Java 开发者必看！手把手带你搞定 Jenkins +Maven 仓库 +Docker 仓库 + 部署 + 自动更新 pom 版本 一条龙部署服务

作者：MingGH

原文链接：https://ld246.com/article/1672835504167

来源网站：链滴

原文发布于：Java开发者必看！手把手带你搞定Jenkins+Maven仓库+Docker，欢迎使用 RSS 订阅取最新更新。

1. 写在开头

为了调通这长长的一串花费了整整4天时间。。。写完后将之前写的API都采取了这样的方式进行部署极大程度的压缩了需要我部署的时间，还是非常值得的。

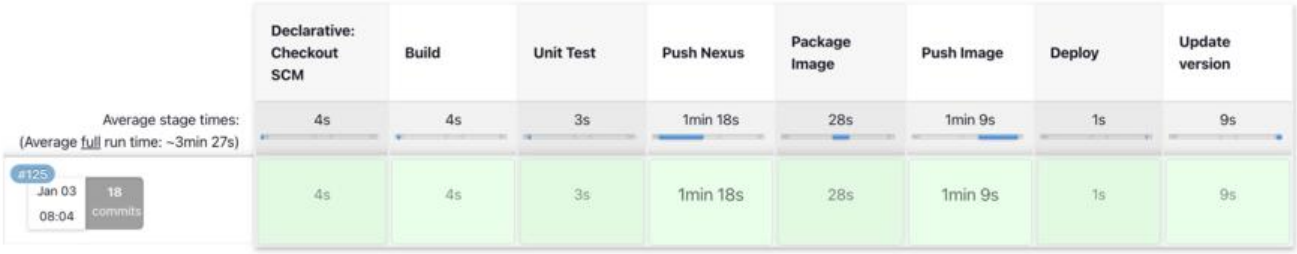
比起上次写的 **使用Jenkins对springboot项目进行docker镜像一键部署,jenkins + docker + sprin boot** 集成了更多的内容，也更加动态化。

因为流程很长，如果你在看这篇博客的时候有任何问题可以通过博客主页的邮箱联系我，我会提供我帮助到的。

1.1 实现的功能

- 一次构建，可以完成如下所有操作

checkout code → build jar package → push package to nexus → build docker image → pu h image to repositories → deploy → update pom.xml version



当中没有接入**Sonarqube**或者**墨菲安全**之类的stage是因为我把这个部份做到了code一提交时触发的**itHub Actions**中。

废话不多说，马上开始教程。

2. 搭建过程

2.1 准备内容

搭建环境：

- 一个简单的Spring Boot项目，使用Java17，我已经准备好了这个，建议先clone到本地： https://github.com/MingGH/demo-springboot-simple
- 阿里云账号（推送nexus和docker image）
- 一台已经安装了docker的电脑或者服务器，如果服务器安装docker有难度，可以参考这篇博客：**用官方安装脚本自动安装**

2.2 通过Docker安装Jenkins

创建一个目录，用来存在Jenkins的数据

```
mkdir -p /dockerData/jenkins/jenkins-data
```

进入到 /dockerData/jenkins 目录，我们在这创建 Dockerfile

```
cd /dockerData/jenkins
vim Dockerfile
```

复制以下内容到文件 Dockerfile 中

```
FROM jenkins/jenkins:2.375.1
USER root
RUN apt-get update && apt-get install -y lsb-release
RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
  https://download.docker.com/linux/debian/gpg
RUN echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
  https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
RUN jenkins-plugin-cli --plugins "blueocean:1.26.0 docker-workflow:563.vd5d2e5c4007f"
```

从Dockerfile中构建镜像

```
docker build -t myjenkins-blueocean:2.375.1-1 .
```

构建完成之后，使用docker images可以看到刚刚构建的镜像。

运行一个Jenkins容器

```
docker run \
  -u root \
  --name jenkins \
  --restart=on-failure \
  --detach \
  --publish 8080:8080 \
  --publish 50000:50000 \
  --volume /etc/localtime:/etc/localtime \
  --volume /var/run/docker.sock:/var/run/docker.sock -v /usr/bin/docker:/usr/bin/docker \
  --volume /dockerData/jenkins/jenkins-data:/var/jenkins_home \
  myjenkins-blueocean:2.375.1-1
```

对当中的一些参数进行解释：

- -u root 容器中的进程以root用户权限运行
- --restart=on-failure如果容器由于错误而退出，则将其重新启动
- --detach 保持容器在后台持续运行
- --publish 8080:8080 映射宿主机8080端口给容器8080端口
- --publish 50000:50000映射宿主机50000端口给容器50000端口
- --volume /etc/localtime:/etc/localtime 容器时间如何与宿主机同步
- --volume /var/run/docker.sock:/var/run/docker.sock -v /usr/bin/docker:/usr/bin/docker 挂宿主机的 /var/run/docker.sock 给容器，这样Jenkins容器就可以调用宿主机的docker，创建其他的

器服务于CICD

- --volume /dockerData/jenkins/jenkins-data:/var/jenkins_home 挂载Jenkins容器的数据到宿机目录下

解决ECDSA host key is known for github.com and you have requested strict checking问题

运行成功之后，别急，还有一步需要操作，进入到Jenkins容器配置ssh，如果不进行配置当拉取代码时候会抛出异常ECDSA host key is known for github.com and you have requested strict checki g，你也可以在这篇文章找到更加详细的说明：Jenkins执行pipeline抛出异常No ECDSA host key is nown for github.com and you****

进入Jenkins容器并执行以下操作

```
docker exec -it -u root jenkins /bin/bash
mkdir -p /root/.ssh
cd ~/.ssh/
touch known_hostsknown_hosts
ssh-keyscan github.com >> ~/.ssh/known_hosts
```

你会看到以下输出，就表示ok了

```
root@240a8671ca6c:~/.ssh# ssh-keyscan github.com >> ~/.ssh/known_hosts
# github.com:22 SSH-2.0-babeld-408889af
# github.com:22 SSH-2.0-babeld-408889af
# github.com:22 SSH-2.0-babeld-408889af
# github.com:22 SSH-2.0-babeld-408889af
# github.com:22 SSH-2.0-babeld-408889af
root@240a8671ca6c:~/.ssh#
```

