

Jenkins

基础知识

Jenkins简介

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

官方简介

2016年介绍:

Jenkins is an automation engine with an unparalleled[空前的] plugin ecosystem[生态系统] to support all of your favorite tools in your delivery pipelines, whether your goal is continuous integration, automated testing, or continuous delivery.

2019年介绍:

In a nutshell[简而言之], Jenkins is the leading open-source automation server. Built with Java, it provides over 1000 plugins to support automating virtually anything, so that humans can actually spend their time doing things machines cannot.

2021年介绍:

Build great things at any scale


The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

简介梳理

Jenkins 是一个专注于软件项目持续交付场景的开源管理平台，借助于各种插件，整个合理的自动化功能，在软件项目迭代过程中，最大程度解放劳动力，从而让人将精力花费在机器不擅长的场景。

官网地址: <https://www.jenkins.io/>

最新版本: 2.307(定期发布版本)、2.289(长期支持版本)



Jenkins

Build great things at any scale

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

[Documentation](#) [Download](#)

简单实践

功能定位

Use Jenkins to automate your development workflow so you can focus on work that matters most. Jenkins is commonly used for:

Building projects

Running tests to detect bugs and other issues as soon as they are introduced

Static code analysis[分析]

Deployment

Execute repetitive[重复] tasks, save time, and optimize[优化] your development process with Jenkins.



Continuous Integration and Continuous Delivery

As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.



Easy installation

Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Linux, macOS and other Unix-like operating systems.



Easy configuration

Jenkins can be easily set up and configured via its web interface, which includes on-the-fly error checks and built-in help.



Plugins

With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery toolchain.



Extensible

Jenkins can be extended via its plugin architecture, providing nearly infinite possibilities for what Jenkins can do.

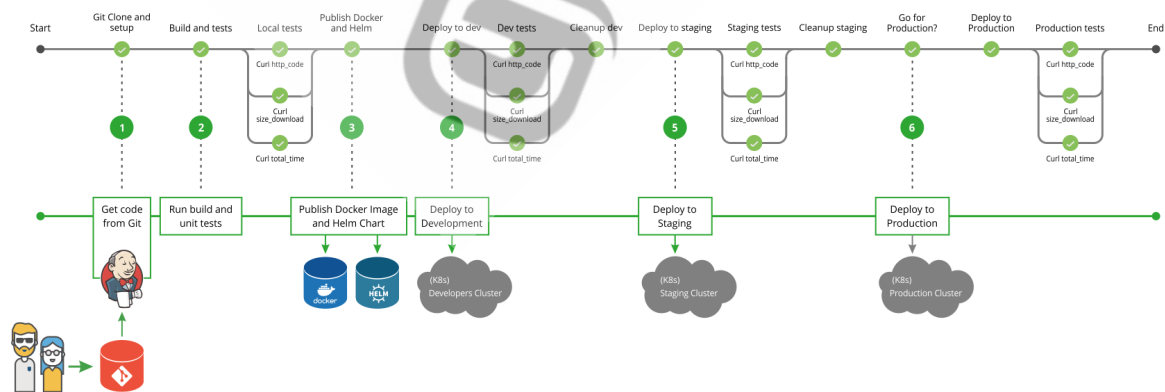


Distributed

Jenkins can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.

优势功能

流水线 交付流水线本身就是一件软件产品，这个产品的最终用户是研发和运营，其核心目的是加快公司项目快速迭代的一套解决方案。它以项目生命周期为核心轴线，将我们工作过程中为了提高研发效率的集成工具、保障项目运营的业务工具、应急处理的流程工具等有机的整合起来，从而将项目产品从研发到客户之间的不同阶段和任务串接在一起，并且以半自动或自动化的方式逐步执行，最终实现项目产品的端到端快速交付目的。它是建立在一系列的最佳实践基础之上的，比如我们的环境标准化、代码版本控制、配置管理、基础设施库、流程自动化等。



小结

环境部署

学习目标

这一节，我们从 软件安装、基本配置、小结 三个方面来学习。

软件安装

安装前提

硬件配置需求

Minimum hardware requirements:

256 MB of RAM

1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Recommended hardware configuration for a small team:

4 GB+ of RAM

50 GB+ of drive space

软件配置需求

You will need to pre-install a supported version of Java:

2.164 (2019-02) and newer: Java 8 or Java 11

2.54 (2017-04) and newer: Java 8

1.612 (2015-05) and newer: Java 7

java环境部署

创建目录

```
mkdir /data/{softs,server} -p
cd /data/softs
```

下载java或者上传java

```
ls /data/softs
```

安装java

```
tar xf jdk-8u121-linux-x64.tar.gz -C /data/server
cd /data/server/
ln -s jdk1.8.0_121/ java
```

配置java环境变量

```
cat >> /etc/profile.d/java.sh <<-EOF
# java env set
export JAVA_HOME=/data/server/java
export JRE_HOME=\$JAVA_HOME/jre
export CLASSPATH=\$JAVA_HOME/lib/tools.jar:\$JAVA_HOME/lib/dt.jar
export PATH=\$JAVA_HOME/bin:\$JRE_HOME/bin:\$PATH
EOF
```

```
source /etc/profile.d/java.sh
chmod +x /etc/profile.d/java.sh
```

检查效果

```
]# java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

检查java目录效果

```
tree -L 1 /data/server/java/
```

安装软件

安装方法1 - 软件包

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key  
add -  
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \  
/etc/apt/sources.list.d/jenkins.list'  
sudo apt-get update  
sudo apt-get install jenkins
```

安装方法2 - Docker

```
docker pull docker:dind  
docker network create jenkins  
docker run --name jenkins-docker --rm --detach --privileged --network jenkins \  
--network-alias docker --env DOCKER_TLS_CERTDIR=/certs \  
--volume jenkins-docker-certs:/certs/client --volume jenkins-  
data:/var/jenkins_home \  
--publish 2376:2376 docker:dind --storage-driver overlay2
```

安装方法3 - 软件包

```
cd /data/softs  
wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war  
java -jar jenkins.war
```

服务配置文件

编写配置文件

```
# vim /lib/systemd/system/jenkins.service  
[Unit]  
Description= jenkins server project  
  
[Service]  
User=root  
ExecStart=/data/server/java/bin/java -jar /data/server/jenkins/jenkins.war &  
ExecStop=/bin/kill -TERM ${MAINPID}  
Restart=always  
RestartSec=5  
  
[Install]  
WantedBy=multi-user.target
```

准备软件启动文件

```
cp jenkins.war /data/server/
```

重启服务

```
systemctl daemon-reload  
systemctl start jenkins.service  
systemctl status jenkins.service
```

查看启动日志

```
tail -f /var/log/syslog
```

结果显示:

jenkins安装后的家目录是 /root/.jenkins

查看端口

```
netstat -tnulp
```

浏览器访问 192.168.8.14:8080, 根据提示, 找到密码内容, 然后输入, 点击 "继续" 继续安装



推荐点击"安装推荐的插件", 他会直接进入到了下图, 点击"选择插件来安装"的话, 有一个选择的界面。

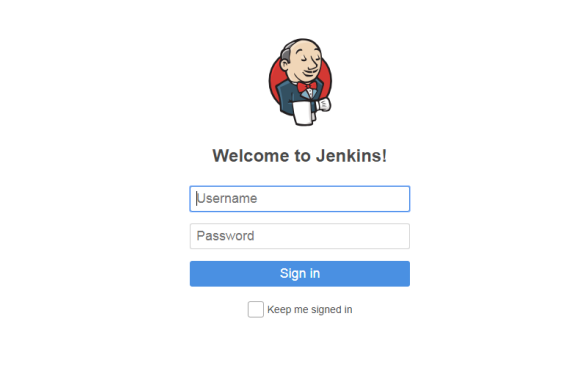


因为安装插件需要从 <https://updates.jenkins.io/update-center.json> 下载, 因为网络原因, 导致插件安装太慢了, 所以我们推荐将所有插件都取消, 然后点击安装


我们这里可以设置一个专有账号, 或者采用默认的admin用户登录, 这里我们设置一个管理账号: jenkins, 密码为123456, 全称jenkins web admin, 然后点击"保存并完成", 效果如下



对于实例配置, 我们使用默认的即可, 点击"保存并完成", 接着点击"开始使用Jenkins"即可进入jenkins 登录页面



在登录界面输入用户名和密码, 然后点击"sign in"



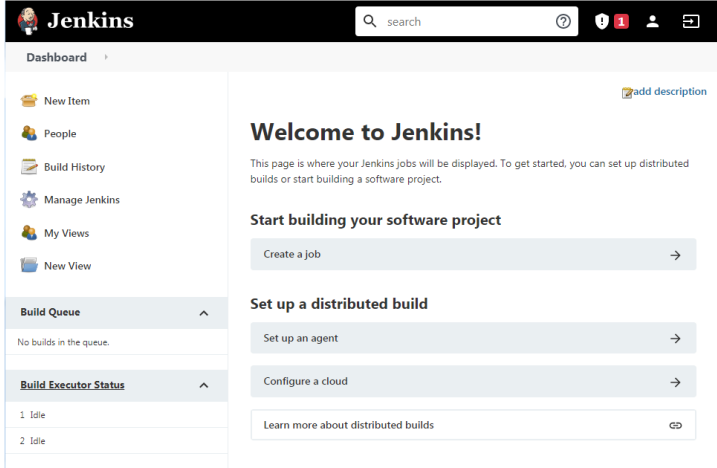
Welcome to Jenkins!

jenkins

.....

Sign in

☒ Keep me signed in

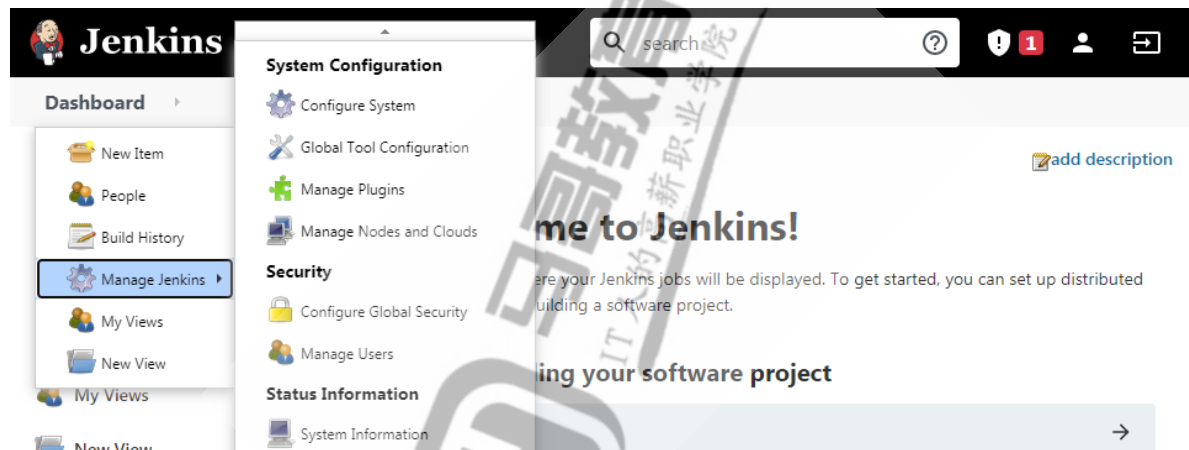


The Jenkins Dashboard shows a sidebar with links: New Item, People, Build History, Manage Jenkins, My Views, and New View. The main area has a 'Welcome to Jenkins!' message and a 'Start building your software project' button. Below this, there's a 'Set up a distributed build' section with buttons for 'Set up an agent', 'Configure a cloud', and a link to 'Learn more about distributed builds'.

到此为止，jenkins软件就安装完毕了。

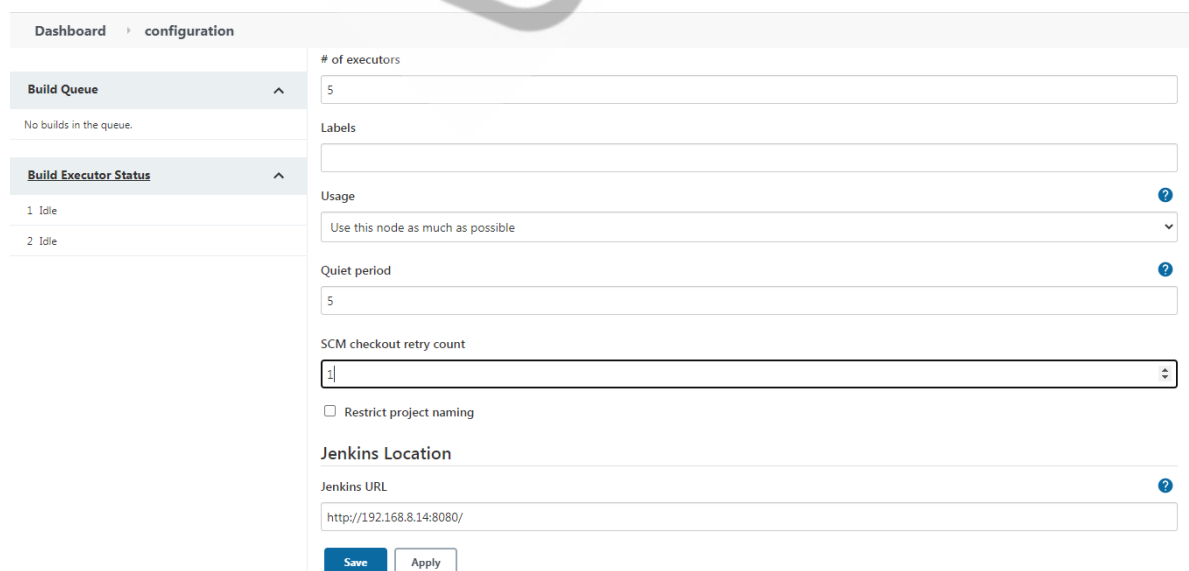
基本配置

点击左上角jenkins，选中下拉框中的"Manage Jenkins"，点击右侧菜单栏的"Configure System"



The image shows the Jenkins 'System Configuration' page. The left sidebar has 'Manage Jenkins' selected. The main content area shows various configuration options under 'System Configuration', 'Security', and 'Status Information'. The 'Configure System' option is highlighted.

