/harbor/releases>



## 解压

tar zxf harbor-online-installer-v1.10.4.tgz

cd harbor

## 编辑配置文件

vim harbor.yml

----

### 修改主机名，注释https

hostname: 192.168.1.200

# http related config

http:

# port for http, default is 80. If https enabled, this port will redirect to https port

port: 80

# https related config

#https:

# https port for harbor, default is 443

# port: 443

# The path of cert and key files for nginx

#certificate: /your/certificate/path

#private\_key: /your/private/key/path

----

## 安装

sh install.sh

[Step 4]: starting Harbor ...

Creating network "harbor\_harbor" with the default driver

Creating harbor-log ... done

Creating registry ... done

Creating harbor-portal ... done

Creating registryctl ... done

Creating redis ... done

Creating harbor-db ... done

Creating harbor-core ... done

Creating nginx ... done

Creating harbor-jobservice ... done

✔ ----Harbor has been installed and started successfully.----

**2.3 创建镜像仓库**



**2.4 CI流水线中制品库集成**

**镜像规范**

业务/服务:commitId

**镜像上传**

docker login 192.168.1.200:8088

Username: admin

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

# docker tag nginx:1.9.1 192.168.1.200:8088/devops/devops-nginx-service:1.9.1

# docker tag nginx:1.9.1 192.168.1.200:8088/devops/devops-nginx-service:1.9.2

# docker push 192.168.1.200:8088/devops/devops-nginx-service:1.9.2

The push refers to repository [192.168.1.200:8088/devops/devops-nginx-service]

5f70bf18a086: Layer already exists

23923d6d8fc4: Layer already exists

2471231f3381: Layer already exists

21d3d9e179b7: Layer already exists

7dc0e946afc6: Layer already exists

a622086a695f: Layer already exists

d55f823e63e3: Layer already exists

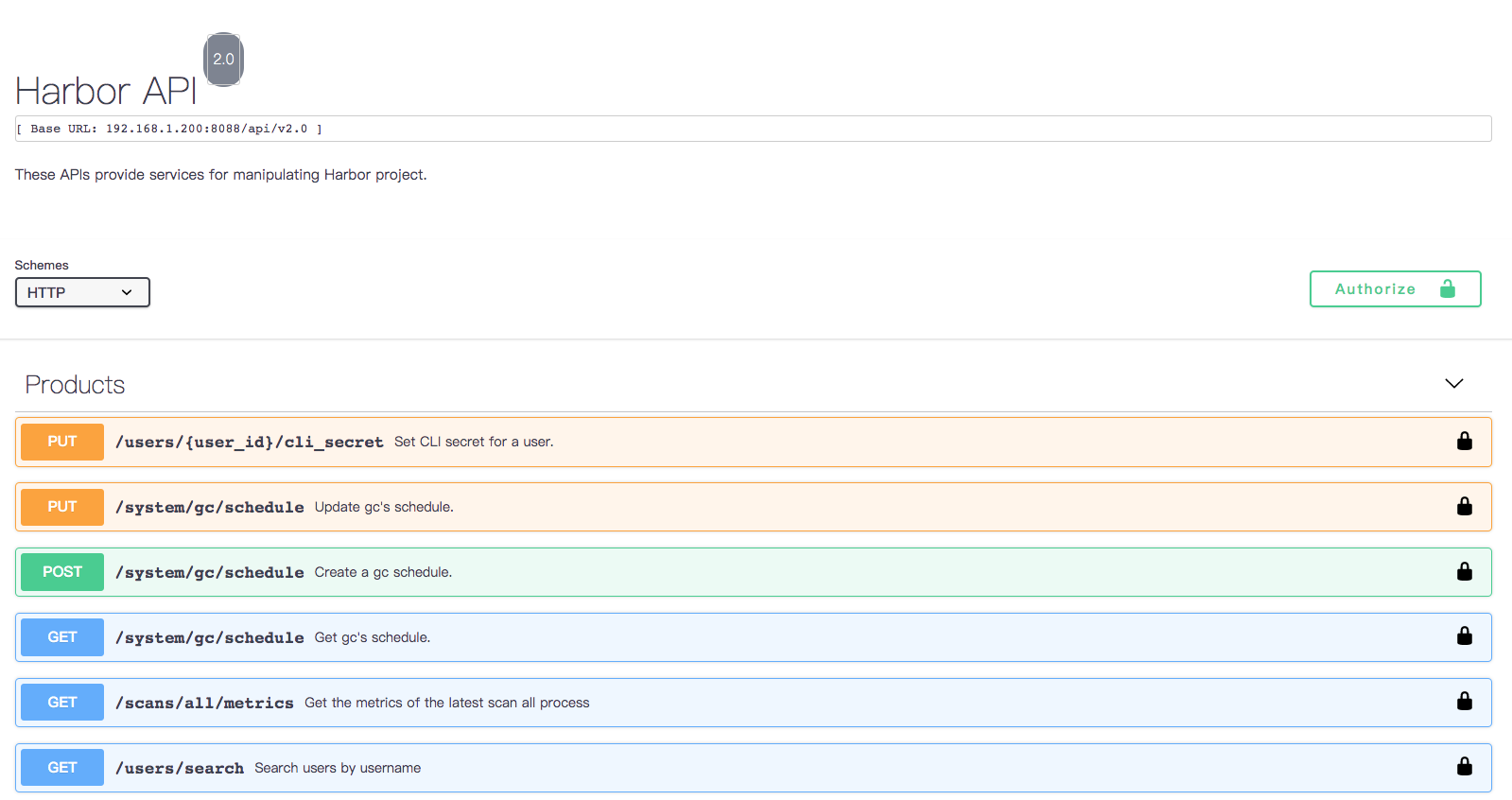
1.9.2: digest: sha256:a42a428525996f3a84d466ee628a074cac568e0e8c99b5d6f7398be342337039 size: 2805