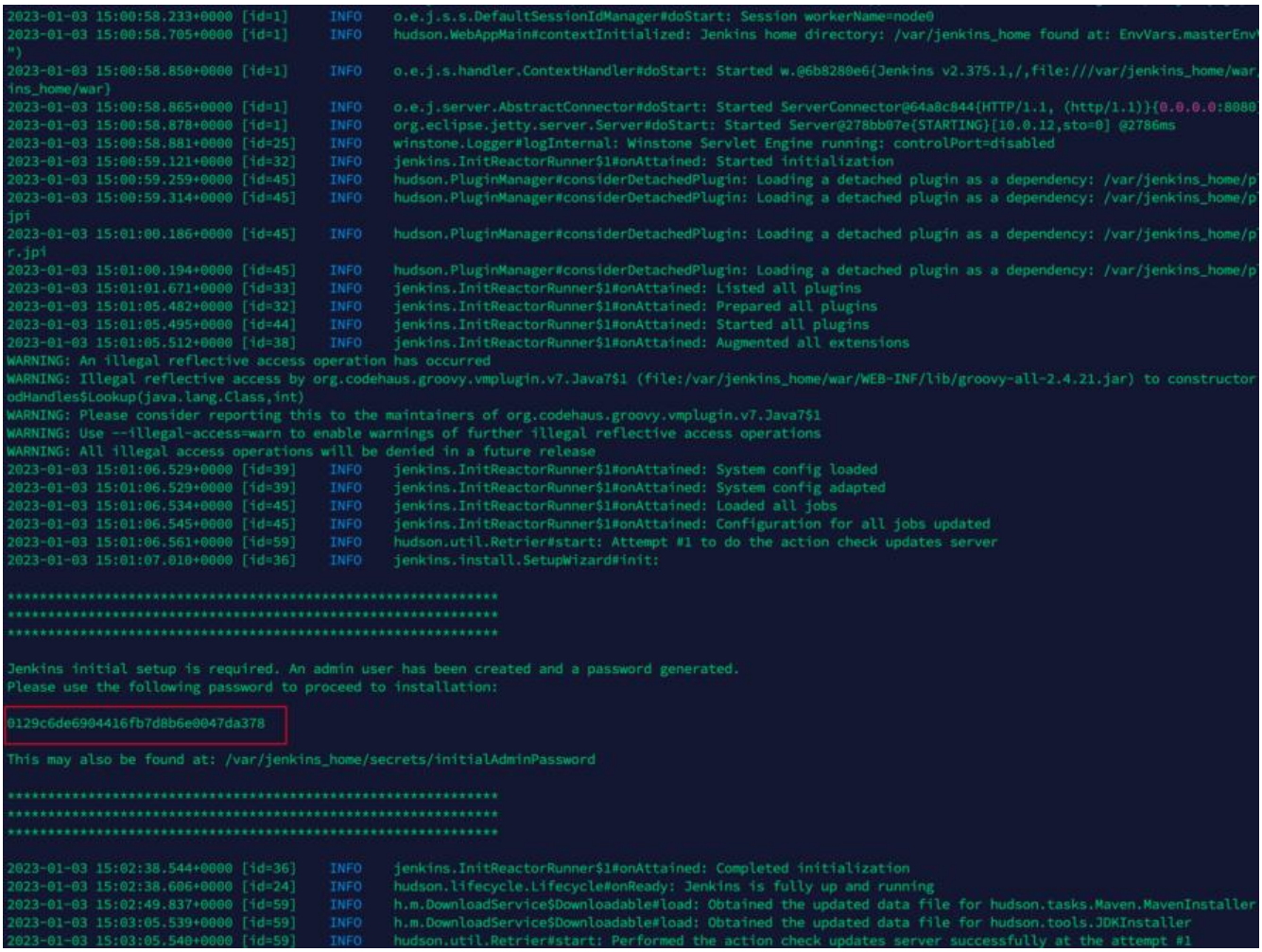
然后输出 exit可以直接推出容器

如果你是某某云的服务器，需要到某某云的控制台开放安全组端口才能进行访问

现在我们就可以以 ip+端口的形式进行访问到Jenkins，打开浏览器 http://服务器公网ip:8080 提示们需要输入密码。

原文链接：Java 开发者必看！手把手带你搞定 Jenkins+Maven 仓库 +Docker 仓库 + 部署 + 自动更新 pom 版本 一条龙部署服务

我们可以直接使用 docker logs jenkins 拿到密码



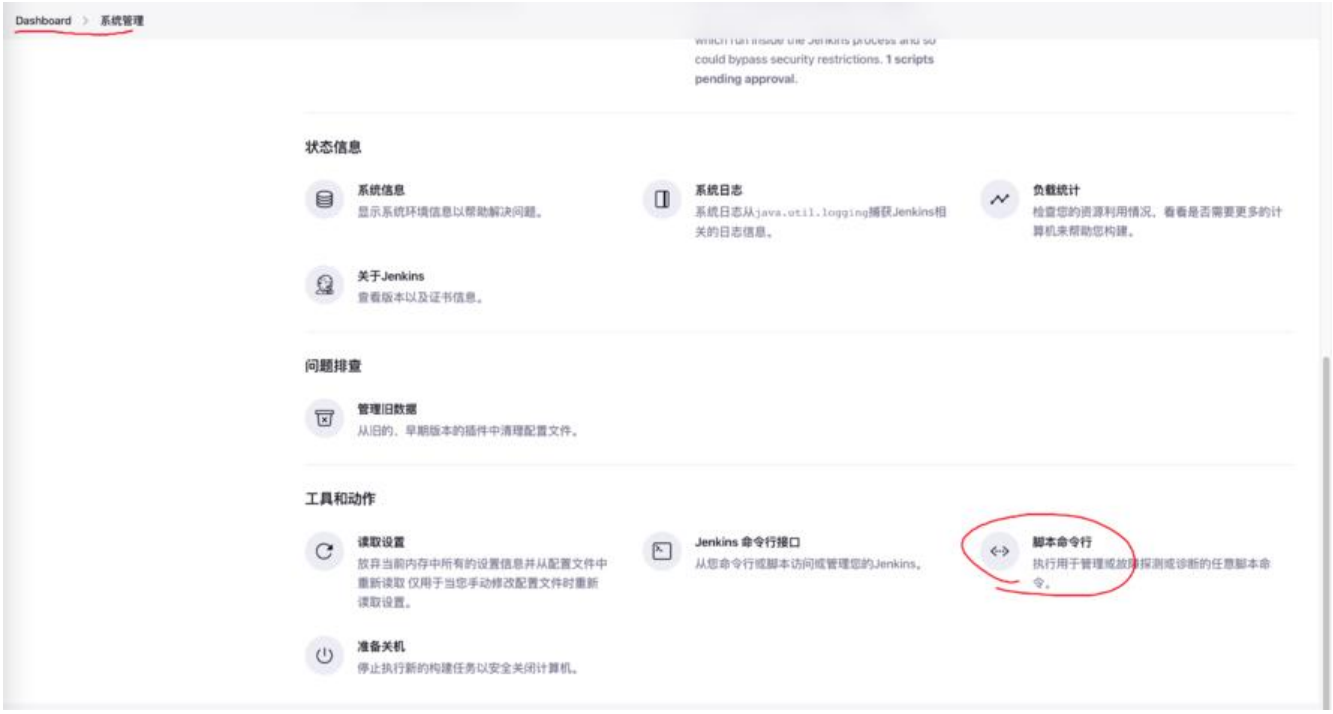
2.3 Jenkins配置：时区

选择安装完推荐的插件之后，Jenkins会进行重启，等待一段时间，然后刷新登录

原文链接：Java 开发者必看！手把手带你搞定 Jenkins+Maven 仓库 +Docker 仓库 + 部署 + 自动更新 pom 版本 一条龙部署服务

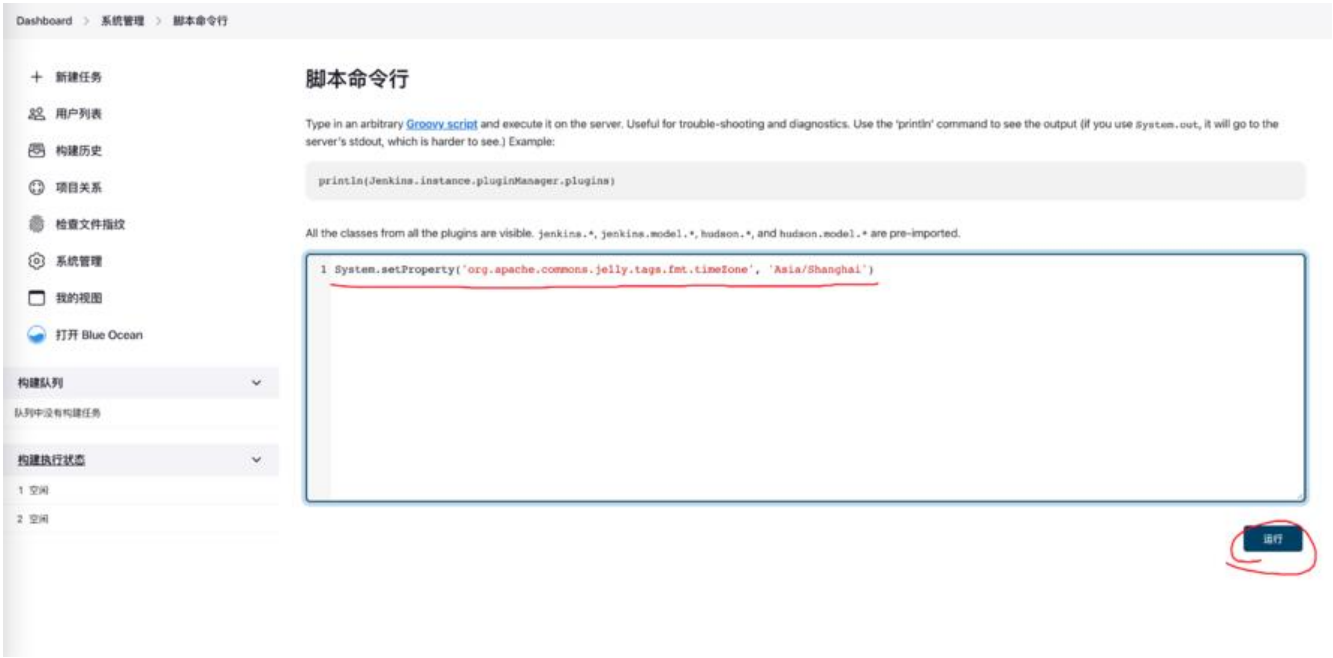
时区配置

进入到 系统管理→脚本命令行



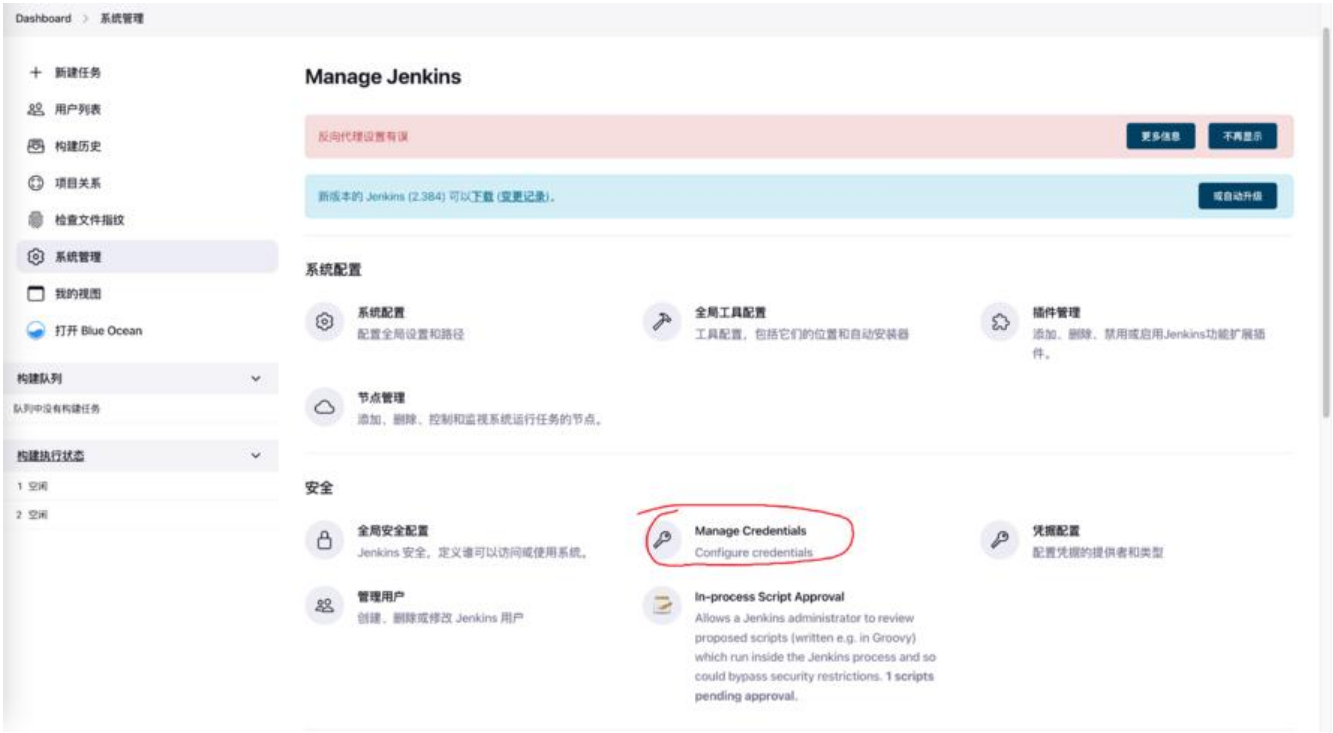
执行如下命令，设置时区为上海

System.setProperty('org.apache.commons.jelly.tags.fmt.timeZone', 'Asia/Shanghai')