进入到全局配置配置界面后，进行如下基本配置



The Jenkins Global Configuration page shows various settings. The 'Build Queue' section shows 'No builds in the queue.' The 'Build Executor Status' section shows two executors in 'Idle' state. The 'System Configuration' section includes fields for '# of executors' (5), 'Labels', 'Usage' (Use this node as much as possible), 'Quiet period' (5), 'SCM checkout retry count' (1), and 'Restrict project naming' (unchecked). The 'Jenkins Location' section includes the 'Jenkins URL' (http://192.168.8.14:8080/). The 'Save' and 'Apply' buttons are at the bottom.

of executors

执行者数量尽量设置为5

Usage - 使用默认值

Master节点选择"Use this node as much as possible"

如果是node节点的话, 只能选择"Only build jobs with label expressions matching this node"

Quiet period - 使用默认值

避免点击触发后, 我们反悔, 设置任务执行前等待时间, 设置为5秒,

SCM checkout retry count

scm重试次数, 我们不重试或者只尝试1次

Jenkins URL - 使用默认值

jenkins的主页面和管理员邮件通知账号

插件管理

对于jenkins的插件安装方式来说, 主要存在以下两种方式:

在线安装 - 从jenkins的官方网站上下载

离线安装 - 从官网<http://updates.jenkins-ci.org>下载到本地, 然后加载到本地环境

在线安装

1 默认情况下, web界面配置的本地插件源功能无法使用, 我们需要手工修改本地文件才会生效

```
cd /root/.jenkins/updates
```

```
sed -i 's#updates.jenkins.io/download#mirrors.tuna.tsinghua.edu.cn/jenkins#g' default.json
```

2 通过插件管理界面安装相关插件

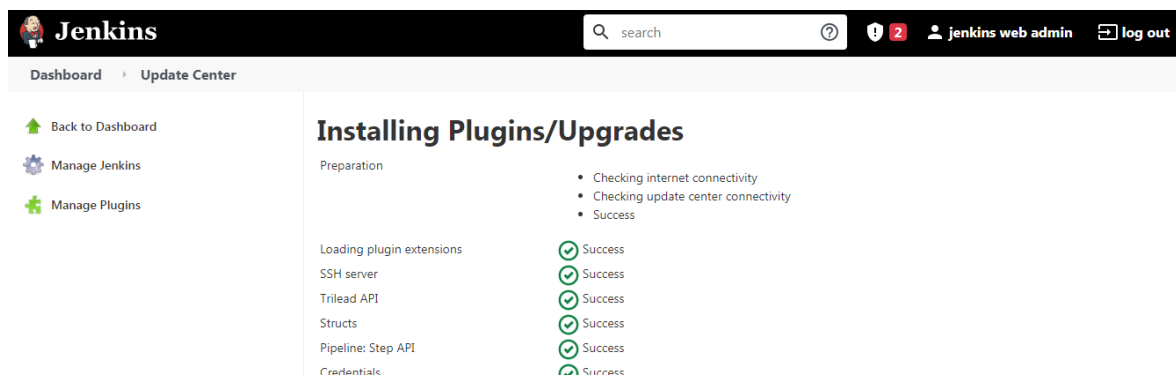
点击左上角jenkins, 选中下拉框中的"Manage Jenkins", 点击右侧菜单栏的"Manage Plugins", 进入插件管理界面

| Install | Name | Version | Released |
|--------------------------|--|---------|------------------|
| <input type="checkbox"/> | Mailer Build Tools email This plugin allows you to configure email notifications for build results | 1.34 | 5 mo 1 day ago |
| <input type="checkbox"/> | Script Security api-plugin Security Allows Jenkins administrators to control what in-process scripts can be run by less-privileged users. | 1.78 | 22 days ago |
| <input type="checkbox"/> | Credentials api-plugin This plugin allows you to store credentials in Jenkins. | 2.5 | 2 mo 20 days ago |
| <input type="checkbox"/> | SSH Credentials api-plugin | 1.19 | 2 mo 20 days ago |

当我们选择好想要的插件后, 比如, Git、Git client、Git Parameter, 然后点击最下面的"Download now and install after restart"

注意:

如果希望是中文界面, 我们安装中文插件即可 Localization: Chinese (Simplified)



离线安装

直接将下载的 *.jpi、*.hpi 文件解压到 /root/.jenkins/plugins 目录下即可，然后重启jenkins 软件即可

```
root@python-auto:~/jenkins/plugins# ls
ace-editor
ace-editor.jpi
antisamy-markup-formatter
antisamy-markup-formatter.jpi
apache-httpcomponents-client-4-api
apache-httpcomponents-client-4-api.jpi
blueocean-commons
blueocean-commons.jpi
blueocean-config
blueocean-config.jpi
blueocean-core-js
blueocean-core-js.jpi
blueocean-dashbaord
blueocean-dashbaord.jpi
git-client.jpi
github
github-api
github-api.jpi
github-branch-source
github-branch-source.jpi
github.jpi
git.jpi
gitlab-api
gitlab-api.jpi
gitlab-hook
gitlab-hook.jpi
gitlab-plugin
pipeline-stage-step
pipeline-stage-step.jpi
pipeline-stage-tags-metadata
pipeline-stage-tags-metadata.jpi
plain-credentials
plain-credentials.jpi
plugin-util-api
plugin-util-api.jpi
popper2-api
popper2-api.jpi
popper-api
popper-api.jpi
pubsub-light
```

小结

其他配置

学习目标

这一节，我们从 认证配置、邮件配置、小结 三个方面来学习。

认证配置

简介

对于jenkins来说，他支持的认证配置方式很多，在不同的场景下，支持的方式也多种多样，只要我们安装好对应的插件、定制对应的配置即可。

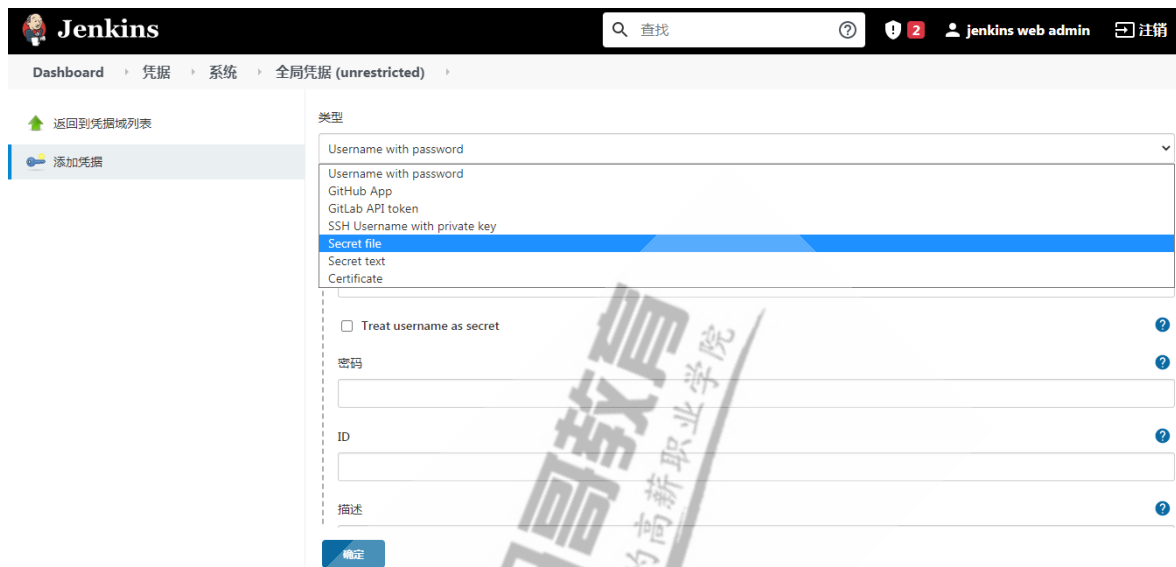
jenkins最近的认证配置主要有两个：

- 秘钥文件认证 - 基于秘钥文件进行认证
- 用户名密码认证 - 基于系统账号的登录密码进行认证

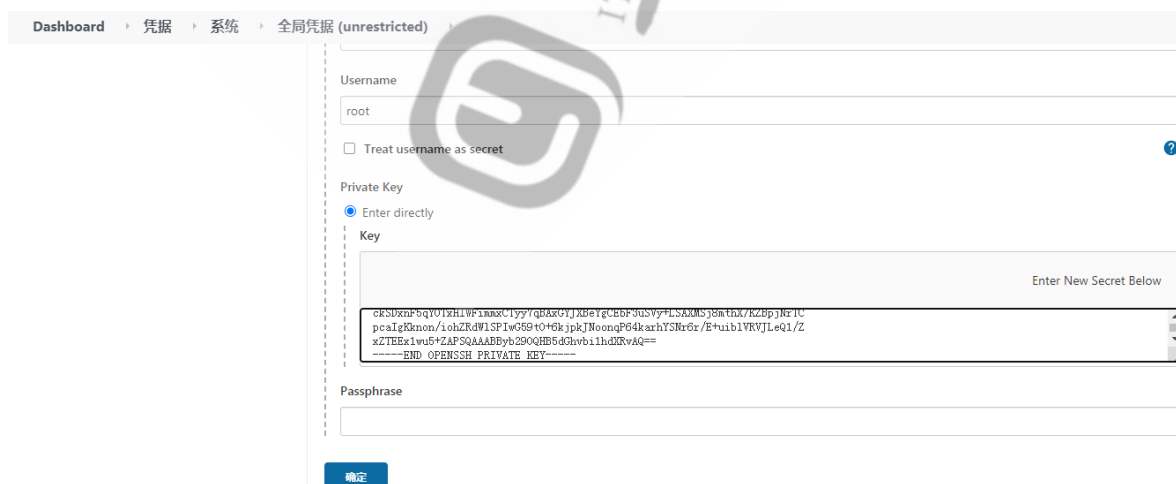
点击左上角jenkins，选中下拉框中的"Manage Jenkins"，点击右侧菜单栏的"Manage Credentials"，进入凭据管理界面，



点击"添加凭据"，进入到凭据管理界面



密钥文件认证：点击"添加凭据"后，选择认证方式为选择 "SSH Username with private key."



配置Username 为jenkins服务器的系统登录账号 root

将jenkins服务器的私钥内容(/root/.ssh/id_rsa)输入到密钥框

点击确定

用户名密码认证：点击"添加凭据"后，选择认证方式为选择 "Username with Password."

Dashboard > 凭据 > 系统 > 全局凭据 (unrestricted) >

返回到凭据域列表

添加凭据

类型

Username with password

范围

全局 (Jenkins, nodes, items, all child items, etc)

用户名

root

☐ Treat username as secret

密码

.....

ID

描述

确定

配置Username 为jenkins服务器的系统登录账号 root
密码Password 为jenkins服务器的系统登录账号 root的登录密码
点击确定

两个认证效果

Jenkins

Dashboard > 凭据 > 系统 > 全局凭据 (unrestricted) >

返回到凭据域列表

添加凭据

全局凭据 (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

| ID | 名称 | 类型 | 描述 |
|---------------------------------------|------------|-------------------------------|----|
| fa45ef57-e914-4266-8907-6b4181eae2b3 | root | SSH Username with private key | |
| 7457c825-50fd-4708-9fa8-ffb9a9af8b2e4 | root/***** | Username with password | |

邮件配置

简介

对于邮件的配置，主要依赖于两个方面：

- jenkins安装好对应的邮件扩展插件 E-mail Notification
- 准备好 126邮箱服务器配置
登录密码: wshs1117@126.com
授权账号: LTZFDDIVDRNYOOAL
smtp服务器: smtp.126.com
smtp服务器端口: 465

点击左上角jenkins，选中下拉框中的"Manage Jenkins"，点击右侧菜单栏的"Configure System",定制管理员邮箱

Jenkins URL

http://192.168.8.14:8080/

系统管理员邮件地址

wshs1117@126.com

Jenkins URL

系统管理员邮件地址

http://192.168.8.14:8080/

wshs1117@126.com

找到 "E-mail Notification" 部分输入以下内容：SMTP服务器和用户默认邮件后缀

邮件通知

SMTP服务器

smtp.126.com

用户默认邮件后缀

@126.com

SMTP 地址: smtp.126.com

SMTP 后缀: @126.com

点击高级，进行SMTP的基本认证配置

☒ 使用SMTP认证

用户名

wshs1117@126.com

密码

.....

☒ 使用SSL协议

☒ Use TLS

SMTP端口

465

Reply-To Address

wshs1117@126.com

字符集

UTF-8

☐ 通过发送测试邮件测试配置

保存

应用

用户名: wshs1117@126.com

密码: LTZFDDIVDRNYOOAL

smtp服务器端口: 465

Reply-To Address(回复地址): wshs1117@126.com

字符集: 使用默认的UTF-8 -- 支持中文

勾选 通过发送测试邮件测试配置

☒ 通过发送测试邮件测试配置

Test e-mail recipient

wshs1117@126.com

Email was successfully sent

Test configuration

保存

应用

点击"Test e-mail recipient",输入邮件发送测试地址,效果如下

Test e-mail recipient: wshs1117@126.com

结果显示:

Email was successfully sent,说明邮箱配置是成功的。

小结

任务构建

学习目标

这一节，我们从 简单实践、任务解析、小结 三个方面来学习。

简单实践

准备工作 - gitlab仓库 - 配置好jenkins服务器与gitlab服务器的sshkey认证

User Settings > SSH Keys > root@python-auto

SSH Key

Title: **root@python-auto**

Created on: **Aug 22, 2021 8:03pm**

Expires: **Never**

Last used on: **Never**

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQDrwZKU3jiJhU9g9ziNqHX60JBGQsULimX2gbVRq/yaqX+2jy

Fingerprints

MD5: 8e:71:69:bc:d4:8f:ec:cc:da:07:f0:47:9d:b3:5f:17

SHA256: 4/tB1tGWhTStcAkroN00Mhw4pM4xfmzOppo4CkJ/w

Delete

- 1 本地创建ssh密钥
- 2 将公钥信息注册到gitlab服务上

注意：
jenkins主机必须具备git的命令，否则失效

回到jenkins首页

Jenkins

查找

jenkins web admin

注销

Dashboard

New Item

People

Build History

Manage Jenkins

My Views

打开 Blue Ocean

更多...