**镜像下载**

docker pull 192.168.1.200:8088/devops/devops-nginx-service:1.9.2

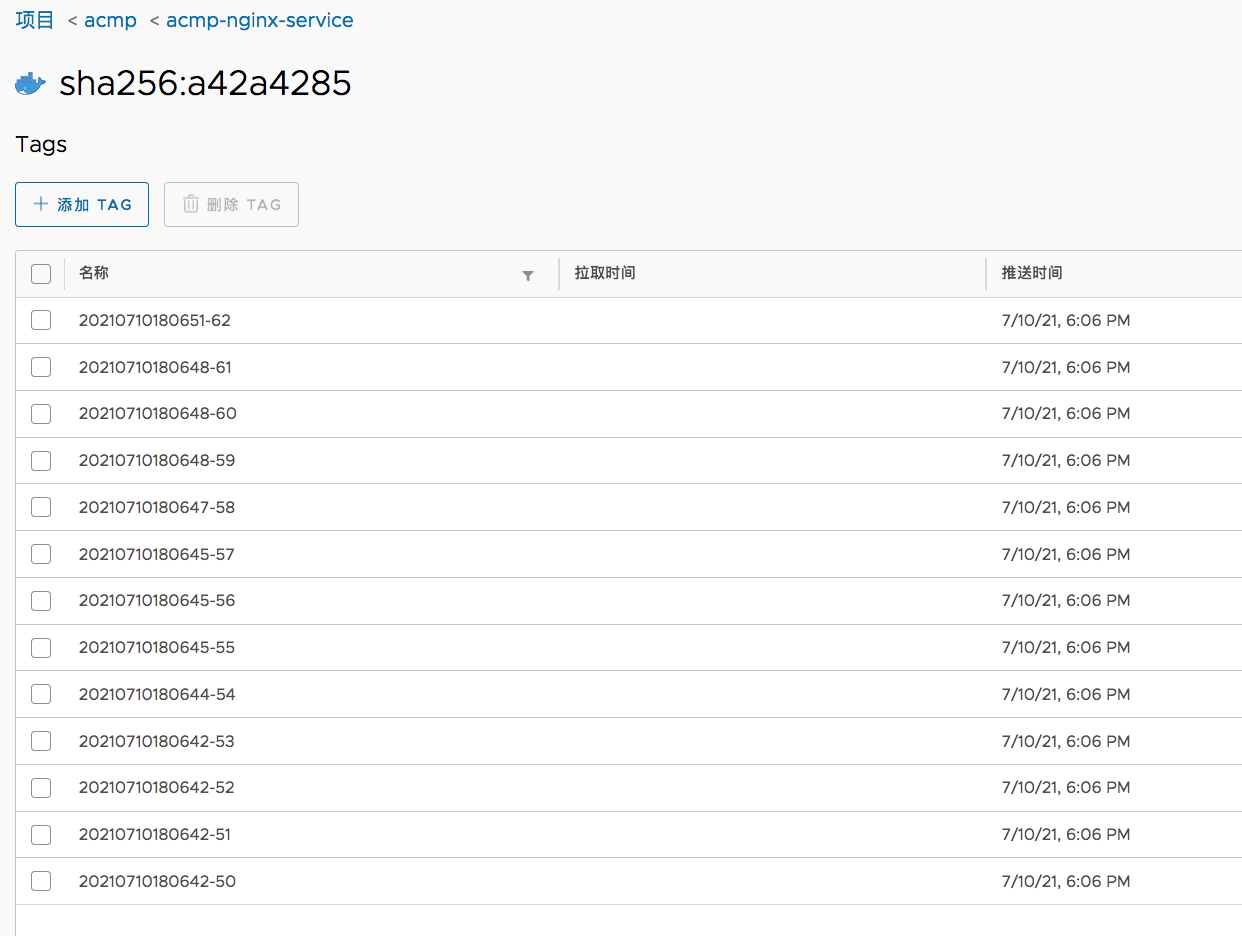
**2.5 Harbor REST API**

<http://192.168.1.200:8088/devcenter-api-2.0>

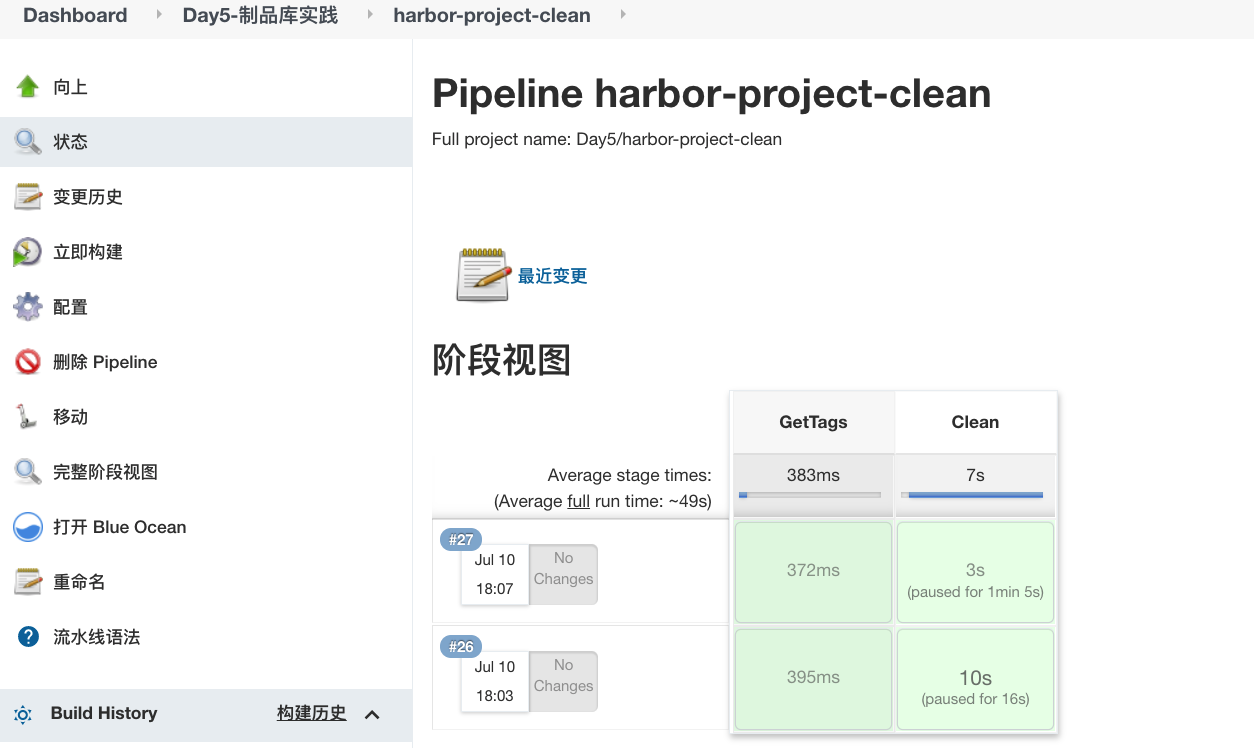


**2.6 Harbor API 实践【New】**

在Harbor中批量的创建一些标签



创建一个Jenkins作业, 不需要配置任何参数。 只需要把Jenkinfile内容放进去。



Jenkinsfile内容：

* 通过API 获取所有的tag标签列表。
* 将标签信息用INPUT交互式的展示。
* 调用删除接口，删除镜像标签。



import groovy.json.JsonSlurper

/\*

清理docker镜像

1. 获取镜像列表

2. 用户选择删除

3. 调用api删除

\*/

pipeline {

agent {

label "build"

}

stages{

stage("GetTags"){

steps{

script{

env.projectName = "acmp"

env.repoName = "acmp-nginx-service"

env.result = GetArtifactTag(env.projectName, env.repoName)

env.result = env.result - '[' - ']'

}

}

}

stage("Clean"){

steps{

script{

def result = input message: "是否删除${env.projectName}项目的${env.repoName}这些标签：",

parameters: [extendedChoice(defaultValue: "${env.result}",

multiSelectDelimiter: ',',

name: 'taga',

quoteValue: false,

saveJSONParameterToFile: false,

type: 'PT\_CHECKBOX',

value: "${env.result}",

visibleItemCount: 20)]

println("${result}")

// println("Delete ${taga}, doing.......")