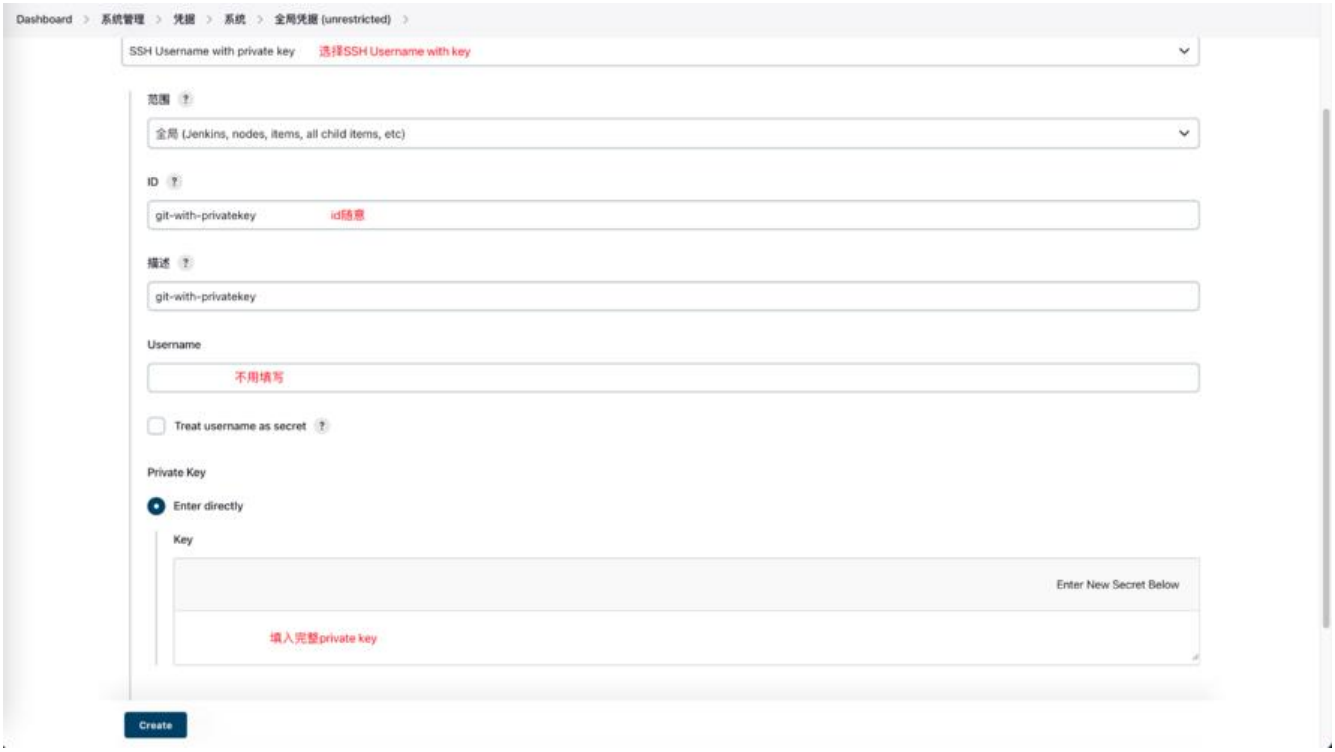


2.4 Jenkins配置：GitHub凭据

回到系统管理，点击 Manage Credentials



创建新的凭据



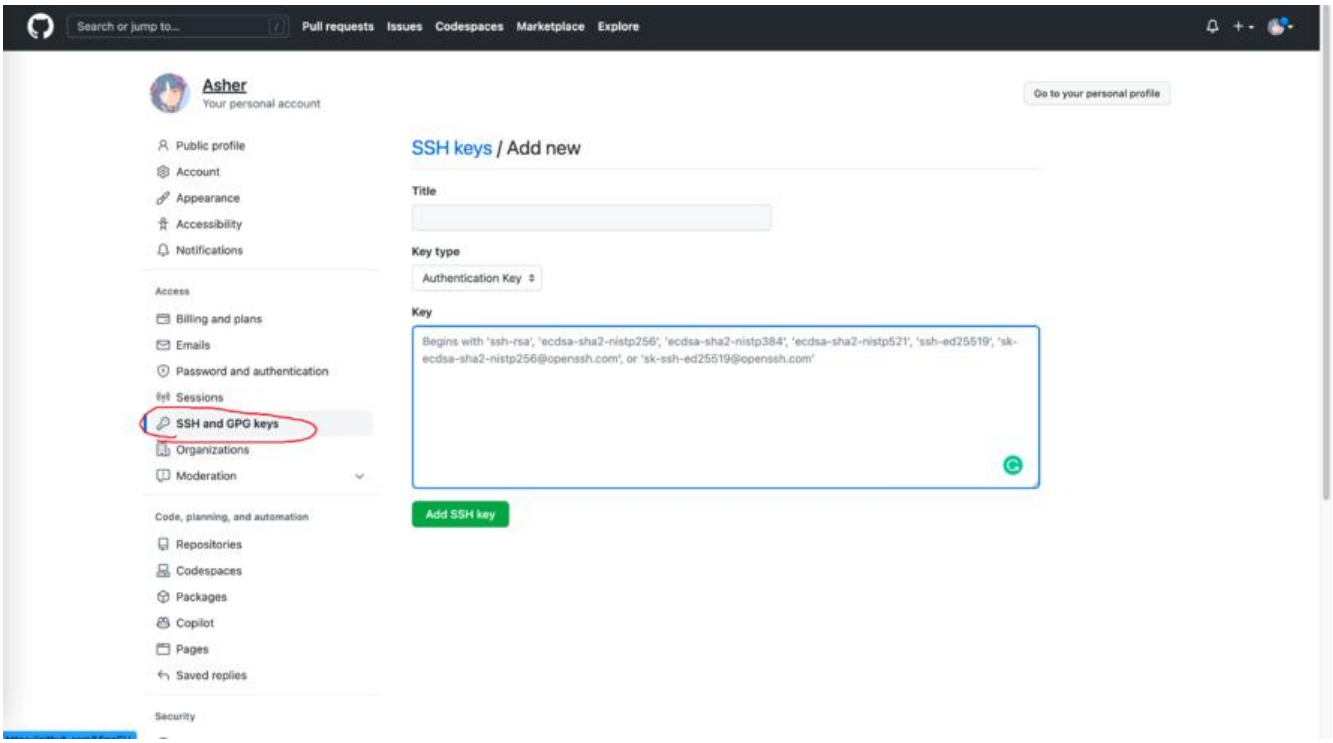
private-key的来源：在部署Jenkins的服务器上生成对应的公钥

ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

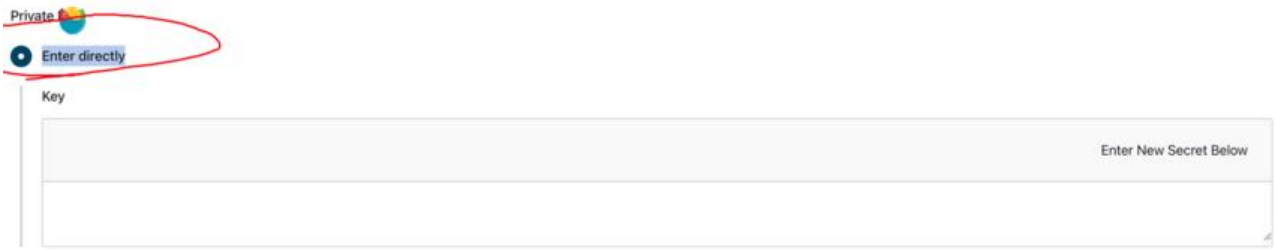
会有三个文件生成

```
[root@ecs-205431 .ssh]# ls
authorized_keys  config  config.save  id_rsa  id_rsa.pub  known_hosts
[root@ecs-205431 .ssh]#
```

然后公钥文件 id_rsa.pub中的内容复制到GitHub SSH配置中



然后把私钥文件 id_rsa 中的内容配置到刚刚Jenkins的那个地方



id_rsa的文件内容类似：

-----BEGIN OPENSSH PRIVATE KEY-----

xxxxxxxxxxxxxxx==
-----END OPENSSH PRIVATE KEY-----

生成公钥的问题可以参考这篇博客：**Git SSH公钥配置**

2.5 准备阿里云效的Maven仓库

地址：https://packages.aliyun.com/maven

注册并登录进去之后，我们可以看到有两个对应的仓库，**生产库-release** 和 **非生产库-snapshot**，别用来存放我们不同环境的jar包，如果你的 pom.xml中的version带 **-snapshot**，那么推送的就是**生产库**， 如果不带 **-snapshot** ，推送的就是生产库。这里需要注意的是，生产库的jar包是不允许重推送的。

点进**非生产库-snapshot**，在仓库指南中，我们下载对应的settings.xml文件，这个文件当Jenkins需推送Jar到Maven仓库的时候需要用到