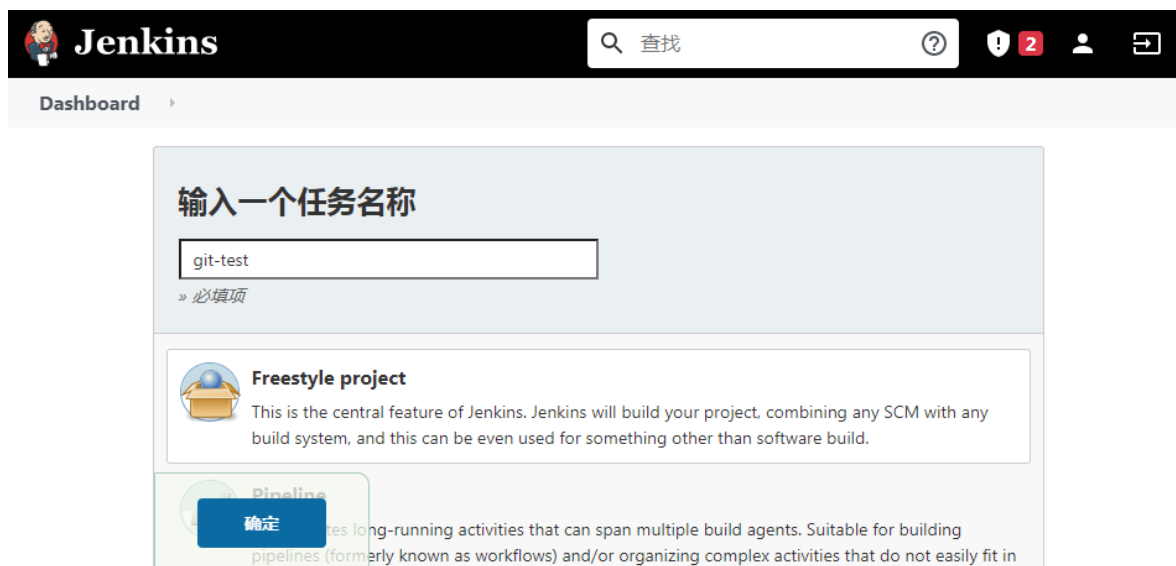
欢迎来到 Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

点击 "Create a job", 选中"Freestyle project",输入任务的名称"git-test",然后点击"ok", 进入到任务构建界面



点击"Source Code Management"栏，因为我们的源代码在git仓库里面，接着点击"Git",接下来开始配置git



输入我们的git仓库地址
选择认证账号"git@192.168.8.14"
分支指定处，直接使用master即可，如果我们的分支不是master，就要指定对应的分支名称

点击"Apply"后再点击"Save",自动回到任务界面

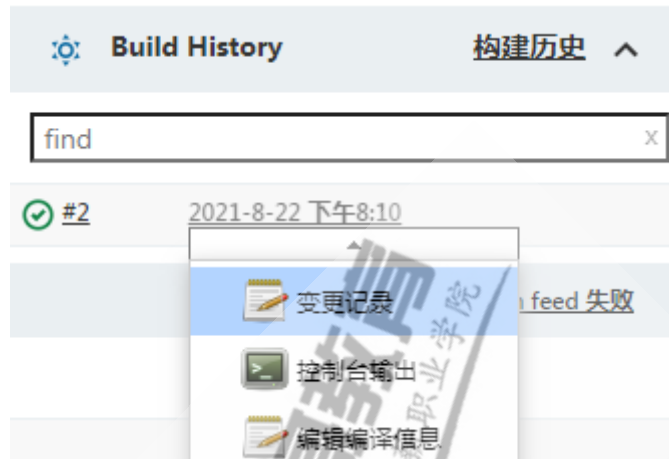


点击左侧栏的"Build Now"



结果显示：在左侧栏的最下面的"Build History"下面多了一个构建记录 "#1"

将鼠标放在构建历史记录上，会出现一个黑色小三角，点击该三角后，会弹出一个菜单



我们选中"Console Output"



结果显示：我们可以在该界面看到，该任务的详细执行效果，里面记录了任务中所有命令的执行流程及该任务的状态记录

任务解析

通过刚才的构建历史记录，我们知道了一个jenkins的job任务执行，都会在jenkins的服务器里面的工作路径中创建一个与job名称一致的目录，格式是 "\$JENKINS_HOME/workspace/JOB_NAME"

Building in workspace /root/.jenkins/workspace/git-test

检查目录

```
root@python-auto:~/jenkins/workspace/git-test# ls
README.txt
root@python-auto:~/jenkins/workspace/git-test# git config --list
user.name=Administrator
user.email=admin@example.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=ssh://git@192.168.8.15:2222/root/mypro-web.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
root@python-auto:~/jenkins/workspace/git-test#
```

jenkins的任务执行目录是 /root/.jenkins/workspace，而在jenkins目录下有个jobs的目录，在该目录里面有我们配置的所有任务记录

```
root@python-auto:~/jenkins/workspace/git-test# tree ~/.jenkins/jobs/
/root/.jenkins/jobs/
├── git-test
│   ├── builds
│   │   ├── 2
│   │   │   ├── build.xml
│   │   │   ├── changelog.xml
│   │   │   └── log
│   │   ├── legacyIds
│   │   └── permalinks
│   ├── config.xml
│   └── nextBuildNumber
3 directories, 7 files
```

jenkins所有的任务都是在 \$JENKINS_HOME/jobs/ 目录下以任务名称创建专用目录
每个任务都有自己的配置文件，任务在构建后，都会在builds目录下创建以历史记录为名的目录，在目录中会有配套的日志记录、配置记录、更改记录。
从这个方面我们可以了解到，配置记录的追踪功能在jenkins内部是一个非常重要的功能。

配置文件解析

构建配置文件

```
]# cat ~/.jenkins/jobs/git-test/config.xml
...
<configVersion>2</configVersion>
<userRemoteConfigs>
  <hudson.plugins.git.UserRemoteConfig>
    <url>ssh://git@192.168.8.15:2222/root/mypro-web.git</url>
    <credentialsId>fa45ef57-e914-4266-8907-6b4181eae2b3</credentialsId>
  </hudson.plugins.git.UserRemoteConfig>
</userRemoteConfigs>
<branches>
  <hudson.plugins.git.BranchSpec>
    <name>*/main</name>
  </hudson.plugins.git.BranchSpec>
</branches>
...

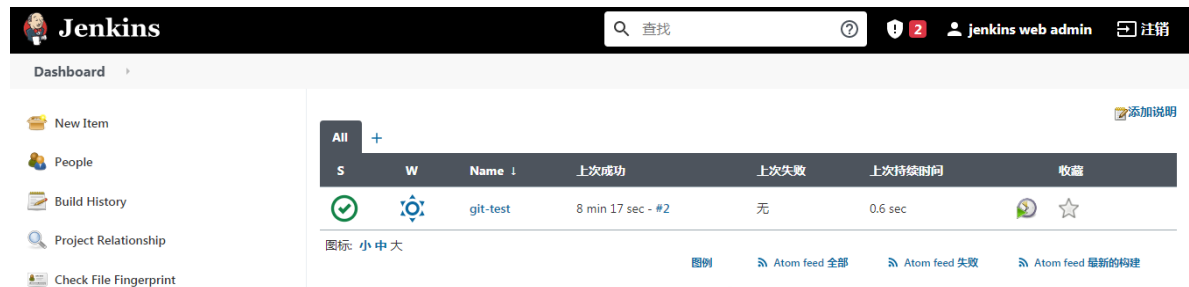
]# cat ~/.jenkins/jobs/git-test/builds/1/build.xml
...
```

```
<remoteUrls>
  <string>ssh://git@192.168.8.15:2222/root/mypro-web.git</string>
</remoteUrls>

...
```

构建解析

点击jenkins图标，回到任务列表



点击任务列表中的"git-test"下拉框的"configure", 进入任务编辑界面,效果如下



点击配置，回到配置界面



一个标准的构建任务配置，主要有六部分组成，这六部分的作用分别是：

| 属性 | 解析 |
|------------------------------|---|
| 全局配置(General) | 项目基本配置，项目名字,描述,参数,禁用项目,并发构建,限制构建默认node等 |
| 源代码管理(Source code managemet) | 源代码仓库配置信息,支持Git,Subversion等 |
| 触发构建(Build Triggers) | 构建触发方式，包含周期性构建,poll scm,远程脚本触发构建,其他项目构建结束后触发等 |
| 构建环境(Build Environment) | 构建环境相关设置，构建前删除工作空间,输出信息添加时间戳,设置构建名称,插入环境变量等 |
| 执行构建(Build) | 设置项目构建任务的执行方式、种类等，具体的执行信息 |
| 构建后行为(Post-build Actions) | Artifact归档,邮件通知,发布单元测试报告,触发下游项目等 |

注意：

由于jenkins是一个功能高度插件化的管理平台，所以，这里六部分的具体内容会根据插件的变化而不同 红色字体的部分，是工作中使用频率比较高的部分。

构建状态



>80%



61% to 80%



41% to 60%



21% to 40%



< 21%

Jenkins会基于一些后处理器任务为构建发布一个稳健指数(从0-100)，这些任务一般以插件的方式实现。而评分部分主要是基于单元测试(JUnit)、覆盖率(Cobertura)和静态代码分析(FindBugs)等对我们的项目代码进行综合评分，分数越高，表明构建后的项目越稳定。

晴雨表主要是针对一个任务的整体执行成功比例来算的。80%成功表示太阳。

小结

构建实践

简单构建

学习目标

这一节，我们从 软件安装、基本配置、小结 三个方面来学习。

软件安装

场景需求

我们接下来，基于自由风格来创建一个完整的任务，将项目代码从git仓库中获取到本地，然后借助于Docker镜像，将该项目运行起来，对外暴露的端口是666，任务构建成功后，浏览器测试效果。 为了保证jenkins的工作空间有足够的空闲，我们基于任务的保留时长及记录保留的数量来进行限制。

案例分析

构建任务种类： 自由风格

构建属性

| 属性 | 说明 |
|-------|------------------------|
| 全局配置 | 定制构建历史保留规则 |
| 源代码仓库 | 采用git配置 |
| 执行构建 | 采用shell命令的方式执行docker构建 |

准备代码仓库

在 `gitlab` 中创建一个项目 `tomcat_pro`

提交项目代码到git仓库

```
获取仓库代码
cd /tmp
git clone ssh://git@192.168.8.15:2222/root/tomcat_pro.git
cd tomcat_pro

准备项目源代码
cd /data/backup
tar xf tomcat-web.tar.gz -C /tmp/tomcat_pro/
cd /tmp/tomcat_pro/
git add . && git commit -m "tomcat-web" && git push origin master

网页代码
<%@ page language="java" import="java.util.*" contentType="text/html;
charset=UTF-8" %>
<html>
  <head>
    <title>电商网站</title>
  </head>
  <body>
    <h1 style="color:red ; font-size:30px ; display:inline;"> 开业大酬宾:
  </h1>
    <%
      String str = "买 1 送 2";
      out.print(str);
    %>
  </body>
</html>
```

Administrator > tomcat_pro > Repository

master tomcat_pro / +

History Find file Web IDE 下载 Clone

tomcat-web
Your Name authored 1 minute ago

c644a3f8

| Name | Last commit | Last update |
|------------|-------------|--------------|
| tomcat-web | tomcat-web | 1 minute ago |

jenkins准备docker环境

软件源配置

```
curl -fsSL https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/ubuntu/gpg |  
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg  
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-  
keyring.gpg] https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/ubuntu  
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >  
/dev/null
```

安装软件

```
apt update
```

```
apt-get -y install docker-ce docker-ce-cli containerd.io
```

加速器配置

```
echo '{"registry-mirrors": ["http://f1361db2.m.daocloud.io"], "insecure-  
registries": []}' > /etc/docker/daemon.json  
systemctl restart docker
```

导入基准镜像

```
docker load < ubuntu-tomcat.tar.gz
```

基本配置

创建配置

点击首页的左侧栏"New item", 选择"Freestyle project", 项目名称为"tomcat-web", 最后点击"OK"



全局配置

在全局配置项部分, 勾选"Discard old builds", 定制构建记录的保存时间为3天, 构建任务的保存数量为5个

Dashboard > tomcat-web >

General 源码管理 构建触发器 构建环境 构建 构建后操作

☒ Discard old builds ?

策略

Log Rotation v

保持构建的天数

3

如果非空，构建记录将保存此天数

保持构建的最大个数

5

如果非空，最多此数目的构建记录将被保存

保存 应用 高级...

仓库配置

在源代码管理部分，勾选git仓库，配置仓库地址和认证账号

Dashboard > tomcat-web >

General 源码管理 构建触发器 构建环境 构建 构建后操作

源码管理

☐ 无

☒ Git ?

Repositories ?

Repository URL ?

ssh://git@192.168.8.15:2222/root/tomcat_pro.git

Credentials ?

root/***** 添加

保存 应用 高级...