// tags = "${taga}" - '[' - ']'

for(t in result.split(',')){

println("Delete >>>>" + t.trim())

DeleteArtifactTag(env.projectName,env.repoName, t.trim())

}

}

}

}

}

}

// 删除镜像tag

def DeleteArtifactTag(projectName,repoName, tagName){

harborAPI = "http://192.168.1.200:8088/api/v2.0/projects/${projectName}/repositories/${repoName}"

apiURL = "artifacts/${tagName}/tags/${tagName}"

sh """ curl -X DELETE "${harborAPI}/${apiURL}" -H "accept: application/json" -u admin:Harbor12345 """

}

// 获取镜像的所有标签

// acmp-nginx-service

def GetArtifactTag(projectName,repoName ){

harborAPI = "http://192.168.1.200:8088/api/v2.0/projects/${projectName}/repositories/${repoName}"

apiURL = "artifacts?page=1&page\_size=10"

response = sh returnStdout: true, script: """curl -X GET "${harborAPI}/${apiURL}" -H "accept: application/json" -u admin:Harbor12345 """

response = readJSON text: """${response - "\n"}"""

tags = []

for (t in response[0].tags){

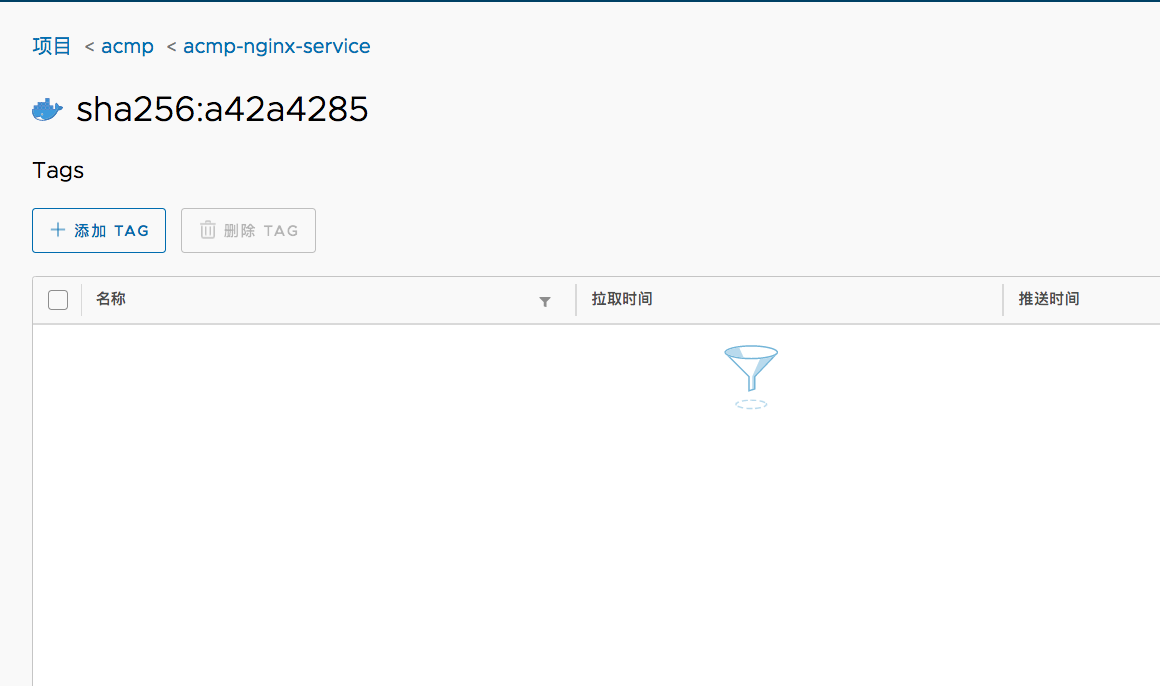
tags << t.name

}

return tags

}

最终效果：【Harbor镜像已经清空了】



<https://player.bilibili.com/player.html?bvid=BV1wh411a7Yd>