在旁边的包文件列表可以看到之后Jenkins推送上去的包

将这个文件上传到服务器 /root/.m2/目录下

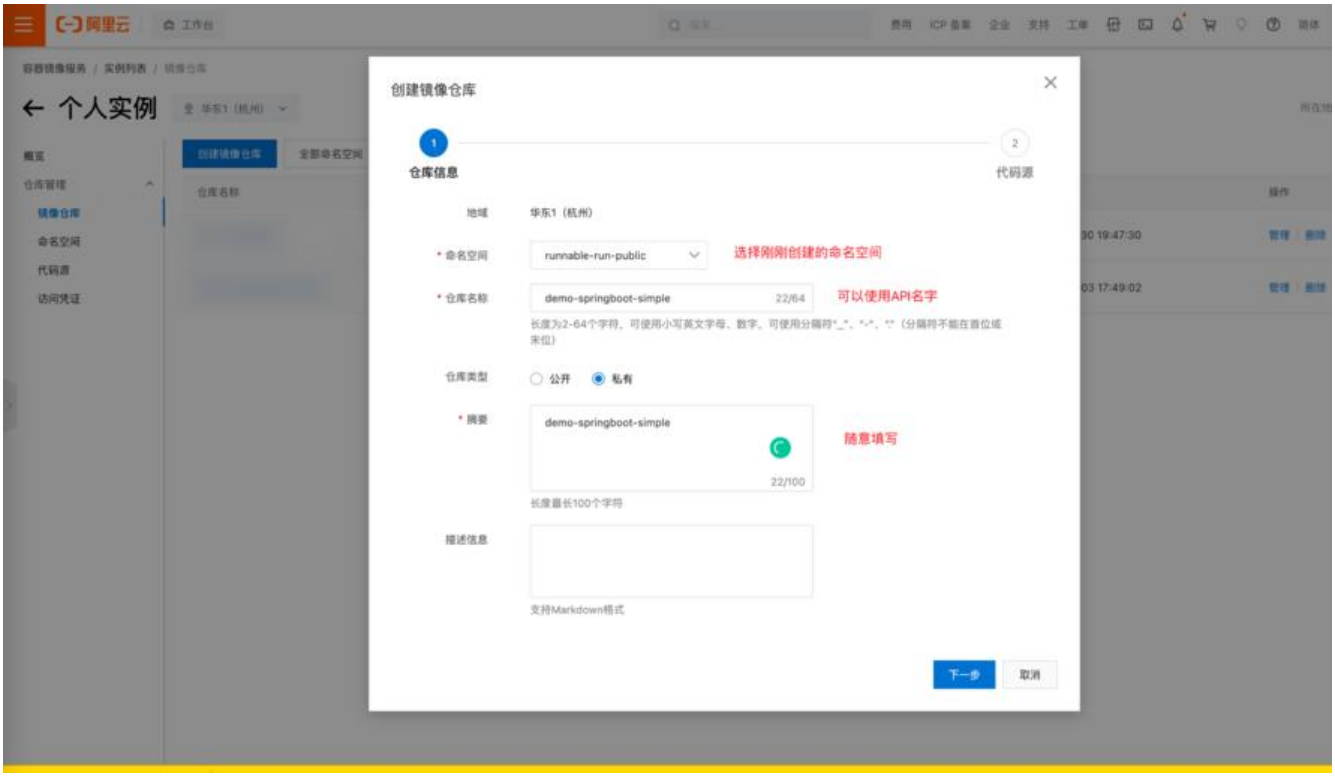
2.6 准备Docker容器镜像仓库

阿里云容器镜像服务地址： https://cr.console.aliyun.com/

登录之后选择个人版

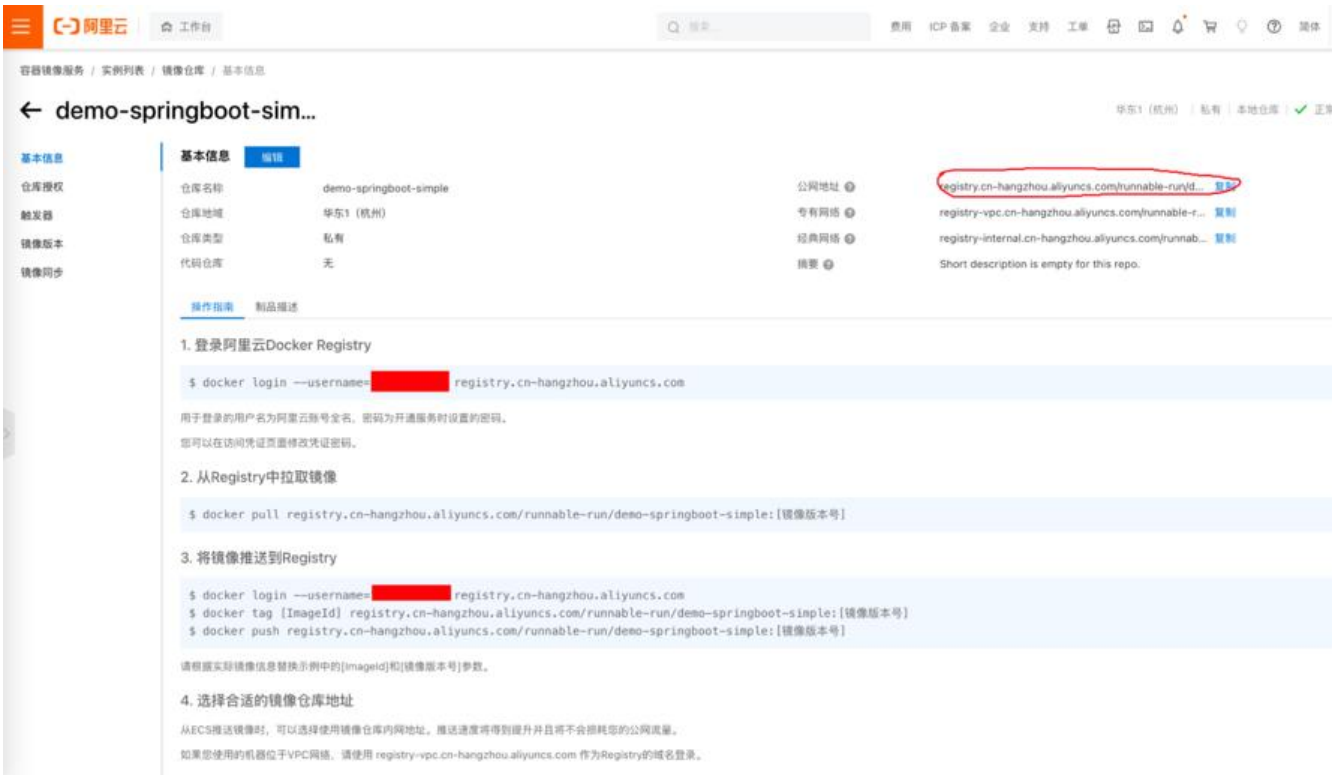
创建命名空间

创建API对应的镜像仓库



原文链接：Java 开发者必看！手把手带你搞定 Jenkins+Maven 仓库 +Docker 仓库 + 部署 + 自动更新 pom 版本 一条龙部署服务

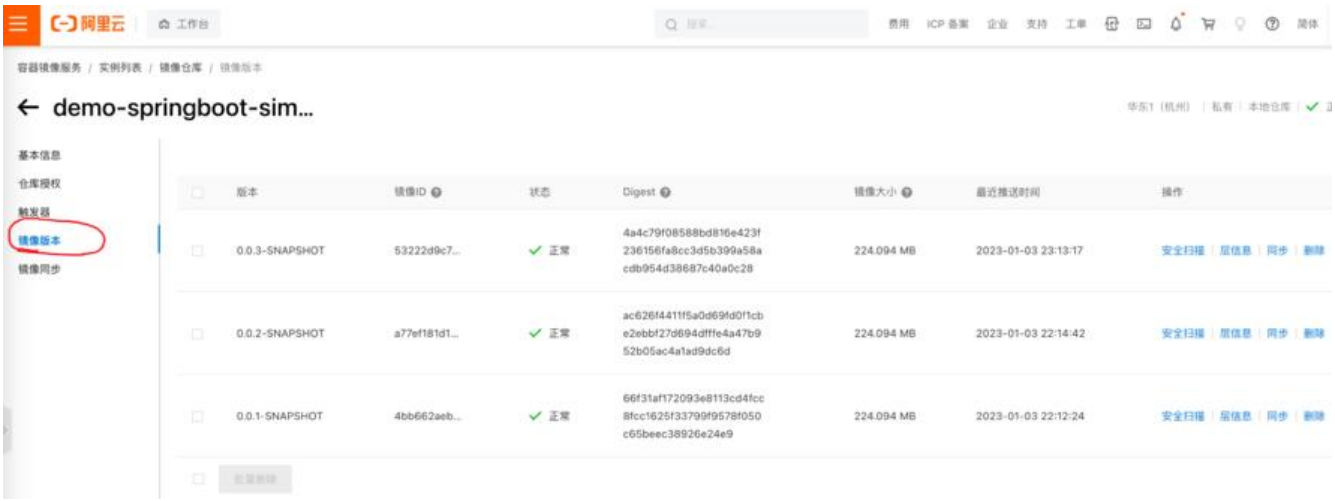
创建完成点击进去 基本信息 一栏有公网地址，这就是Docker Image需要推送的地址，我们可以先复出来。