Add Repository

执行构建

在构建部分，我们选择最常用的"Execute shell"模式，这时为了知道如下的命令在哪里执行，所以这里先测试一下

我在哪里执行下面的命令

pwd

当前目录下有哪些东西

ls

检查效果

依次点击"Apply"和"Save"后，回到任务界面，然后点击"Build now"

Dashboard > tomcat-web > #2

返回到工程

状态集

变更记录

控制台输出

文本方式查看

编辑编译信息

删除构建 '#2'

Git Build Data

打开 Blue Ocean

上一次构建

控制台输出

Started by user `jenkins web admin`
Running as SYSTEM
Building in workspace `/root/.jenkins/workspace/tomcat-web`
The recommended git tool is: NONE
using credential `d883581a-8c76-4bf7-b268-4681d0f47d49`
> git rev-parse --resolve-git-dir `/root/.jenkins/workspace/tomcat-web/.git` # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url `ssh://git@192.168.8.15:2222/root/tomcat_pro.git` # timeout=10
Fetching upstream changes from `ssh://git@192.168.8.15:2222/root/tomcat_pro.git`
> git --version # timeout=10
> git --version # 'git version 2.25.1'
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- `ssh://git@192.168.8.15:2222/root/tomcat_pro.git`
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision `c644a3f8606ff123ec4ee7c65531f49d7406bb7b` (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f `c644a3f8606ff123ec4ee7c65531f49d7406bb7b` # timeout=10
Commit message: "tomcat-web"
> git rev-list --no-walk `c644a3f8606ff123ec4ee7c65531f49d7406bb7b` # timeout=10
[tomcat-web] \$ `/bin/sh -xe /tmp/jenkins7622845936676971961.sh`
+ pwd
/root/.jenkins/workspace/tomcat-web
+ ls
tomcat-web
Finished: SUCCESS

改造构建

在构建部分的"Execute shell"模式里，输入构建docker镜像的基本命令

```
echo "开始 tomcat_web 任务"
# 准备基本目录
[-d /data/docker/image ] || mkdir -p /data/docker/image
tar -C tomcat-web -zcf tomcat-web/ROOT.tar.gz ROOT --remove-files
mv -i tomcat-web /data/docker/image/

# 构建镜像
docker build -t tomcat-web:v0.1 /data/docker/image/tomcat-web

# 重启项目
num=$(docker ps | grep tomcat-web | wc -l)
[ $num -eq 1 ] && docker rm -f tomcat-web
docker run -d --name tomcat-web -p 666:8080 tomcat-web:v0.1

# 清理旧代码
rm -rf /data/docker/image/tomcat-web

echo "结束 tomcat_web 任务"
```

检查效果

依次点击"Apply"和"Save"后，回到任务界面，然后点击"Build now"，由于我们本地有镜像的构建缓存，所以我们这次构建会很快

我们可以点击构建记录，查看构建过程，点击"console OutPut"，查看构建记录，浏览器访问效果

小结

触发构建

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

功能简介

触发构建的含义主要有三个：触发当前任务执行，触发其他任务执行、Hook触发

| 触发当前任务: 类型 | 说明 |
|--|--------------|
| Trigger builds remotely (e.g., from scripts) | 远程触发构建 |
| Build periodically | 定期构建 |
| Poll SCM | 轮训检查源代码变动后构建 |

| 触发其他任务执行: 类型 | 说明 |
|---|-------------------|
| Build after other projects are built (Build Triggers) | 其他任务执行完毕后，再执行当前任务 |
| Build other projects (Post-build Actions) | 当前任务执行完毕后，再执行其他任务 |

| Hook触发: 类型 | 说明 |
|-------------------------|---------------------|
| git,Gitlab,GitHub hooks | 代码仓库发生变动时候，自动执行当前任务 |

注意：
这些触发的功能，主要是借助于不同的功能插件来实现的，我们接下来演示一个构建后触发任务的效果，其他的触发大家可以在课余进行学习，我们会在后序gitlab集成中介绍hook的内容。

插件介绍

对于构建后的任务触发功能，依赖于一个插件：Parameterized Trigger，找到该插件后，安装它

UpdatesAvailableInstalledAdvanced

| Install ↑ | Name | Version | Released |
|--------------------------|--|---------|------------------|
| <input type="checkbox"/> | <div>Parameterized Trigger</div> <div>Build ToolsBuild Triggers</div> <div>The Jenkins Plugins Parent POM Project</div> <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div> | 2.41 | 2 mo 18 days ago |

简单实践

任务需求

我们目前有两个任务，`git-test`和`tomcat_web`，我们的需求就是，当`git-test`任务执行完毕后，自动触发`tomcat_web`任务。

- 触发构建1

A执行成功后，触发B任务执行

`git-test` 定制

构建后动作部分选择"Build other projects",
在"Projects to build" 后选择"`tomcat_web`"



点击"Apply"和"Save",之后的效果如下



结果显示：在`git-test`任务下面多了一条Downstream Projects条目，而且内容是`tomcat-web`

点击 `git-test` 任务，执行构建后，查看job列表

| S | W | Name ↓ | 上次成功 | 上次失败 | 上次持续时间 | 收藏 |
|---|---|------------|-------------|-------------------|----------|----|
| | | git-test | 23 sec - #3 | 4 hr 8 min - #1 | 0.99 sec | |
| | | tomcat-web | 16 sec - #6 | 6 min 55 sec - #4 | 2.3 sec | |

图标: 小 中 大

图例 [Atom feed 全部](#) [Atom feed 失败](#) [Atom feed 最新的构建](#)

我们可以查看两个任务的构建历史记录的相关地方：

🟢控制台输出

```
Started by user jenkins web admin
Running as SYSTEM
Building in workspace /root/.jenkins/workspace/git-test
The recommended git tool is: NONE
using credential d7616c9e-3897-4e9a-81fd-335861ba4de6
> git rev-parse --resolve-git-dir /root/.jenkins/workspace/git-test/.git # time
Fetching changes from the remote Git repository
> git config remote.origin.url ssh://git@192.168.8.15:2222/root/mypro-web.git #
Fetching upstream changes from ssh://git@192.168.8.15:2222/root/mypro-web.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
using GIT_SSH to set credentials
> git fetch --tags --force --progress -- ssh://git@192.168.8.15:2222/root/mypro-
Seen branch in repository origin/main
Seen 1 remote branch
> git show-ref --tags -d # timeout=10
Checking out Revision 288326b703a5e532a93def69fe9f9948323882f1 (origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 288326b703a5e532a93def69fe9f9948323882f1 # timeout=10
Commit message: "Update README.txt"
> git rev-list --no-walk 288326b703a5e532a93def69fe9f9948323882f1 # timeout=10
Triggering a new build of tomcat-web
Finished: SUCCESS
```

- 触发构建2

🟢控制台输出

```
Started by upstream project "git-test" build number 3
originally caused by:
Started by user jenkins web admin
Running as SYSTEM
Building in workspace /root/.jenkins/workspace/tomcat-web
The recommended git tool is: NONE
using credential d883581a-8c76-4bf7-b268-4681d0f47d49
> git rev-parse --resolve-git-dir /root/.jenkins/workspace/tomcat-web/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url ssh://git@192.168.8.15:2222/root/tomcat_pro.git # timeout=10
Fetching upstream changes from ssh://git@192.168.8.15:2222/root/tomcat_pro.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- ssh://git@192.168.8.15:2222/root/tomcat_pro.git +re
> git rev-parse refs/remotes/origin/master [commit] # timeout=10
Checking out Revision 8300a06867a990fc63d4c831d017331d7b8572b7 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 8300a06867a990fc63d4c831d017331d7b8572b7 # timeout=10
Commit message: "aaa"
> git rev-list --no-walk 8300a06867a990fc63d4c831d017331d7b8572b7 # timeout=10
[tomcat-web] $ /bin/sh -xe /tmp/jenkins7111754489013022369.sh
+ echo 开始 tomcat-web 任务
+ echo 开始 tomcat-web 任务
```

B执行依赖于A任务执行成功

- 1 取消刚才的 `git-test` 任务后面的触发后动作
- 2 `tomcat-web` 定制
构建触发器部分选择"Build after other projects are built",
在"关注的项目" 后选择"git-test"

Dashboard > tomcat-web >

General 源码管理 构建触发器 构建环境 构建 构建后操作

构建触发器

☐ 触发远程构建 (例如,使用脚本)

☒ Build after other projects are built

关注的项目

git-test.

☒ 只有构建稳定时触发

☐ 即使构建不稳定时也会触发

☐ 即使构建失败时也会触发

☐ Build periodically

☐ Build when a change is pushed to GitLab. GitLab webhook URL: http://192.168.8.14:8080/project/tomcat-web

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

点击"Apply"和"Save",之后的效果如下

Dashboard

tomcat-web

返回面板

状态

修改记录

工作空间

Build Now

配置

删除 Project

收藏夹

打开 Blue Ocean

重命名

Build History

构建历史

Project tomcat-web

工作区

最新修改记录

上级项目

git-test

相关链接

Last build(#6),4 min 38 sec之前

Last stable build(#6),4 min 38 sec之前

Last successful build(#6),4 min 38 sec之前

Last failed build(#4),11 min之前

Last unsuccessful build(#4),11 min之前

Last completed build(#6),4 min 38 sec之前

Project git-test

工作区

最新修改记录

下级项目

tomcat-web

相关链接

Last build(#3),5 min 1 sec之前

Last stable build(#3),5 min 1 sec之前

Last successful build(#3),5 min 1 sec之前

Last failed build(#1),4 hr 12 min之前

Last unsuccessful build(#1),4 hr 12 min之前

Last completed build(#3),5 min 1 sec之前

结果显示：在tomcat-web任务下面多了一条Upstream Projects条目，而且内容是git-test

点击 git-test 任务，执行构建后，查看job列表

| All | | | | | | | |
|-----|---|------------|--------------|------------------|----------|----|---|
| S | W | Name ↓ | 上次成功 | 上次失败 | 上次持续时间 | 收藏 | |
| ✓ | ☁ | git-test | 17 sec - #4 | 4 hr 16 min - #1 | 0.54 sec | 🕒 | ☆ |
| ✓ | ⚙ | tomcat-web | 7.3 sec - #9 | 无 | 2.3 sec | 🕒 | ☆ |

图标: 小中大

图例

Atom feed 全部

Atom feed 失败

Atom feed 最新的构建

注意：

这里是触发构建，所以如果我们直接点击 tomcat-web 的任务构建，这看不到触发构建的效果，所以我们要在git-test里面点击 build now。

小结

参数构建

学习目标

这一节，我们从 软件安装、基本配置、小结 三个方面来学习。

软件安装

场景需求

当我们的业务场景中，不同项目的代码、甚至同一个项目的不同版本的代码都存放在不同的分支上，在这种情况下，我们直接获取所有项目代码的方式就不合适了，所以我们就需要基于功能分支的名称来定制地获取代码，从而能快速高效的完成docker镜像的构建工作。

对于这种基于参数的构建方式在jenkins平台中有非常多的插件可以实现不同的参数构建功能，我们常用的就是git 参数构建场景，这就需要一个插件

获取插件

到git的插件管理平台，找到 git parameter plugin插件，然后安装它。

注意：该插件一般会与Git插件同时安装。

| Updates | Available | Installed | Advanced | |
|-------------------------------------|---|-----------|------------------------------|-----------|
| Enabled | Name ↓ | Version | Previously installed version | Uninstall |
| Git Parameter Plug-In | | | | |
| <input checked="" type="checkbox"/> | Adds ability to choose branches, tags or revisions from git repositories configured in project. | 0.9.13 | | Uninstall |

在git参数定制构建的过程中，往往会涉及获取git版本信息的需求，这就需要用到插件Version Number插件

| Updates | Available | Installed | Advanced | |
|-------------------------------------|---|-----------|------------------------------|----------------------------|
| Enabled | Name ↓ | Version | Previously installed version | Uninstall |
| <input checked="" type="checkbox"/> | <p>Pipeline: API</p> <p>Plugin that defines Pipeline API.</p> | 2.46 | | <button>Uninstall</button> |
| <input checked="" type="checkbox"/> | <p>Version Number Plug-In</p> <p>This plugin allows much richer version numbers to be created and used.</p> <p>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</p> | 1.9 | | <button>Uninstall</button> |

基本配置

任务需求

在git仓库定制多个分支(dev,test,release,prod),然后分别在不同的分支上定制tomcat-web首页，在构建的时候，选择分支进行构建，在构建历史记录中将其分支信息显示出来

定制仓库

```
cd /tmp/tomcat_pro
git checkout -b dev
sed -i "s/买 1/买 dev/" tomcat-web/ROOT/index.jsp
git add . && git commit -m "dev" && git push origin dev:dev
git checkout master && git checkout -b test
sed -i "s/买 1/买 test/" tomcat-web/ROOT/index.jsp
git add . && git commit -m "test" && git push origin test:test
git checkout master && git checkout -b release
sed -i "s/买 1/买 release/" tomcat-web/ROOT/index.jsp
git add . && git commit -m "release" && git push origin release:release
git checkout master && git checkout -b prod
sed -i "s/买 1/买 prod/" tomcat-web/ROOT/index.jsp
git add . && git commit -m "prod" && git push origin prod:prod
```

定制任务

进入到tomcat-web任务，在全局部分定制git参数构建，选择"Git parameter",按照如下方式进行定制

General

源码管理

构建触发器

构建环境

构建

构建后操作

☒ This project is parameterized

Git Parameter

名称 ?

branch

描述 ?

选择要发布代码所在的分支

[Plain text] 预览

参数类型 ?

分支

默认值 ?

master

高级...

注意：Name 定制的名称，在当前任务中就是一个变量，我们可以在其他部分以\${NAME}的方式来调用该变量的值。

我们在创建一个"Choice parameter", 在Choices部分输入一系列的环境值，以便于构建任务的命名

General

源码管理

构建触发器

构建环境

构建

构建后操作

Choice Parameter

名称 ?

environment

选项 ?

Dev
Test
Release
Prod

描述 ?

构建任务命名时候，显示所处的分支环境信息

[Plain text] 预览

添加参数

在源代码仓库部分，修改获取分支的方式为"\${branch}"

General

源码管理

构建触发器

构建环境

构建

构建后操作

ssh://git@192.168.8.15:2222/root/tomcat_pro.git

Credentials

root/*****

添加

Name

origin

Refspec

Add Repository

Branches to build

指定分支 (为空时代表any)

\${branch}

点击仓库的高级部分

然后再Name右侧写上仓库的名字origin(git remote)，便于后面创建提交tag标签用

默认的值是只获取master分支

上面我们采用branch的参数方式自动获取所有分支，这里使用\${branch}直接调用即可。

定制历史记录名称

在构建变量定制部分选择"Create a formatted version number"，在定制格式里面输入如下内容：
\${branch}_\${environment}

General

源码管理

构建触发器

构建环境

构建

构建后操作

构建环境

☐ Use secret text(s) or file(s)

☒ Create a formatted version number

Environment Variable Name

BUILD_VERSION

Version Number Format String

\${branch}_\${environment}

Prefix Variable

Skip Builds worse than

SUCCESS

Don't increment builds today / this week / this month / this year / all time after a build-run with a result worse than the selected one.

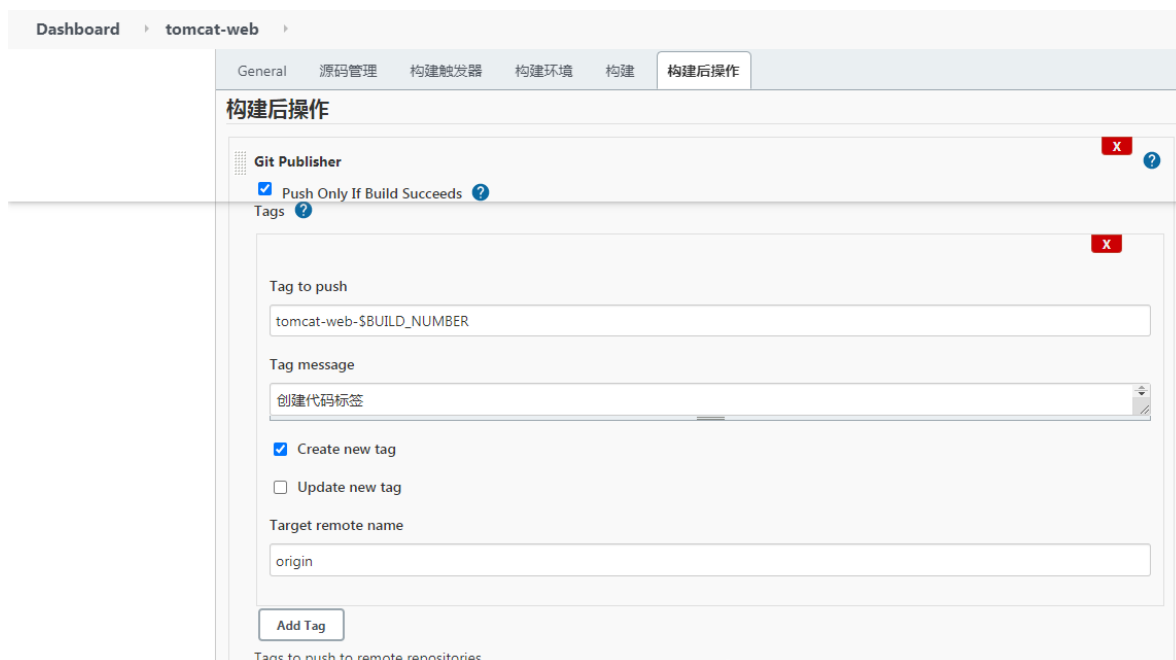
☒ Build Display Name Use the formatted version number for build display name.

注意：

这里的构建变量是我们定制变量，如果同名的话，可以自动覆盖同名变量

一定要勾选 "Build Display Name"，否则的话，我们定制的变量不会自动生效。

定制构建后动作



在 "Post-build Actions" 部分, 点击"ADD post-build action"
选择"Git publisher"
勾选"Push Only if Build Succeeds"
点击"Add Tag"
Tag to push 右侧添加"tomcat-web-\$BUILD_NUMBER"
Tag message 右侧添加"创建代码标签"
勾选"Create new tag"
Target remote name 右侧添加"origin", 名字与仓库名称一致

注意:

一定要保证我们的仓库是具备 `user.name` 和 `user.email` 的相关信息,
否则会因为git认证的原因导致提交失败

我们点击"Apply"和"Save"即可, 回到任务首页查看效果。



结果显示: 我们可以看到左侧边栏有个"Build with Parameters"

点击参数构建，查看效果

The screenshot shows the Jenkins web interface. At the top is the Jenkins logo and a search bar. Below is a navigation bar with 'Dashboard' and 'tomcat-web'. On the left is a sidebar with icons for '返回面板', '状态', '修改记录', '工作空间', 'Build with Parameters', '配置', '删除 Project', '收藏夹', and '打开 Blue Ocean'. The main area is titled 'Project tomcat-web' and contains the following configuration options:

- branch:** A dropdown menu with options: origin/master, origin/test, origin/prod, origin/dev, origin/release.
- environment:** A dropdown menu with the option 'Dev'.
- Text:** 需要如下参数用于构建项目: (Need the following parameters for building the project:)
- Text:** 选择要发布代码所在的分支 (Select the branch where the code to be released is located)
- Text:** 构建任务命名时候，显示所处的分支环境信息 (When naming the build task, display the branch and environment information)
- Button:** 开始构建 (Start Build)

选择分支界面里面的"origin/test",环境部分选择对应的Test，然后点击"Build",构建历史效果如下

The screenshot shows the 'Build History' page in Jenkins. It has a search bar with the text 'find'. Below the search bar, there is a list of builds. The first build is highlighted and shows a green checkmark icon, the text 'origin/test Test', and the timestamp '2021-8-26 下午4:50'.

结果显示：构建的历史记录名称已经跟改为了 "分支名称_环境"的格式了

查看构建历史记录

控制台输出

```
Started by user jenkins web admin
Running as SYSTEM
Building in workspace /root/.jenkins/workspace/tomcat-web
The recommended git tool is: NONE
using credential d883581a-8c76-4bf7-b268-4681d0f47d49
> git rev-parse --resolve-git-dir /root/.jenkins/workspace/tomcat-web/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url ssh://git@192.168.8.15:2222/root/tomcat_pro.git # timeout=10
Fetching upstream changes from ssh://git@192.168.8.15:2222/root/tomcat_pro.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- ssh://git@192.168.8.15:2222/root/tomcat_pro.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/test^{commit} # timeout=10
Checking out Revision 18f5a148d81e0d3f4acdc4db534a9ffc27e11e71 (refs/remotes/origin/test)
> git config core.sparsecheckout # timeout=10
> git checkout -f 18f5a148d81e0d3f4acdc4db534a9ffc27e11e71 # timeout=10
Commit message: "test"
First time build. Skipping changelog.
+ echo 结束 tomcat_web 任务
结束 tomcat_web 任务
The recommended git tool is: NONE
using credential d883581a-8c76-4bf7-b268-4681d0f47d49
> git tag -l tomcat-web-11 # timeout=10
> git tag -a -f -m 创建代码标签 tomcat-web-11 # timeout=10
Pushing tag tomcat-web-11 to repo origin
> git --version # timeout=10
> git --version # 'git version 2.25.1'
using GIT_ASKPASS to set credentials
> git push ssh://git@192.168.8.15:2222/root/tomcat_pro.git tomcat-web-11 # timeout=10
Finished: SUCCESS
```

结果显示：可以在构建记录中，明显看到切换分支的效果，而且在任务结束后，做了一个提交标签的动作。

gitlab仓库查看效果

Administrator > tomcat_pro > Tags

Tags give the ability to mark specific points in history as being important

Filter by tag name



Last updated ▾



New tag

tomcat-web-12 创建代码标签
1524c7a4 · prod · 25 minutes ago



tomcat-web-11 创建代码标签
18f5a148 · test · 25 minutes ago



小结

gitlab集成

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

场景需求

在我们现有配置好的Gitlab和Jenkins环境中，目前的操作流程是，我们将代码推送到gitlab里面，然后在Jenkins里通过定制任务，执行构建子任务中的命令，实现项目的部署效果，这种方式的最大特点就是手工执行构建。

手工方式构建的方式，太繁琐了，虽然jenkins存在定时构建的功能，能实现按周期更新项目的效果，但是代码提交的频率和任务定时构建的频率是不好把控的，我们希望有一种功能，当我们提交到代码仓库后，然后自动触发jenkins构建任务，这是一个比较好的方式。

这种方式是建立在 Gitlab 和 jenkins 能够无缝联通的前提下。

操作思路

- 1 准备好 Gitlab 和 jenkins 环境
- 2 使用专用通信机制实现双方的认证
 - gitlab生成 api token
 - jenkins 定制 apitoken的认证凭证
 - 基于api凭证联通gitlab
- 3 定制hook触发任务

插件安装

到插件管理界面，安装"GitLab Plugin、Gitlab Authentication、Gitlab API"

| 可更新 | 可选插件 | 已安装 | 高级 | |
|-------------------------------------|---|--------|---------|---------------|
| 启用 | 名称 ↓ | 版本 | 上一个安装版本 | 卸载 |
| <input checked="" type="checkbox"/> | <div>Gitlab API Plugin</div> <div>This plugin provides GitLab API for other plugins.</div> | 1.0.6 | | <div>卸载</div> |
| <input checked="" type="checkbox"/> | <div>Gitlab Authentication plugin</div> <div>This is the an authentication plugin using gitlab OAuth.</div> | 1.10 | | <div>卸载</div> |
| <input checked="" type="checkbox"/> | <div>GitLab Plugin</div> <div>This plugin allows GitLab to trigger Jenkins builds and display their results in the GitLab UI.</div> | 1.5.20 | | <div>卸载</div> |

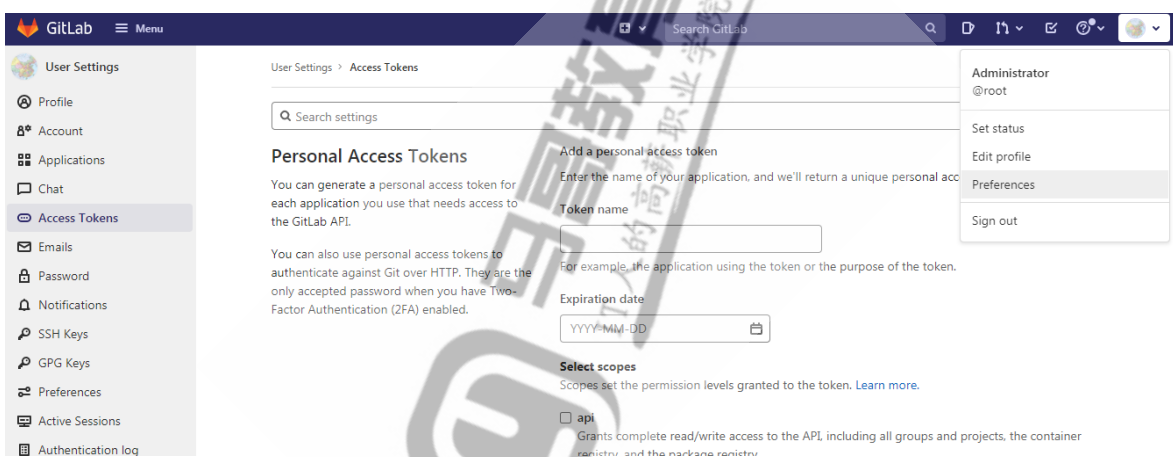
简单实践

需求

虽然我们可以借助于之前的两种认证方式来实现gitlab的正常构建任务，但是对于gitlab环境来说，jenkins提供了一个专门的认证方式"Gitlab API Token",接下来我们就完成以下这种专用的认证机制。

- gitlab生成 API认证token

登录Gitlab界面，点击右上角"图标"，找到"Settings"选项，然后点击，效果如下



点击左侧边栏的"Access Tokens",依次输入名称-jenkins，过期时间-选定，作用范围-api，效果如下

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

Token name

For example, the application using the token or the purpose of the token.

Expiration date

Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

☒ api

Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

选定完毕后，点击最下面的"Create personal access token",就会生成专用的gitlab访问的API token值

🔔 Your new personal access token has been created.

🔍 Search settings

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your new personal access token

7RvxYZ9hvyYvQ_5Lpx



Make sure you save it - you won't be able to access it again.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

可以看到，gitlab就会生成一个专用的token值"7RvxYZ9hvyYvQ_5Lpx"，接下来我们就可以来定制jenkins认证了。

- jenkins使用gitlab API通信

jenkins进入到凭证界面，点击"Add Credentials"，在kind栏选择"Gitlab API token"一项，然后将我们刚才拷贝的token信息，输入到"API token"后面的输入框，随便输入一个id即可，效果如下

Jenkins

Dashboard > 凭证 > 系统 > 全局凭据 (unrestricted)

返回到凭据列表

添加凭据

类型: GitLab API token

范围: 全局 (Jenkins, nodes, items, all child items, etc)

API token:

ID:

描述: gitlab API token 认证

确定

点击"OK"后jenkins对gitlab的认证账号创建完毕了

🏠 全局凭据 (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

| ID | 名称 | 类型 | 描述 |
|--------------------------------------|--|-------------------------------|---------------------|
| d883581a-8c76-4bf7-b268-4681d0f47d49 | root/***** | Username with password | |
| d7616c9e-3897-4e9a-81fd-335861ba4de6 | root | SSH Username with private key | |
| 53977118-daef-4cdb-aff2-32fe7e0d962b | GitLab API token (gitlab API token 认证) | GitLab API token | gitlab API token 认证 |

图标: 小 中 大

- jenkins 集成 Gitlab

点击jenkins的"Configure System"，进入到jenkins的全局配置界面，找到"Gitlab",效果如下

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

jenkins

A name for the connection

Gitlab host URL

http://192.168.8.15:7080/

The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

Credentials

GitLab API token (gitlab API token 认证) ▼

添加 ▼

API Token for accessing Gitlab

高级...