设置密码

在个人版中，访问凭证中，设置 固定密码 ，这个密码之后会在项目demo-springboot-simple 中的eploy_docker.sh 用到。

之后Jenkins推送镜像上来，可以在这里 镜像版本找到对应的镜像

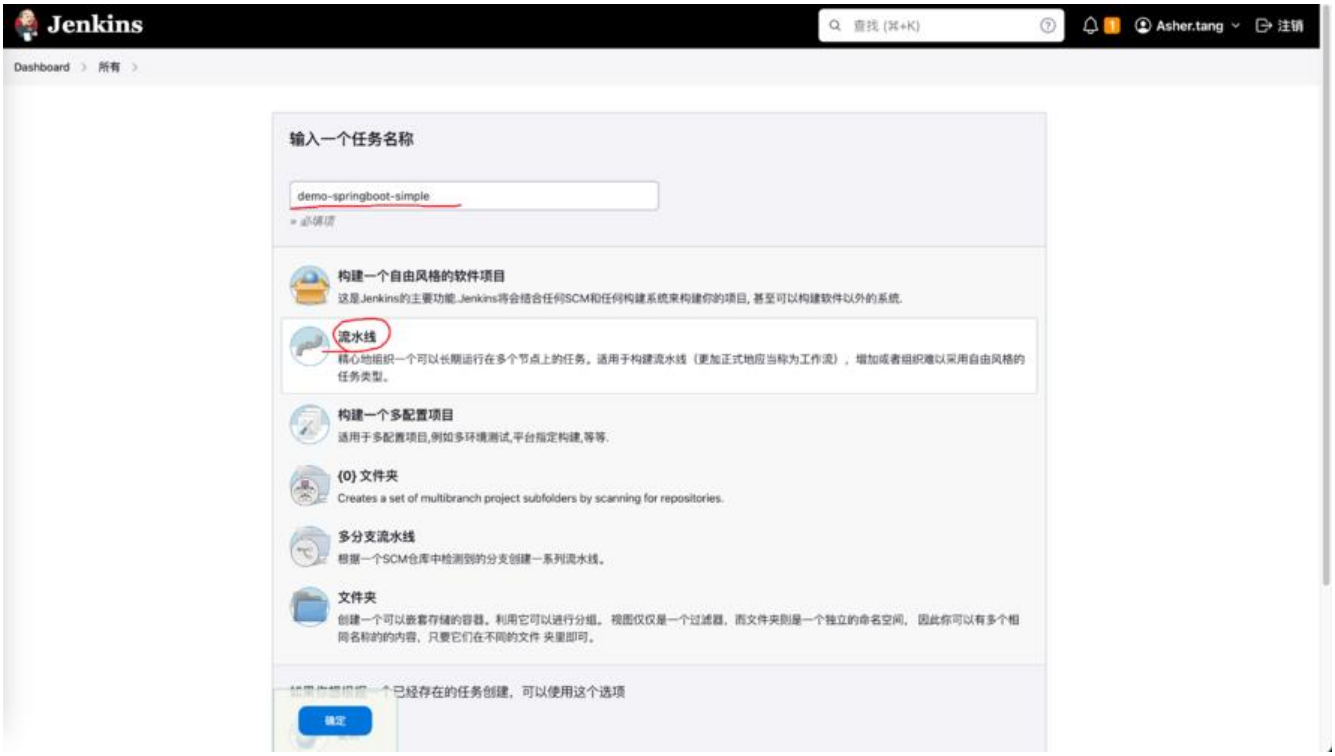


3. 通过pipeline把上面内容都串起来

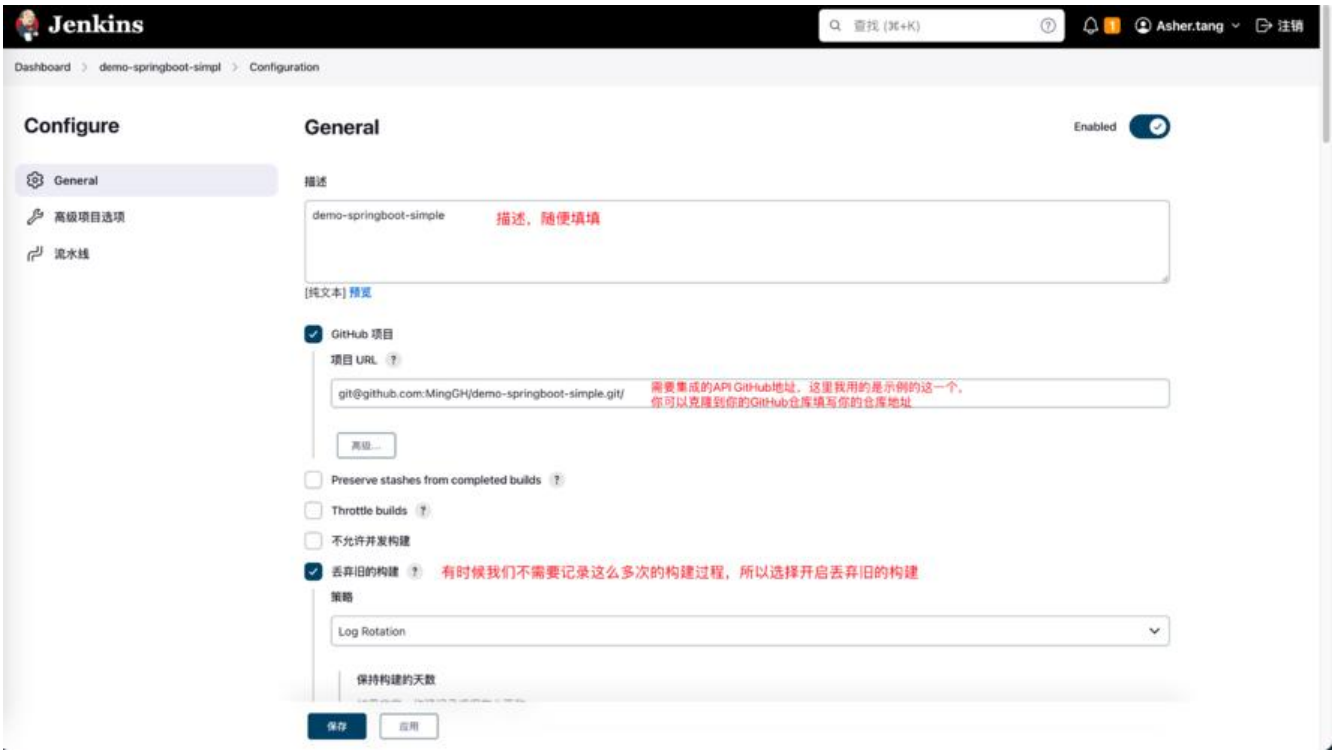
有了上面的准备的内容，那我们就可以通过Jenkins pipeline把这些内容都串在一起了。

3.1 Jenkins新建pipeline

再次登录Jenkins，这次我们新建一条pipeline,pipeline的名字就叫API的名字，这里大家照着我的配一样的填写就行，需要注意的地方我会说出来。

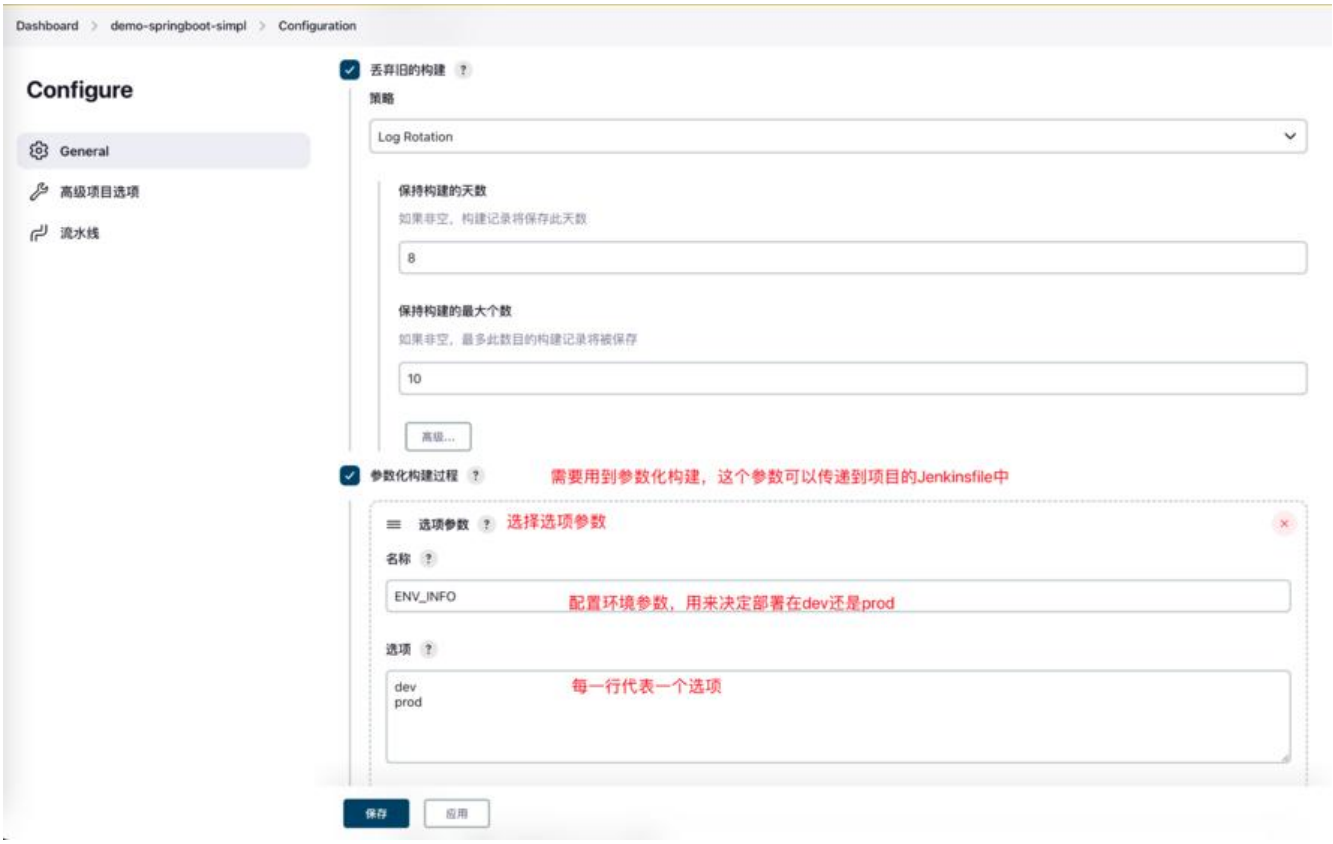


点击确定后配置一些参数



添加参数化构建过程，总共会添加三个参数ENV_INFO，GIT_Branch，Deploy_Port

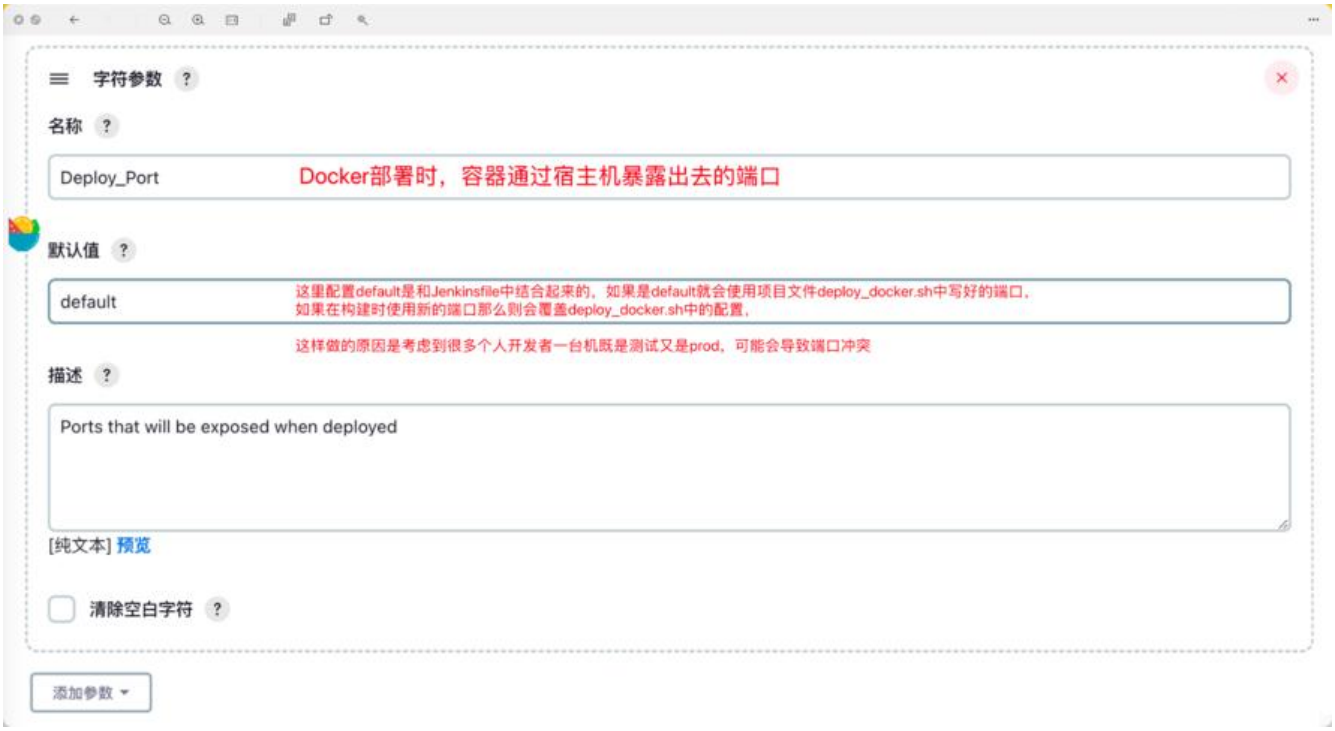
ENV_INFO



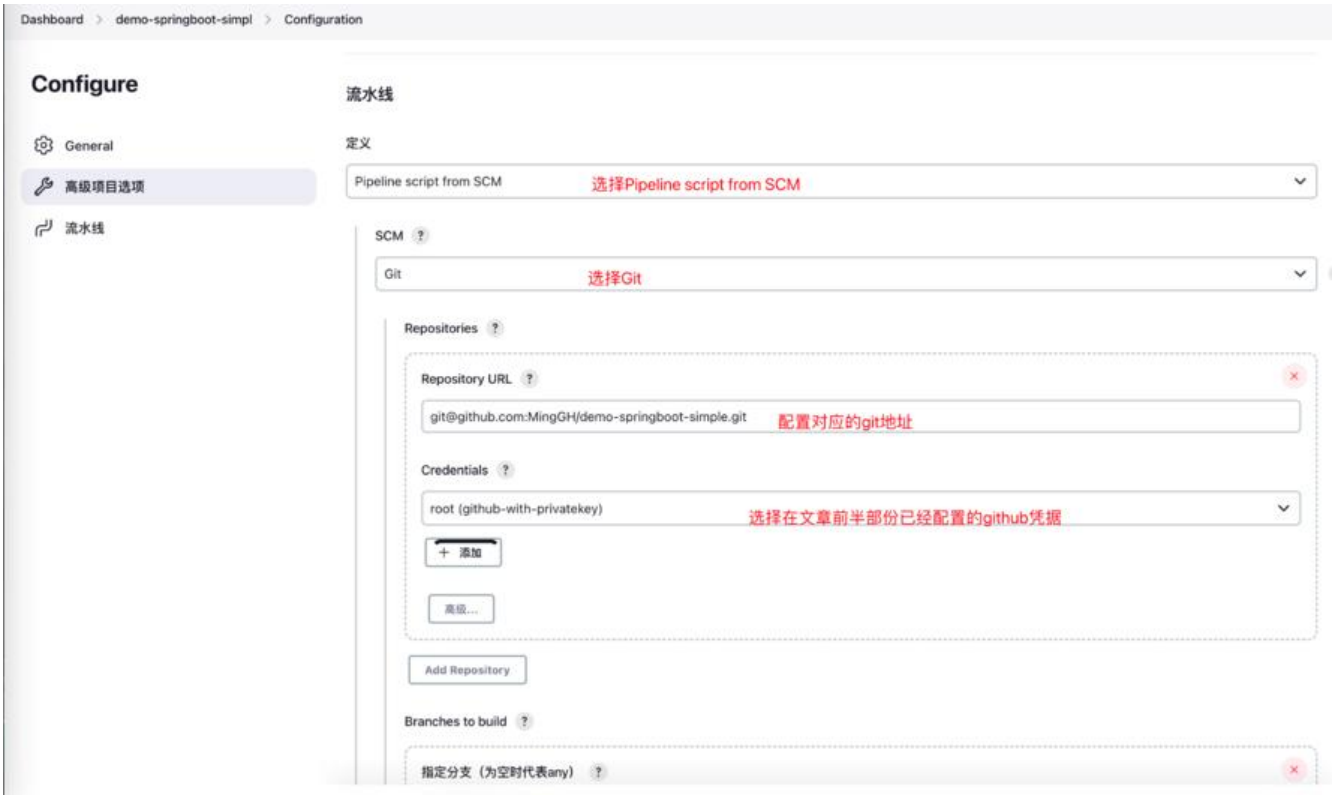
GIT_Branch

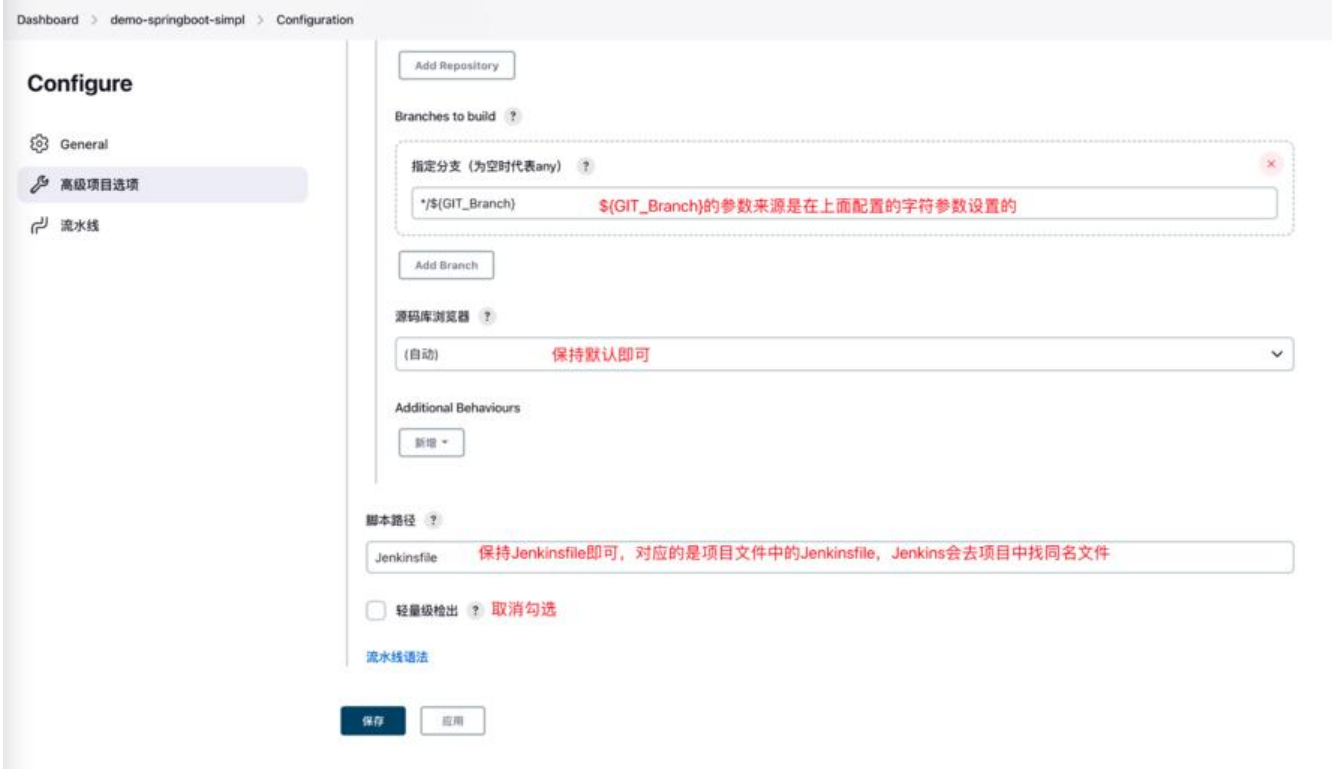

字符参数
选择字符参数
名称
GIT_Branch
这个参数用于决定检出代码时需要用哪个分支
默认值
develop
因为我们多半部署的都是dev环境，所以填写测试环境用的分支即可
描述
[纯文本] 预览
清除空白字符

Deploy_Port


字符参数
名称
Deploy_Port
Docker部署时，容器通过宿主机暴露出去的端口
默认值
default
这里配置default是和Jenkinsfile中结合起来的，如果是default就会使用项目文件deploy_docker.sh中写好的端口，
如果在构建时使用新的端口那么则会覆盖deploy_docker.sh中的配置，
这样做的原因是考虑到很多个人开发者一台机既是测试又是prod，可能会导致端口冲突
描述
Ports that will be exposed when deployed
[纯文本] 预览
清除空白字符
添加参数

Pipeline script from SCM





点击新建，此时一条pipeline就创建成功，是不是已经迫不及待的想点击build试试？等等，先听我把面这一部份讲完，当出现问题时才知道去哪找解决方案

3.2 项目demo-springboot-simple 讲解

回到最开始需要让你下载那个项目：**demo-springboot-simple**

application-dev.yaml 和 application-prod.yaml

项目是一个很简单的Spring Boot项目, 默认端口是 5002。定义了两个不同的配置文件，分别是 application-dev.yaml 和 application-prod.yaml 用来模拟不同环境时切换不同的参数。

HelloController

创建了一个 HelloController用来演示最简单的GET请求，会根据不同的部署环境，返回不同的内容。

HelloController的内容如下：

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import run.runnable.demospringbootsimple.config.AppConfig;

/**
 * @author Asher
 * on 2023/1/3
 */
@Controller
public class HelloController {
```

```
    @GetMapping("/hello")
    @ResponseBody
    public String hello(){
        return "hello " + appConfig.getEnv();
    }

    private AppConfig appConfig;
    @Autowired
    public void setAppConfig(AppConfig appConfig) {
        this.appConfig = appConfig;
    }
}
```

3.3 项目配置： Jenkinsfile (有需要修改内容)

当Jenkins在面板上点击构建时，会找到对应目录下的Jenkinsfile,所以第一个我们需要说明的就是Jenkinsfile。

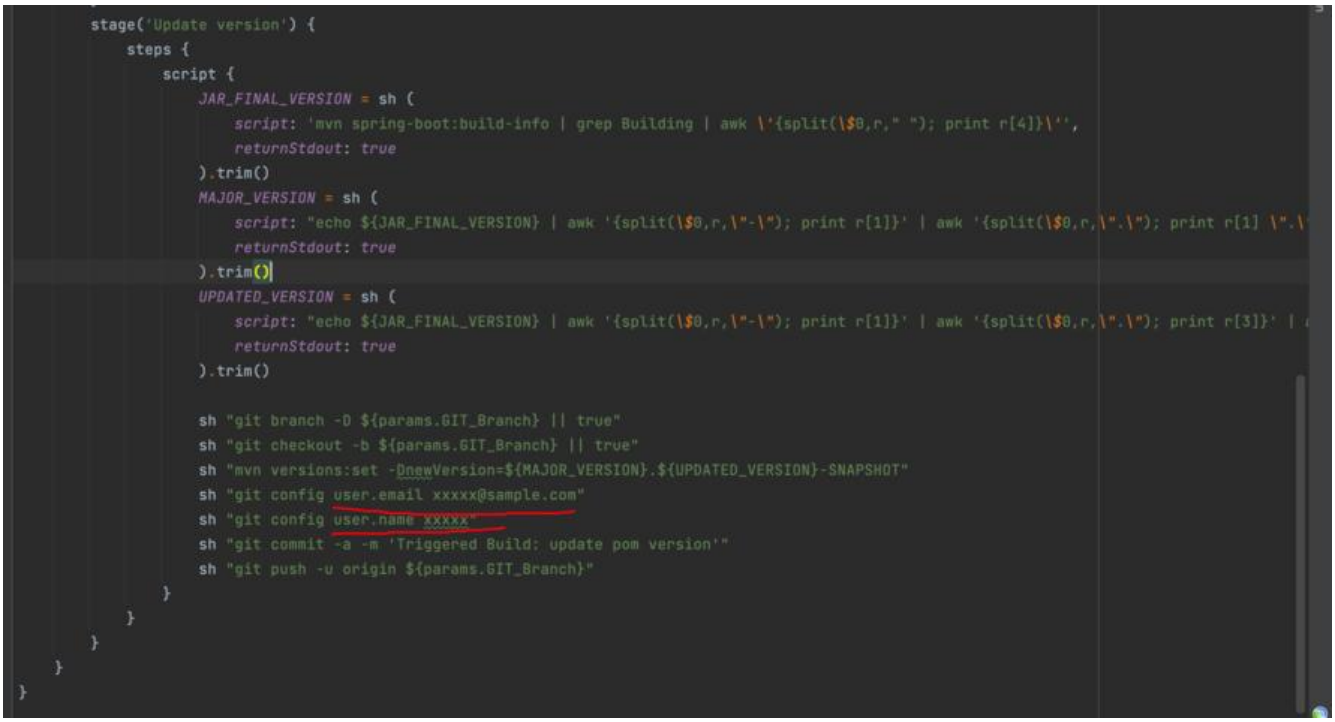
在Jenkinsfile中我定义了7个stage,分别是**Build, Unit Test, Push Nexus, Package Image, Push Image, Deploy, Update version**

当Jenkins进行build的时候就会出现对应的这几个阶段。

这里因为篇幅过长，使用了新的一篇文章来进行讲解：**demo-springboot-simple中Jenkinsfile详解**

需要修改的内容为：

在这个 Update version Stage中需要修改你的git邮箱地址和username，因为在步骤2.4中已经**配置公钥在GitHub上**，且在Jenkins agent 中配置了 /root/.ssh的目录映射，所以这里只需要配置用户名邮箱就行。