Test Connection

删除

Success

设定配置

connection name: jenkins -- 我们在gitlab生成API token时候, 用到的用户名

Gitlab host URL: http://192.168.8.14:7080 -- Gitlab的web界面登录地址

Credentials: 选择我们刚才创建的 Gitlab API token 账号即可

点击 Test connection

出现 success 说明账号联通功能成功

然后依次点击全局配置页面底下的"apply"和"save"即可。

- 简单测试

改造 tomcat-web 任务，将默认的git认证取消掉

| General | 源码管理 | 构建触发器 | 构建环境 | 构建 | 构建后操作 |
|---|------|-------|------|----|-------|
| 源码管理 | | | | | |
| <input type="radio"/> 无 | | | | | |
| <input checked="" type="radio"/> Git | | | | | |
| Repositories | | | | | |
| Repository URL | | | | | |
| ssh://git@192.168.8.15:2222/root/tomcat_pro.git | | | | | |
| Credentials | | | | | |
| - 无 - 添加 | | | | | |
| Name | | | | | |
| origin | | | | | |
| Refspec | | | | | |
| | | | | | |
| Add Repository | | | | | |

仅仅定义仓库的地址 ssh://git@192.168.8.15:2222/root/tomcat_pro.git
无需指定Credential, 点击高级后, 输入Name为origin

其他的内容不用修改, 依次点击 "apply" 和 "save"后, 点击 参数化构建

浏览器，查看效果，项目发布成功

小结

hook构建

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

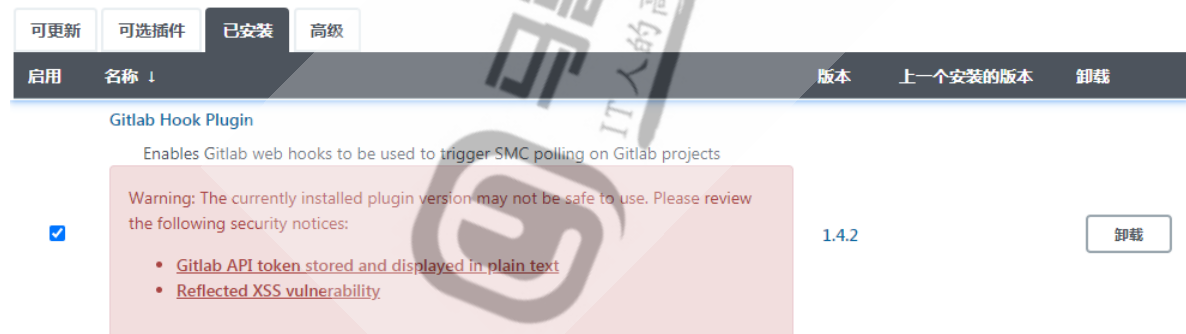
基础知识

需求

Gitlab正好提供了一个webhook功能，通过对gitlab的webhook进行属性设置，可以实现每当gitlab仓库代码被提交代码，他就会自动触发一个动作。这样我们就可以在jenkins上关联这个webhook的事件，从而自动触发jenkins的任务构建，这样就无需大量的人工干预甚至频繁构建，提高工作效率。

软件安装

插件管理界面，安装"GitLab Hook"



操作步骤

- 1 jenkins上创建一个可以被gitlab关联到的任务
- 2 gitlab定制hook触发的策略

注意：

默认情况下，gitlab 是无法向外发出请求的，所以我们需要单独设置向外通信的功能


基本配置

- 创建jenkins的任务

创建一个自由风格的任务gitlab-hook


输入一个任务名称

» 必填项

**Freestyle project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

确定



全局配置部分，定制gitlab的认证信息

General 源码管理 构建触发器 构建环境 构建 构建后操作

☐ GitHub 项目

☐ 使用自定义的工作空间 ?

GitLab Connection

jenkins

源代码仓库，定制gitlab的仓库信息和远程仓库名称，无需指定credential认证账号信息

General 源码管理 构建触发器 构建环境 构建 构建后操作

源码管理

☐ 无

☒ Git ?

Repositories ?

Repository URL ?

Credentials ?

- 无 -

添加

安装完gitlab hook插件后，在构建触发器部分多出一个"Build when a change is pushed to GitLab..."的选项，我们需要勾选该功能，具体配置如下：

General 源码管理 构建触发器 构建环境 构建 构建后操作

构建触发器

☐ 触发远程构建 (例如,使用脚本) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://192.168.8.14:8080/project/gitlab-hook ?

Enabled GitLab triggers

☒ Push Events

☐ Push Events in case of branch delete

☒ Opened Merge Request Events

点击右下角的"Advanced"，进入触发器的高级配置,在"Allowed branched"部分按需选择webhook作用于哪些分支，然后点击"Generate"生成唯一的secret token。

General

源码管理

构建触发器

构建环境

构建

构建后操作

Secret token

0c07308751f89b0f41d527fc9564c391

Generate

Clear

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

注意：

这里提示的 GitLab webhook URL地址，一定要记下来，因为会在Gitlab里面进行配置

http://192.168.8.14:8080/project/gitlab-hook

secret token 也必须保存下来：0c07308751f89b0f41d527fc9564c391

构建任务的内容，与之前的效果差不多，只不过我们这里使用一个完整的脚本来显示

General

源码管理

构建触发器

构建环境

构建

构建后操作

构建

Execute shell

命令

#!/bin/bash
创建定制的tomcat-web镜像
CONTAINER_NAME='tomcat-web'
IMAGE_VERSION=v0.1
CONTAINER_PORT=8080
HOST_PORT=666

查看 可用的环境变量列表

```
#!/bin/bash
# 创建定制的tomcat-web镜像
CONTAINER_NAME='tomcat-web'
IMAGE_VERSION=v0.1
CONTAINER_PORT=8080
HOST_PORT=666
DOCKERFILE_DIR='/data/docker/image'
PRO_NAME='tomcat-web'

echo "开始 ${PRO_NAME} 任务"

# 准备基本目录
[ -d ${DOCKERFILE_DIR} ] || mkdir -p ${DOCKERFILE_DIR}
tar -C ${PRO_NAME} -zcf ${PRO_NAME}/ROOT.tar.gz ROOT --remove-files
mv -i ${PRO_NAME} ${DOCKERFILE_DIR}

# 构建镜像
docker build -t ${CONTAINER_NAME}:v0.1 ${DOCKERFILE_DIR}/${PRO_NAME}

# 重启项目
num=$(docker ps | grep ${CONTAINER_NAME} | wc -l)
[ $num -eq 1 ] && docker rm -f ${CONTAINER_NAME}
docker run -d --name ${CONTAINER_NAME} -p ${HOST_PORT}:${CONTAINER_PORT}
${CONTAINER_NAME}:v0.1
```

```
# 清理旧代码
rm -rf ${DOCKERFILE_DIR}/${PRO_NAME}

echo "结束 ${PRO_NAME} 任务"
```

构建后动作



勾选"Push Only if Build Succeeds"

点击"Add Tag",添加标签属性

Tag to push 右侧添加"tomcat-hook-\$BUILD_NUMBER"

Tag message 右侧添加"gitlab hook 提交标签"

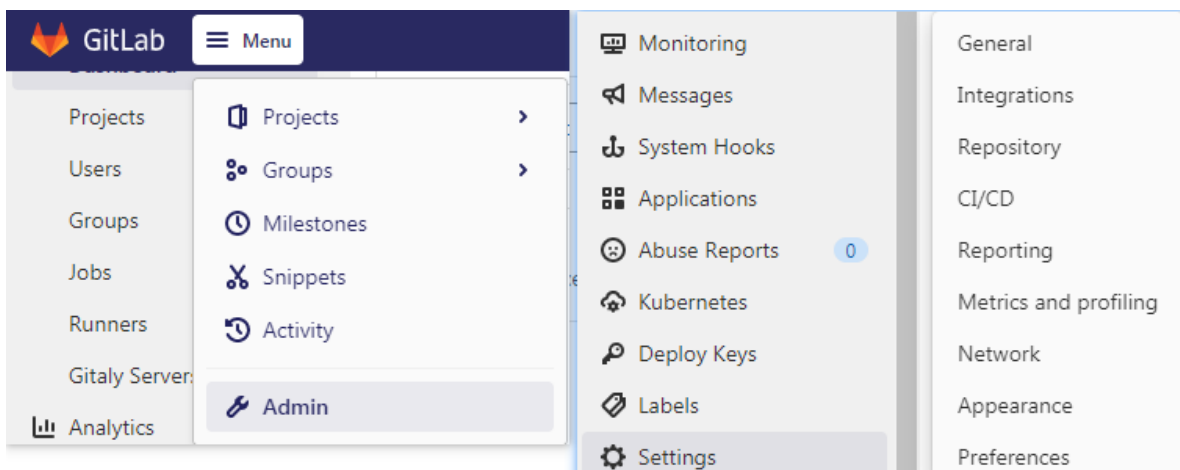
勾选"Create new tag"

Target remote name 右侧添加"origin",名字与仓库名称一致

依次点击"Apply"和"Save"

- gitlab 关联jenkins任务

设置网络，允许接受外部服务的请求，点击顶栏"Admin Area" -- "Settings" -- "Network"



进入"Network"页面后，勾选"Allow requests to the local network from hooks and services"

Search settings

Outbound requests

Collapse

Allow requests to the local network from hooks and services.

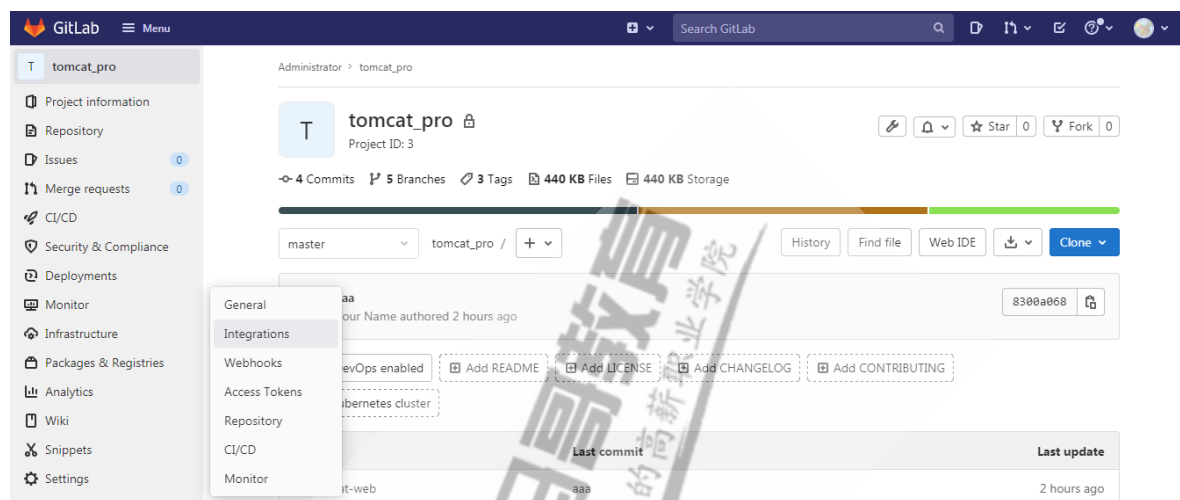
- ☒ Allow requests to the local network from web hooks and services
- ☒ Allow requests to the local network from system hooks

注意:

如果不做这项的话, 任何外部发送到gitlab的请求, 都会被阻止, 尤其是在做webhook的时候会发生如下报错:

```
url is blocked: Requests to the local network are not allowed
```

回到gitlab的项目首页, 并且进入到tomcat-web项目, 点击左下角的"Settings"



点击弹出栏的"Intergrations", 找到jenkins 关联的位置

Administrator > tomcat_pro > Integration Settings

 Search settings

Integrations

[Integrations](#) enable you to make third-party applications part of your GitLab workflow. If the available integrations don't meet your needs, consider using a [webhook](#).

Active integrations

| Integration | Description | Last updated |
|---|-------------|--------------|
| You haven't activated any integrations yet. | | |

Add an integration

| Integration | Description |
|-------------------------|-----------------------------------|
| Jenkins | Run CI/CD pipelines with Jenkins. |

注意:

Integrations 下面的 **webhook** 的超连接才是我们需要的

Add an integration 后面的jenkins是通过用户名和密码的方式进行认证

点击 webhook后, 将我们刚才复制的webhook链接地址以及secret token复制到这个地方, 保证"push event"处于勾选状态

Webhooks

[Webhooks](#) enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

URL must be percent-encoded if necessary.

Secret token

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ **Push events**

URL is triggered by a push to the repository

☐ **Enable SSL verification**

Add webhook

注意:

因为我们jenkins没有实现https, 所以这里取消"Enable SSL versification"

点击"Add webhook", 在webhooks列表中, 就会出现我们刚才定制的webhook

Project Hooks (1)

http://192.168.8.14:8080/project/gitlab-hook

Test ▼ Edit Delete

Push Events SSL Verification: disabled

点击右侧"Test"列表下面的"Push events",

☐ **Feature flag events**

URL is triggered when a feature flag is

☐ **Releases events**

URL is triggered when a release is crea

SSL verification

☒ **Enable SSL verification**

Add webhook

Push events

Tag push events

Issues events

Confidential issues events

Note events

Confidential note events

Merge requests events

Job events

Pipeline events

Project Hooks (1)

http://192.168.8.14:8080/project/gitlab-hook

Test ▼ Edit Delete

Push Events SSL Verification: disabled

就可以查看到测试效果, 如果是浅蓝色的状态返回值200, 表示配置成功

Hook executed successfully: HTTP 200

Search settings

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

URL must be percent-encoded if necessary.