```
        stage('Update version') {
            steps {
                script {
                    JAR_FINAL_VERSION = sh (
                        script: 'mvn spring-boot:build-info | grep Building | awk \'{split(\$0,r," "); print r[4]}\'',
                        returnStdout: true
                    ).trim()
                    MAJOR_VERSION = sh (
                        script: "echo ${JAR_FINAL_VERSION} | awk '{split(\$0,r,\"-\"); print r[1]}' | awk '{split(\$0,r,\".\"); print r[1] \".\"
                        returnStdout: true
                    ).trim()
                    UPDATED_VERSION = sh (
                        script: "echo ${JAR_FINAL_VERSION} | awk '{split(\$0,r,\"-\"); print r[1]}' | awk '{split(\$0,r,\".\"); print r[3]}' |
                        returnStdout: true
                    ).trim()

                    sh "git branch -D ${params.GIT_Branch} || true"
                    sh "git checkout -b ${params.GIT_Branch} || true"
                    sh "mvn versions:set -DnewVersion=${MAJOR_VERSION}.${UPDATED_VERSION}-SNAPSHOT"
                    sh "git config user.email xxxxx@sample.com"
                    sh "git config user.name xxxxx"
                    sh "git commit -a -m 'Triggered Build: update pom version'"
                    sh "git push -u origin ${params.GIT_Branch}"
                }
            }
        }
    }
}
```

3.5 项目配置：deploy_docker.sh

说完了Jenkinsfile，希望你已经明白Jenkins执行时的主要的流程，那么在 deploy_docker.sh中的内是docker部署到本地和docker推送到**Docker容器镜像仓库时的一些命令。**

你需要配置的内容如图：

```
#!/bin/bash

DEFAULT_PORT=5002
APP=demo-springboot-simple

PROG_NAME=$0
ACTION=$1
TAGNAME=$2
PROFILE=$3
PASS_PORT=$4

APP_PORT=${PASS_PORT:-$DEFAULT_PORT}
APP_NAME=${APP}-${PROFILE}
APP_HOME=/dockerData/runnable-run/${APP_NAME}
APP_OUT=${APP_HOME}/logs

# 阿里云仓库命名空间
APP_NAMESPACE=[redacted]
# 推送至阿里云仓库的地址
APP_REGISURL=registry.cn-hangzhou.aliyuncs.com
APP_RESP=${APP_REGISURL}/${APP_NAMESPACE}/${APP}
# 阿里云账号
APP_USERNAME=[redacted]
# 阿里云配置个人容器镜像服务时，设置的密码
APP_PASSWORD=[redacted]

mkdir -p ${APP_HOME}
mkdir -p ${APP_HOME}/logs

usage() {
    echo "Usage: $PROG_NAME {start|stop|restart|push}"
    exit 2
}

start_application() {
    echo "starting start_application"
    docker run \
     -e "SPRING_PROFILES_ACTIVE=${PROFILE}" \
     --log-opt max-size=1024m \
```

3.4 项目配置：pom.xml (有需要修改内容)

需要修改的配置

在 pom.xml 文件中你需要修改properties中的内容，

- docker.namespace是在阿里云的Docker容器镜像仓库时创建的。
- docker.registry.address 则是镜像仓库基本信息中的公网地址，例如：registry.cn-hangzhou.aliyncs.com

```
    <properties>
        <java.version>17</java.version>
        <docker.namespace></docker.namespace>
        <docker.registry.address></docker.registry.address>
    </properties>
```

在plugin中我增加了dockerfile-maven-plugin，通过这个plugin可以使用maven构建docker imag，但是需要项目中存在Dockerfile，这个会马上说到。

```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>dockerfile-maven-plugin</artifactId>
    <version>1.4.13</version>
    <configuration>
        <repository>${docker.registry.address}/${docker.namespace}/${project.artifactId}</repos
tory>
        <tag>${project.version}</tag>
        <buildArgs>
            <JAR_NAME>${project.artifactId}</JAR_NAME>
            <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
        </buildArgs>
    </configuration>
</plugin>
```

这里需要注意的是标签 <buildArgs>下定义的 <JAR_NAME> 和 <JAR_FILE> 将会传递到Dockerfil中用于进行构建镜像

3.4 项目配置：Dockerfile（无修改）

Dockerfile中配置了构建image时的一些参数，如下：

```
FROM openjdk:17-jdk-slim-buster

ARG JAR_NAME
ENV PROJECT_NAME ${JAR_NAME}
ENV PROJECT_HOME /usr/local/${PROJECT_NAME}

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
RUN echo 'Asia/Shanghai' >/etc/timezone
RUN mkdir $PROJECT_HOME && mkdir $PROJECT_HOME/logs

ARG JAR_FILE
COPY ${JAR_FILE} $PROJECT_HOME/${JAR_NAME}.jar

ENTRYPOINT java -jar -Xmn128m -Xms256m -Xmx256m $PROJECT_HOME/$PROJECT_NAME.
ar
```

当中一些参数的解释，如果需要更加详细的说明，可以参考这个：**什么是 Dockerfile?**

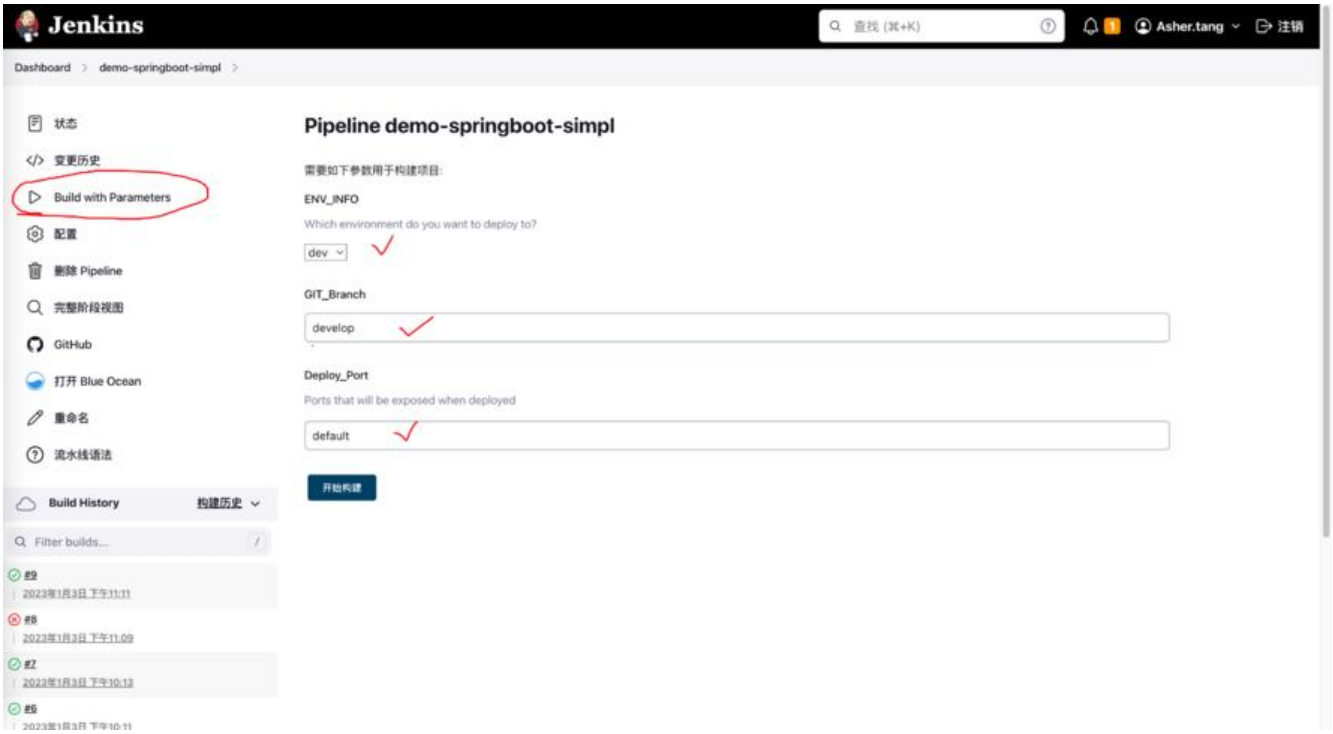
- FROM openjdk:17-jdk-slim-buster image基于openjdk:17-jdk-slim-buster
- ENV PROJECT_NAME ${JAR_NAME} 设置环境变量，定义了环境变量，那么在后续的指令中，就以使用这个环境变量。这个参数中的 ${JAR_NAME}的来源是pom.xml中的 <JAR_NAME>${project.rtifactId}</JAR_NAME>

- RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime 指定容器运行时区
- RUN mkdir $PROJECT_HOME && mkdir $PROJECT_HOME/logs 创建项目路径
- COPY ${JAR_FILE} $PROJECT_HOME/${JAR_NAME}.jar 将maven构建出来的Jar文件复制到容器具体位置
- ENTRYPOINT java -jar -Xmn128m -Xms256m -Xmx256m $PROJECT_HOME/$PROJECT_NAE.jar 容器的入口，通过java -jar进行启动，并指定运行时的一些参数

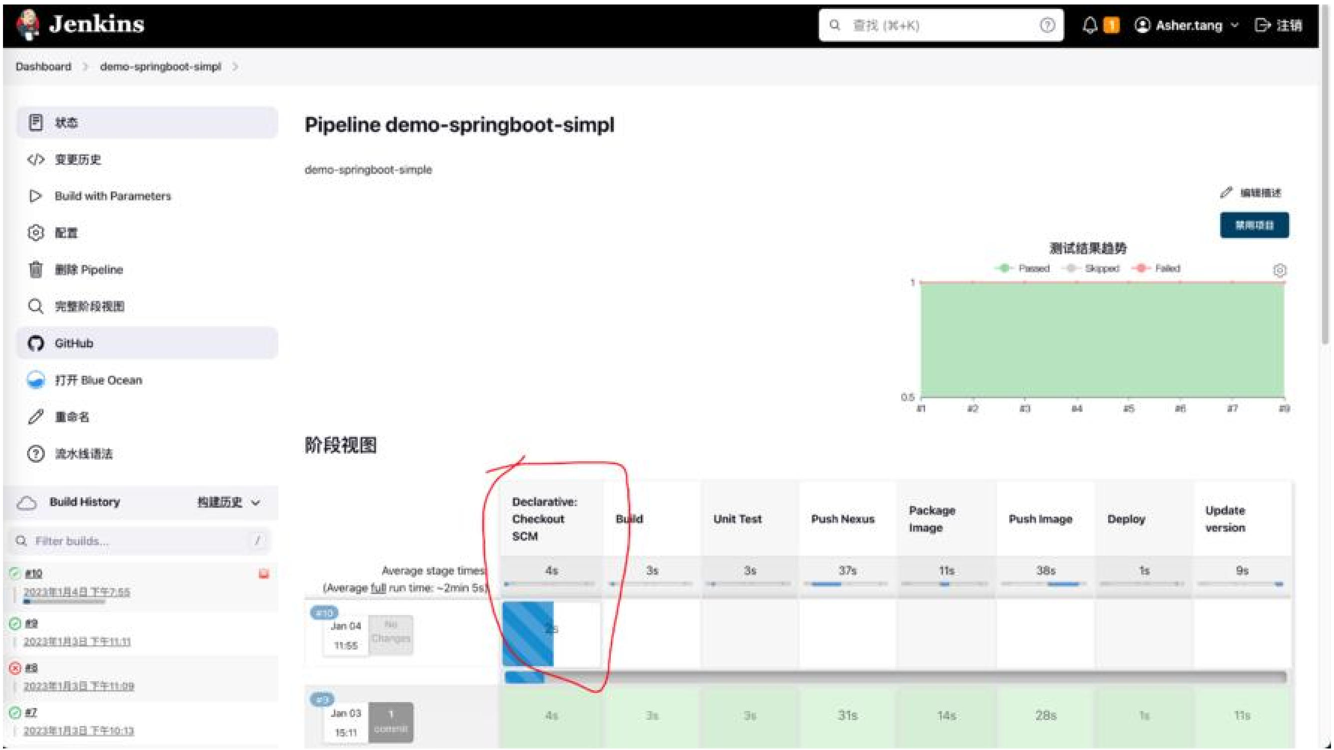
完成以上配置之后，提交你的代码到具体的分支。

4. Build with Parameters

终于到了最后一步，回到Jenkins的面板, 点击**Build with Parameters,** 填入你的参数，进行构建



此时你就可以看到pipeline已经在跑了

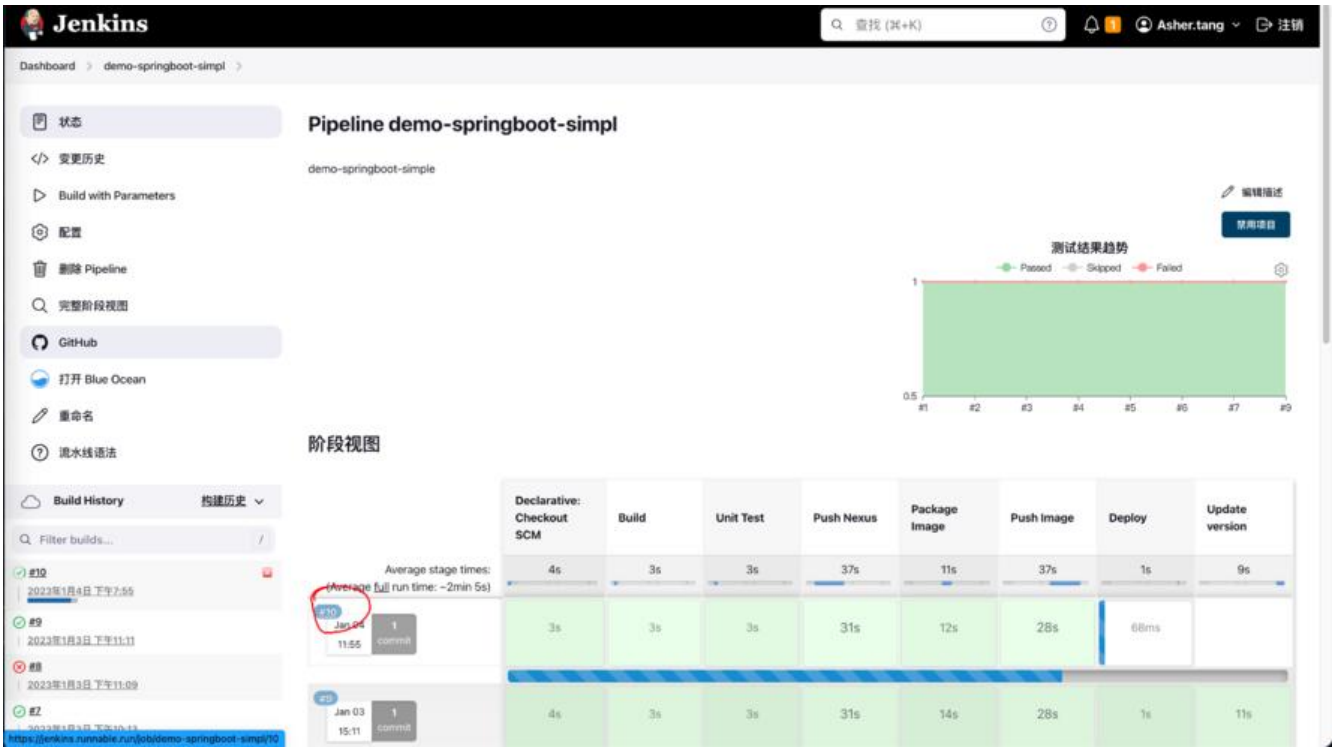


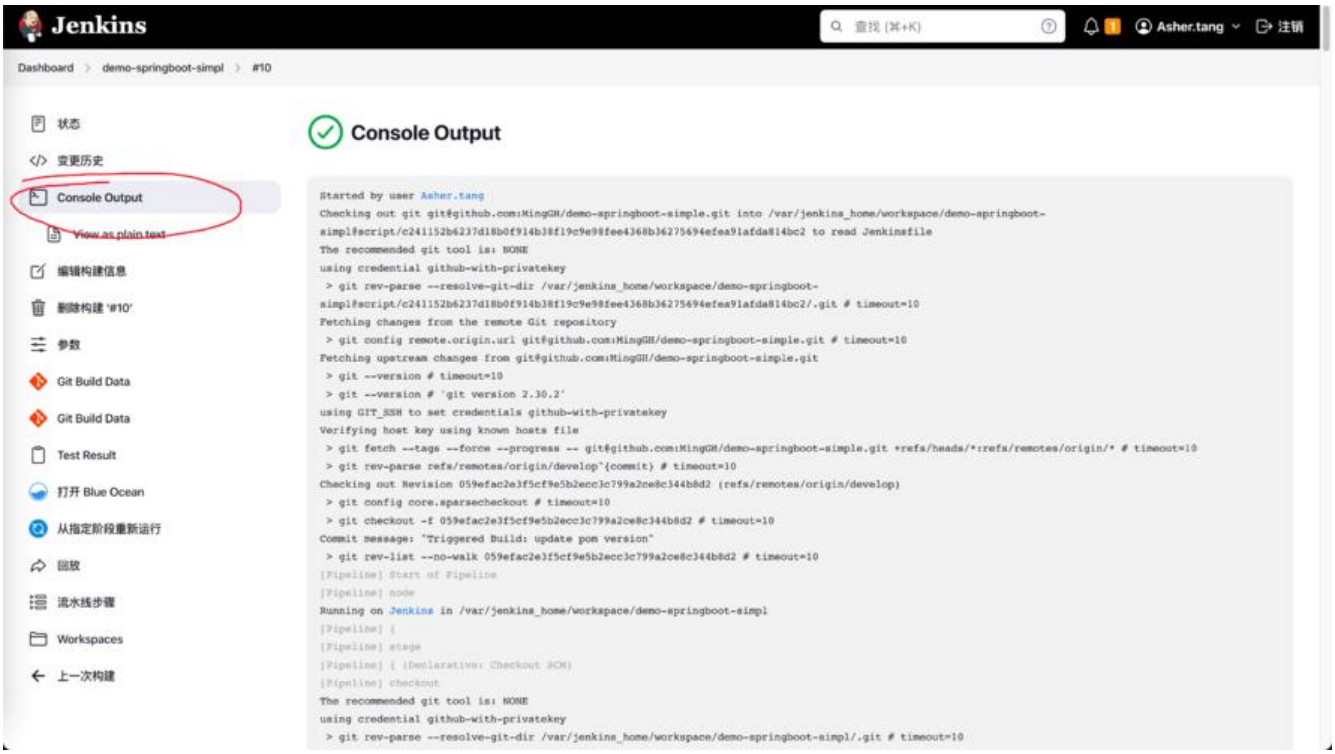
执行完成的状态应该是所有的stage都是绿色，且在你服务器上能看到有一个新的容器产生

[root@ecs-205431 ~]# docker ps -s
CONTAINER ID IMAGE COMMAND
REATED STATUS PORTS NAMES SIZE
9c0ee6e2726f registry.cn-hangzhou.aliyuncs.com/xxxx/demo-springboot-simple:0.0.4-SNAPHOT "/bin/sh -c 'java -j..." 3 minutes ago Up 3 minutes 0.0.0.0:5002->5002/tcp, :::5002->002/tcp demo-springboot-simple-dev 32.8kB (virtual 419MB)

5. 问题排查

因为流程很长，出现问题不要慌。直接点击那一个构建过程，进去看看日志。





6. 参考内容

The Problem With ‘src refspec does not match any’

Error: src refspec master does not match any – How to Fix in Git

git 获取当前分支名

git: rename local branch failed

Change System Time Zone

Jenkins分布式构建与并行构建

build-a-java-app-with-maven

Installing Jenkins

cut or awk command to print first field of first row

How to automatically increment pom version with maven, for example 1.2.0 to 1.3.0

Starting Spring Boot Application in Docker With Profile

使用Jenkins对springboot项目进行docker镜像一键部署,jenkins + docker + springboot