Secret token

编写项目代码，将tomcat-web首页文件内容修改一下，然后推送到gitlab

```
cd /tmp/tomcat_pro
git checkout master
sed -i "s/买 1/买 hook/" tomcat-web/ROOT/index.jsp
git add . && git commit -m "hook" && git push origin master
```

查看jenkins的构建状态

Dashboard > gitlab-hook > #2

[返回到工程](#)
[状态集](#)
[变更记录](#)
[控制台输出](#)
[编辑编译信息](#)
[删除构建 '#2'](#)
[Polling Log](#)
[Git Build Data](#)
[打开 Blue Ocean](#)

构建 #2 (2021-8-26 18:43:16)

Started by GitLab push by Administrator

Changes

- 1. hook (details)

Started by GitLab push by Administrator

Revision: 86401e671b4080e3a5ed2eb5f51b271e521f118

Repository: ssh://git@192.168.8.15:2222/root/tomcat_pro.git

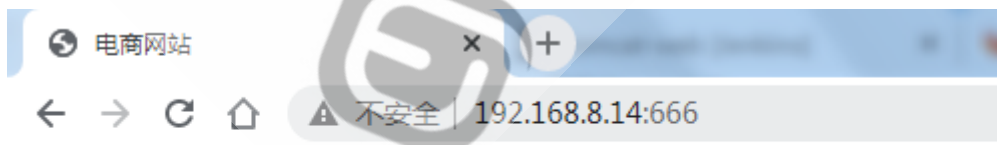
- origin/master

[永久保留这次编译](#)

启动时间: 30 sec
构建用时: 4.2 sec

[编辑描述](#)

浏览器访问192.168.8.14:666,检查tomcat的访问效果



开业大酬宾：买 hook 送 21

浏览器访问192.168.8.15:7080,检查gitlab仓库的代码效果

Tags give the ability to mark specific points in history as being important

Filter by tag name



Last updated



New tag

tomcat-hook-2 gitlab hook 提交标签

86401e67 · hook · 3 minutes ago



小结

进阶实践

流水线

学习目标

这一节，我们从 基础知识、基本配置、小结 三个方面来学习。

基础知识

简介

所谓的流水线，其实就是将我们之前的一个任务或者一个脚本就做完的事情，划分为多个子任务，然后分别执行，他们两者实现的最终效果是一样的，但是由于原始任务划分为多个子任务之后，以流水线的方式来执行，那么我们可以随时看任意子任务的执行效果，即使在某个阶段出现问题，我们也可以随时直接定位问题的发生点，大大提高项目的效率。

Pipeline，就是一套运行于**Jenkins**上的工作流框架，将原本独立运行于单个或者多个节点的任务连接起来，实现单个任务难以完成的复杂流程编排与可视化。**Pipeline**作为**Jenkins2.x**的最核心的特性，提供了一组可扩展的工具，通过**Pipeline Domain Specific Language (DSL) syntax**可以达到**Pipeline as Code**的目的，帮助**Jenkins**实现从**CI**到**CD**与**DevOps**的转变，下图就是一个简单的项目发布流水线样式。



功能特性

| 特性 | 说明 |
|------|---------------------------------------|
| 代码实现 | 以代码的形式实现，不仅可以被纳入版本控制，还可以通过编辑代码实现目标效果。 |
| 可持续性 | 任务的执行，不受jenkins等其他环境的限制。 |
| 操作灵活 | 子任务可以随意终止和继续 |
| 多功能 | 支持复杂CD要求，包括fork/join子进程，循环和并行执行工作的能力 |
| 可扩展 | 支持DSL的自定义扩展以及与其他插件集成。 |

表现样式

General

构建触发器高级项目选项流水线

General

源码管理构建触发器构建环境构建构建后操作

上面是 pipeline 风格，下面是 freestyle 风格

任务编排

自由风格借助于"Build Trigger"中的"Block xxx is building"或者"Post-build Action"中的"Build other projects" 属性将多个独立的巨型任务关联在一起，逻辑复杂(并行|级联依赖)场景下不太适用。

pipeline风格以一个任务的方式来实现，将各种我们能想到的任意复杂场景进行合理的组合。它不仅仅可以以web页面的方式进行定制化配置，还可以直接以Jenkinsfile的方式来实现。

不过，无论哪种方式，其核心都是大量的脚本代码的正常运行，而且在web界面定制脚本格式与jenkinsfile中的脚本格式一致。通常认为最佳实践是在Jenkinsfile Jenkins中直接从源代码控制中加载Pipeline。

基本配置

任务拆解

| 阶段标识 | 作用解析 |
|---------|--|
| Stage阶段 | 一个Pipeline可以划分为若干个Stage，每个Stage代表阶段任务中的一组操作。 |
| Node节点 | 一个Node就是一个Jenkins节点，或者是Master或node，是任务执行时候的目标主机环境。 |
| Step步骤 | Step是最基本的操作单元，表示完成阶段任务的一系列的命令组成，它遵循jenkinsfile语法。 |

注意：

jenkinsfile 一般有两种表现样式：脚本式 和 声明式

脚本式

```
node('工作结点') {  
    stage 'Prepare'  
        任务1执行命令  
    stage 'Checkout'  
        任务2执行命令  
    stage 'Build'  
        任务3执行命令  
    stage 'Deploy'  
        任务4执行命令  
    stage 'Test'  
        任务5执行命令  
}
```

特点解析

最外层有node{}包裹

可直接使用groovy语句

这是jenkins流水线初期通用的一种样式，而且符合我们平常的linux命令操作习惯

groovy语法

Groovy是一种基于Java平台的面向对象语言，使用方式基本与使用 Java代码的方式相同，使用样式如下：

```
class Example {
    static void main(String[] args) {
        // x is defined as a variable
        String x = "Hello";

        // The value of the variable is printed to the console
        println(x);
    }
}
```

自从jenkins进入到2.x时代后，Groovy语法开始大幅度发展，进而演进出 声明式 的语法，而且官方也提供了非常多的复杂命令，用于实现各种循环、条件、调试、并行、选择等场景。

声明式

```
pipeline {
    agent any
    stages{
        stage ('Prepare'){
            steps{
            }
        }
        ...
        stage ('Test'){
            steps{
            }
        }
    }
}
```

声明式特点

最外层必须由pipeline{ //do something }来进行包裹

不需要分号作为分隔符，每个语句必须在一行内

不能直接使用groovy语句（例如循环判断等），需要被script {}包裹

常见指令

无论是脚本式语法还是声明式语法，他们本质上都是执行各种命令，对于不同的命令需要采用专用的语法来实现指定的功能，常见的语法命令及其样式如下：

| 命令 | 作用 | 示例 |
|------|------|--|
| echo | 输出内容 | echo 'Building..' |
| sh | 执行命令 | sh 'command_sample' sh([script: 'echo hello']) |
| git | 获取代码 | git credentialsId: 'b...a38', url: ' http://xx/root/xx.git ', branch: 'master' git ' https://xx/xx.git ' |
| env | 设置变量 | env.PATH = "/usr/local/java/bin:\$PATH" |

注意:

jenkins 在安装的时候, 自动帮我们提供了大量的帮助手册

<http://192.168.8.14:8080/job/pipeline-test/pipeline-syntax/>

插件管理

到插件管理界面, 安装"Pipeline"相关的插件, 为了方便后续的BlueOcean的学习, 我们可以提前将流水线的可视化插件也安装了。

| 可更新 | 可选插件 | 已安装 | 高级 |
|--------------------------|--|---------|------------------|
| Install ↑ | Name | Version | Released |
| <input type="checkbox"/> | Pipeline <div>Command Line Interface Miscellaneous Agent Launchers and Controllers Build Triggers</div> <p>A suite of plugins that lets you orchestrate automation, simple or complex. See Pipeline as Code with Jenkins for more details.</p> | 2.6 | 2 yr 10 mo ago |
| <input type="checkbox"/> | Pipeline: Milestone Step Plugin that provides the milestone step | 1.3.2 | 6 mo 25 days ago |
| <input type="checkbox"/> | Pipeline: REST API <div>User Interface</div> <p>Provides a REST API to access pipeline and pipeline run data.</p> | 2.19 | 9 mo 6 days ago |
| <input type="checkbox"/> | Pipeline: Stage View <div>User Interface</div> <p>Pipeline Stage View Plugin.</p> | 2.19 | 9 mo 6 days ago |

小结

简单实践

学习目标

这一节, 我们从 简单实践、进阶实践、小结 三个方面来学习。


简单实践

创建一个流水线风格的任务"pipeline", 效果如下

输入一个任务名称


pipeline-test

» 必填项



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job

确定

直接找到最下面的Pipeline，可以选择右上角的简单示例，选择一个"Hello world"，

Dashboard » pipeline-test »

General 构建触发器 高级项目选项 流水线

流水线

定义

Pipeline script

脚本

```
1 pipeline {
2   agent any
3
4   stages {
5     stage('Hello') {
6       steps {
7         echo 'Hello World'
8       }
9     }
10  }
11 }
12
```

Hello World

try sample Pipeline...

Hello World

GitHub + Maven


Scripted Pipeline

☒ 使用 Groovy 沙盒






保存

应用


一旦选择好对应的模式后，就会直接在输入框内部写上一个简单的示例，但是这个示例有些太简单，所以我们可以点击左下角的"Pipeline Syntax(流水线语法)"，查看更多的pipeline语法。效果如下


 Jenkins


🔍 查找


  1  2  jenkins web admin  注销


Dashboard » pipeline-test » 流水线语法


 返回


 片段生成器


 Declarative Directive Generator


 Declarative Online Documentation

 步骤参考

 全局变量参考

 在线文档

 Examples Reference

 IntelliJ IDEA GDSDL

概览

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

步骤

示例步骤

VersionNumber: Determine the correct version number

VersionNumber

Version Number String

左侧是一些帮助选项，右侧是确认信息如何编写的简单语法，在右侧的"sample step"右侧的下拉框中选择我们想要的执行语法，并输入一些定制化的内容，点击最下面的"Generate Pipeline Script"，就可以生成我们想要的groovy语法。

Dashboard > pipeline-test > 流水线语法

步骤参考

全局变量参考

在线文档

Examples Reference

IntelliJ IDEA GDSL

步骤

示例步骤

echo: Print Message

echo

消息

你好啊 jenkins pipeline

生成流水线脚本

echo '你好啊 jenkins pipeline'

我们还可以点击左侧的"Examples Reference", 查看一下模板文件,

jenkins.io/doc/pipeline/examples/#parallel-from-grep

Jenkins cd Blog Documentation Plugins Community Subprojects About English Download Search

Pipeline Examples

The following examples are sourced from the [pipeline-examples](#) repository on GitHub and contributed to by various members of the Jenkins project. If you are interested in contributing your own example, please consult the [README](#) in the repository.

Ansi Color Build Wrapper

Synopsis

This shows usage of a simple build wrapper, specifically the `AnsiColor` plugin, which adds ANSI coloring to the console output.

```
// This shows a simple build wrapper example, using the AnsiColor plugin.
node {
    // This displays colors using the 'xterm' ansi color map.
    ansiColor('xterm') {
        // Just some echoes to show the ANSI color.
        stage "\u001B[31mI'm Red\u001B[0m Now not"
    }
}
```

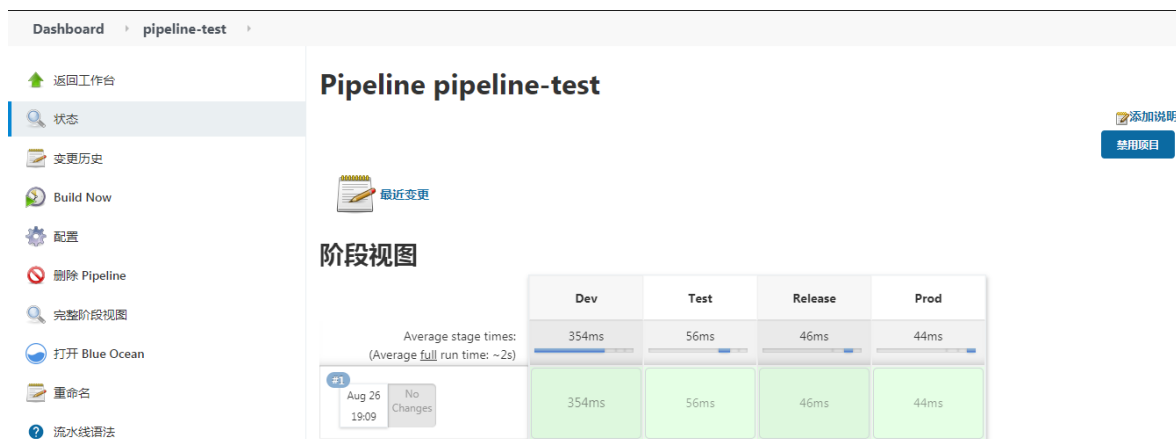
Table of Contents

- Ansi Color Build Wrapper
- Archive Build Output Artifacts
- Artifactory Generic Upload Download
- Artifactory Gradle Build
- Artifactory Maven Build
- Configfile Provider Plugin
- External Workspace Manager
- Get Build Cause
- Gitcommit
- Gitcommit_changeset
- Ircnotify Commandline
- Jobs In Parallel
- Load From File
- Maven And Jdk Specific Version

根据模板文件的内容，我们可以简单的来改造一下默认的流水线代码

```
node {
    stage('Dev') {
        echo "开发环境下执行项目构建"
    }
    stage('Test') {
        echo "测试环境下执行项目构建"
    }
    stage('Release') {
        echo "预发布环境下执行项目构建"
    }
    stage('Prod') {
        echo "生产环境下执行项目构建"
    }
}
```

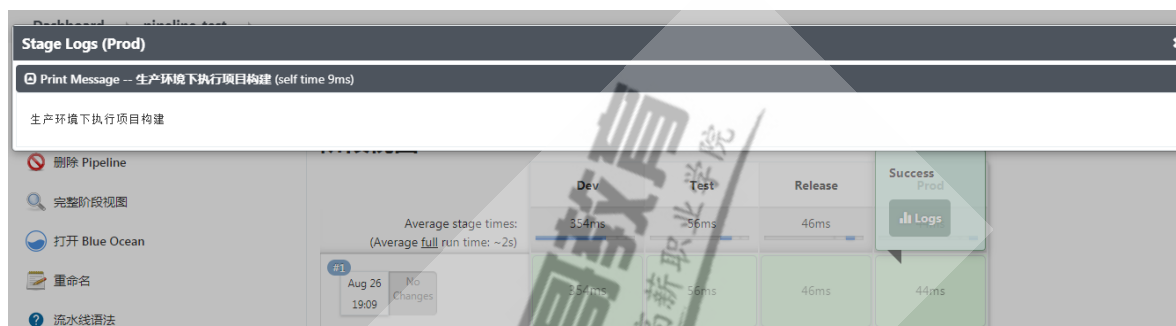
依次点击"Apply"和"Save"之后，我们点击一下"Build now"，不仅仅可以看到阶段步骤，还可以看到每一阶段执行了多长时间。效果如下



注意：

当我们查看历史记录输出的时候，会发现，每一步的执行都是一个独立的任务，也就是说，进入到当前任务的根目录下执行子任务。

当我们把鼠标放到具体的步骤上面，可以看到更多的显示信息



这样每一步出现问题，我们都可以直接看到该部分的执行数据。

进阶实践

语法特点：

| 关键字 | 作用 |
|----------|---------------------------------|
| pipeline | 声明其内容为一个声明式的pipeline脚本 |
| agent | 执行节点（job运行的slave或者master节点） |
| stages | 阶段集合，包裹所有的阶段（例如：打包，部署等各个阶段） |
| stage | 阶段，被stages包裹，一个stages可以有多个stage |
| steps | 步骤,为每个阶段的最小执行单元,被stage包裹 |
| post | 执行构建后的操作，根据构建结果来执行对应的操作 |
| input | 用于临时中断流水线，可以在内部配置一些按钮等属性 |

按照声明式的语法结构，我们将之前的job改造为如下效果，内容如下

```
pipeline {  
  agent any
```



```

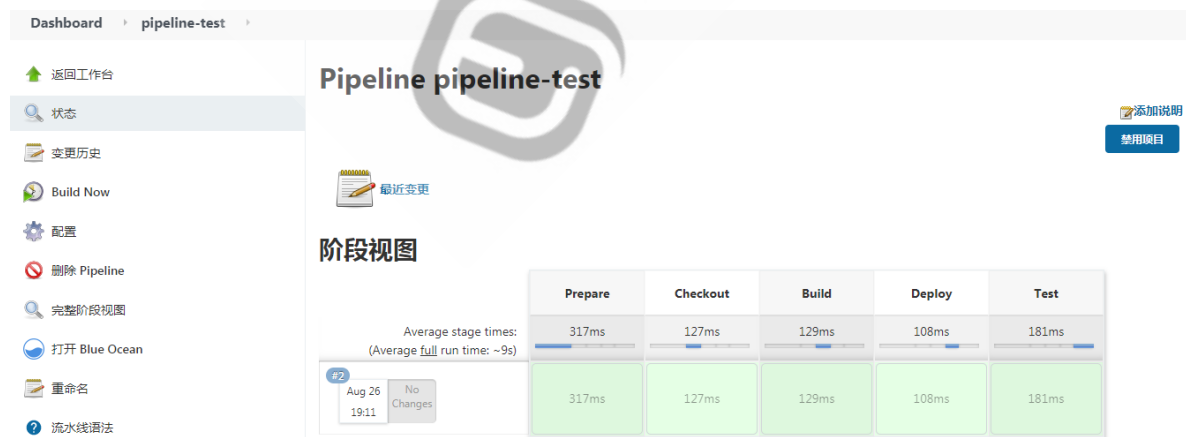
stages{
  stage ('Prepare'){
    steps{
      echo "准备环境"
    }
  }
  stage ('Checkout'){
    steps{
      echo "获取代码"
    }
  }

  stage ('Build'){
    steps{
      echo "构建镜像"
    }
  }
  stage ('Deploy'){
    steps{
      echo "部署项目"
    }
  }

  stage ('Test'){
    steps{
      echo "测试效果"
    }
  }
}

```

保存项目之后，我们再来"Build now"一下，效果如下



小结

进阶实践

学习目标

这一节，我们从 命令要点、案例改造、小结 三个方面来学习。

命令要点

流水线命令特点

pipeline的声明式语法有一个非常重要的问题，

- 1 steps内部的命令，每一条单独的命令都在当前任务的工作目录下执行。

即使A命令切换到了一个新的目录，接下来的B命令并不会在对应的新目录中执行，而是在当前任务的工作目录下执行。如果非要在切换后的目录下执行命令B，那么采用shell中的 && 符号将多条命令拼接在一起即可。

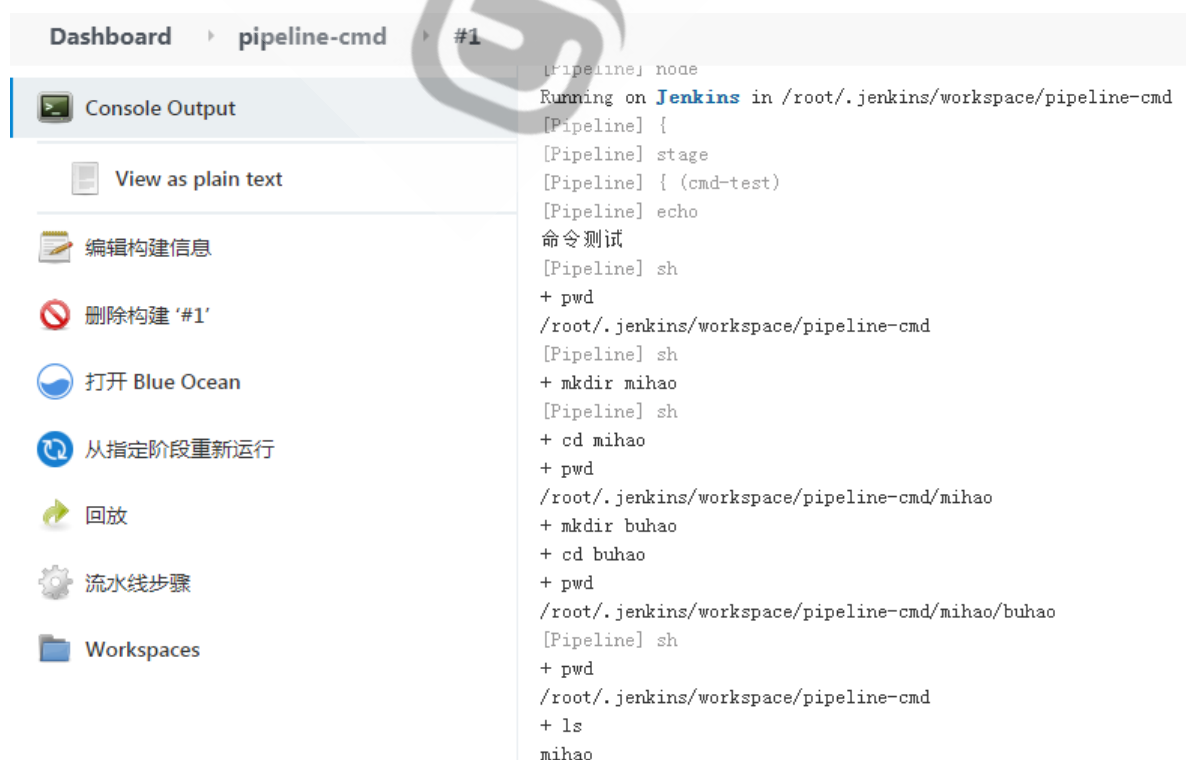
- 2 在流水线任务中，中文方式编写命令不友好，推荐在外面写好命令，再拷贝到里面
- 3 默认情况下，不支持shell里面的表达式，因为groovy有自己的条件表达式
- 4 如果jenkins的工作目录下存在同名目录，则获取失败

建立一个临时的流水线测试一下：

```
pipeline {
    agent any

    stages {
        stage('cmd-test') {
            steps {
                echo "命令测试"
                sh 'pwd'
                sh 'mkdir mihao '
                sh 'cd mihao && pwd && mkdir buhao && cd buhao && pwd'
                sh 'pwd && ls'
            }
        }
    }
}
```

执行任务构建后的效果



The screenshot shows the Jenkins interface for a pipeline build. The breadcrumb navigation is 'Dashboard > pipeline-cmd > #1'. The 'Console Output' tab is selected, displaying the following log:

```
[Pipeline] node
Running on Jenkins in /root/.jenkins/workspace/pipeline-cmd
[Pipeline] {
[Pipeline] stage
[Pipeline] { (cmd-test)
[Pipeline] echo
命令测试
[Pipeline] sh
+ pwd
/root/.jenkins/workspace/pipeline-cmd
[Pipeline] sh
+ mkdir mihao
[Pipeline] sh
+ cd mihao
+ pwd
/root/.jenkins/workspace/pipeline-cmd/mihao
+ mkdir buhao
+ cd buhao
+ pwd
/root/.jenkins/workspace/pipeline-cmd/mihao/buhao
[Pipeline] sh
+ pwd
/root/.jenkins/workspace/pipeline-cmd
+ ls
mihao
```

On the left sidebar, there are several options: 'View as plain text', '编辑构建信息' (Edit build information), '删除构建 '#1'' (Delete build '#1'), '打开 Blue Ocean' (Open Blue Ocean), '从指定阶段重新运行' (Run from specified stage), '回放' (Replay), '流水线步骤' (Pipeline steps), and 'Workspaces'.

结果显示:

每一个sh命令都是在当前任务的工作目录中执行,如果需要切换工作目录,需要将多条命令合并,这是由于jenkins的job任务彼此之间是独立的运行空间,所以资源不共享。

流水线测试示例2

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                sh 'num=$(docker ps | grep tomcat-web | wc -l)'
                echo "$num"
                echo '-----'
            }
        }
    }
}
```

执行任务构建后的效果



The screenshot shows the Jenkins interface with the 'Console Output' tab selected. The left sidebar contains navigation links: '返回到项目', '状态', '变更历史', 'Console Output' (selected), 'View as plain text', '编辑构建信息', '删除构建 '#2'', '打开 Blue Ocean', '从指定阶段重新运行', and '回放'. The main console output area displays the following text:

```
Started by user jenkins web admin
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /root/.jenkins/workspace/pipeline-cmd
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] sh
+ wc -l
+ grep tomcat-web
+ docker ps
+ num=0
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
groovy.lang.MissingPropertyException: No such property: num for class: groovy.lang.Binding
    at groovy.lang.Binding.getVariable(Binding.java:63)
```

结果显示:

groovy 不支持我们常用的这种bash进阶语法,因为groovy 有自己独有的语法

流水线示例3

```

pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                sh 'git clone ssh://git@192.168.8.15:2222/root/tomcat_pro.git'
                echo '-----'
                sh 'git clone ssh://git@192.168.8.15:2222/root/tomcat_pro.git'
            }
        }
    }
}

```

执行任务构建后的效果

Dashboard ▶ pipeline-cmd ▶ #4

- 返回到项目
- 状态
- 变更历史
- Console Output**
- View as plain text
- 编辑构建信息
- 删除构建 '#4'
- 打开 Blue Ocean
- 从指定阶段重新运行
- 回放
- 流水线步骤

Console Output

Started by user **jenkins web admin**
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node:
Running on **Jenkins** in /root/.jenkins/workspace/pipeline-cmd
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] sh
+ git clone ssh://git@192.168.8.15:2222/root/tomcat_pro.git
正克隆到 'tomcat_pro'...
[Pipeline] echo

[Pipeline] sh
+ git clone ssh://git@192.168.8.15:2222/root/tomcat_pro.git
fatal: 目标路径 'tomcat_pro' 已经存在，并且不是一个空目录。
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 128
Finished: FAILURE

结果显示：如果任务目录下，出现同名目录就会导致任务中断

案例改造

需求

将我们之前的pipeline-test 任务改造为 webhook的流水线模式

jenkins 定制webhook 任务

| General | 构建触发器 | 高级项目选项 | 流水线 |
|---|-------|--------|-----|
| 构建触发器 | | | |
| <input type="checkbox"/> Build after other projects are built | | | |
| <input type="checkbox"/> Build periodically | | | |
| <input checked="" type="checkbox"/> Build when a change is pushed to GitLab. GitLab webhook URL: http://192.168.8.14:8080/project/pipeline-test | | | |
| Enabled GitLab triggers | | | |
| <input checked="" type="checkbox"/> Push Events | | | |

| Webhook | URL |
|--|---|
| Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook. | <input type="text" value="http://192.168.8.14:8080/project/pipeline-test"/> |
| | URL must be percent-encoded if necessary. |
| | Secret token |
| | <input type="text" value="d7d1bcc5ffdabf55f8c22cbbddb7cc79"/> |
| Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header. | |

- 1 启动 **webhook** 模式，并设定**secret**认证密码
 http://192.168.8.14:8080/project/pipeline-test
 d7d1bcc5ffdabf55f8c22cbbddb7cc79
- 2 **gitlab** 的webhook 关联jenkins的任务

脚本改造

```

pipeline {
  agent any
  stages {
    stage('Prepare') {
      steps {
        echo "开始 tomcat_web 任务"
      }
    }
    stage('get_code') {
      steps {
        echo "获取代码"
        sh 'git clone ssh://git@192.168.8.15:2222/root/tomcat_pro.git'
        sh 'cd tomcat_pro && pwd && git config --local user.name "jenkins" && git config --local user.email "jenkins@example.com"'
      }
    }
    stage('update_code') {
      steps {
        echo "更新代码"
        sh '[-d /data/docker/image ] || mkdir -p /data/docker/image'
        sh 'tar -C tomcat_pro/tomcat-web -zcf tomcat_pro/tomcat-web/ROOT.tar.gz ROOT --remove-files'
        sh 'mv -i tomcat_pro/tomcat-web /data/docker/image/'
      }
    }
    stage('build_code') {
      steps {
        echo "构建镜像"
        sh 'docker build -t tomcat-web:v0.1 /data/docker/image/tomcat-web'
      }
    }
    stage('Deploy') {
      steps {
        echo "部署项目"
        sh 'docker rm -f tomcat-web'
      }
    }
  }
}

```

```

        sh 'docker run -d --name tomcat-web -p 666:8080 tomcat-web:v0.1'
    }
}
stage('check') {
    steps {
        echo "检查项目"
    }
}
stage('clean') {
    steps {
        echo "清理旧代码"
        sh 'rm -rf /data/docker/image/tomcat-web'
    }
}
stage('stop') {
    steps {
        echo "结束 tomcat_web 任务"
        sh 'rm -rf tomcat_pro'
    }
}
}
}

```

注意：

我们这里仅仅是将我们最初的构建命令放置到jenkinsfile的不同的steps中的位置了，因为每一步都是从任务的根目录下执行的，所以我们在涉及到文件使用的时候，需要注意一下命令涉及到的文件路径

执行后效果

阶段视图

| | | Prepare | get_code | update_code | build_code | Deploy | Check | Clean | Stop |
|---|-------------------------------|---------|----------|-------------|------------|--------|-------|-------|-------|
| Average stage times: (Average full run time: ~13s) | | 226ms | 2s | 736ms | 1s | 917ms | 197ms | 446ms | 194ms |
| #6 | Aug 27 07:04 No Changes | 249ms | 2s | 1s | 2s | 2s | 325ms | 722ms | 344ms |

小结