Jenkins Pipeline 学习笔记-20171207

0.有益参考资料

Jenkins与Docker的持续集成实践：https://yq.aliyun.com/articles/224577
Jenkins Pipeline插件十大最佳实践：http://blog.didispace.com/jenkins-pipeline-top-10-action/

1.镜像：jenkinsci/blueocean（docker pull jenkinsci/blueocean）
地址：https://hub.docker.com/r/jenkinsci/blueocean/

2.macos 下安装 jenkins

```
docker run -u root --rm -d -p 8080:8080 -v jenkins-data:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --
name jenkins-blueocean jenkinsci/blueocean
```

3.解锁Jenkins UI界面，并配置Jenkins

- 访问：http://127.0.0.1:8080
- 解锁 Jenkins UI界面（Unlock Jenkins）：初试密码通过 docker logs jenkins-blueocean 可以从日志中查看到，或查看文件 /var/jenkins_home/secrets/initialAdminPassword，同时这个密码也是默认创建的Jenkins管理用户 admin 的初始密码
- 安装Jenkins插件：选择Install suggested plugins即可
- 创建第一个管理员账号（Creating the first administrator user）：wenba | FjRVsdh6o6BJwc5nfVK1Y1c4NJeEqU5m | jie.yu@wenba100.com
- 插件更新：更新 Blue Ocean 至 1.3.4 版本；更新 Jenkins 至 2.89.1 版本；

4.访问jenkins容器

```
docker exec -it jenkins-blueocean bash
```

5.通过 Jenkinsfile 描述 Pipeline，实现 Pipeline as Code

```
// Declarative //
pipeline {
    agent any ①

    stages {
        stage('Build') { ②
            steps { ③
                sh 'make' ④
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml' ⑤
            }
        }
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

1 agent indicates that Jenkins should allocate an executor and workspace for this part of thePipeline.
2 stage describes a stage of this Pipeline.
3 steps describes the steps to be run in this stage
4 sh executes the given shell command
5 junit is a Pipeline step provided by the plugin:junit[JUnit plugin] for aggregating test reports.

6.Pipeline基本流程

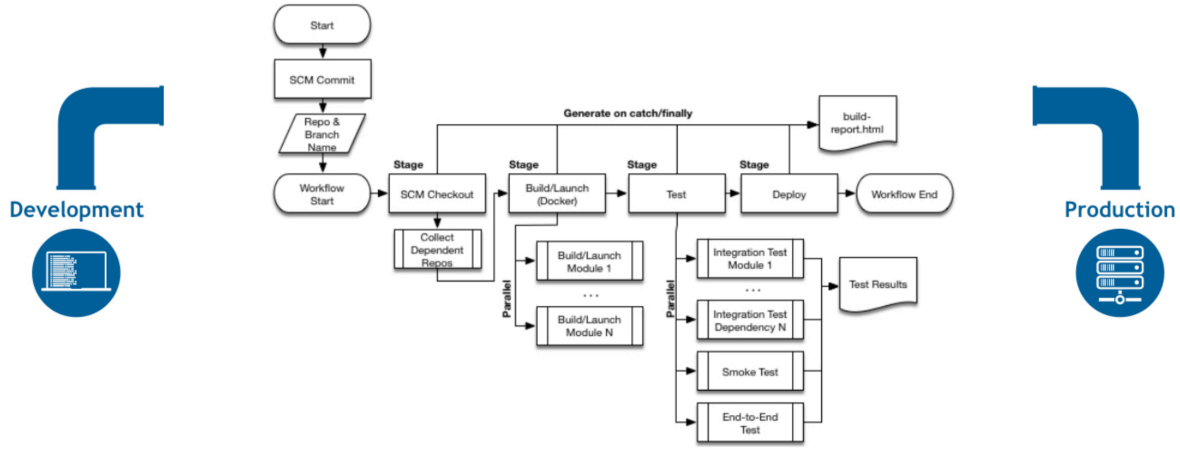


Figure 1. Pipeline Flow

重要名词解释
Step：一个单一的任务（single task），告诉 Jenkins 需要做什么，如 sh 'make' 即是执行 shell 命令 make
Node：Pipeline执行的大部分工作都是在一个或多个声明的节点步骤的上下文中完成的。 限制节点内部的工作可以做两件事情：

1. 通过向Jenkins队列添加一个项目来安排块中包含的步骤。 只要执行程序在节点上空闲，这些步骤就会运行。
2. 创建一个工作空间（特定于该特定管道的目录），在该工作空间中，可以对从源代码管理检出的文件执行工作。

Stage：阶段是定义整个管道的一个概念上不同的子集的步骤，例如："构建"，"测试"和"部署"，许多插件使用它来可视化或呈现Jenkins管道状态/进度。

另一个角度理解：

1. step，其实跟jenkins1中的概念一样， 是jenkins里job中的最小单位，可以认为是一个脚本的调用和一个插件的调用。
2. node， 是pipleline里groovy的一个概念，node可以给定参数用来选择agent，node里的steps将会运行在node选择的agent上。这里与jenkins1的区别是， job里可以有多个node，将job的steps按照需求运行在不同的机器上。例如一个job里有好几个测试集合需要同时运行在不同的机器上。
3. stage，是pipeline里groovy里引入的一个虚拟的概念，是一些step的集合，通过stage我们可以将job的所有steps划分为不同的stage，使得整个job像管道一样更容易维护。pipleline还有针对stage改进过的view，使得监控更清楚。

jenkins的实现是标准的master/slave模式，用户与master交互，master将job分布到slave上运行。
jenkins的基本概念：

1. master, 也就是jenkins的server，是jenkins的核心，主要负责job的定时运行，将job分发到agent运行，和对job运行状态的监控。
2. agent/slave/node，agent是相对于master的概念，主要作用是监听master的指令，然后运行job。
3. executor，executor是虚拟的概念，每一个agent都可以设置executor的数量，表示可以同时运行的job的数量。

7.安装 Pipeline插件

Managing Plugins：https://jenkins.io/doc/book/managing/plugins/
Creating your first Pipeline：https://jenkins.io/doc/pipeline/tour/hello-world/

8.定义 Pipeline

一个基本的 Pipeline 可以通过以下任何一种方式创建：

- 通过直接在Jenkins Web UI中输入脚本，即 Pipeline script 方式（Groovy代码）
- 通过创建一个可以签入到项目的源代码控制库中的 Jenkinsfile，即 Pipeline script from SCM 方式

Jenkinsfile 的内建文档见：http://127.0.0.1:8080/pipeline-syntax/

- Snippet Generator：自动生成各个 step 的示例代码
- Global Variables Reference：可直接使用的全局变量
  - env：环境变量可以从Groovy代码作为env.VARNAME或简单地作为VARNAME访问（http://127.0.0.1:8080/pipeline-syntax/globals#env）
  - params：将构建中定义的所有参数公开为具有各种类型值的只读映射
  - currentBuild：可以用来引用当前正在运行的构建

参考文档
Pipeline Steps Refrence：https://jenkins.io/doc/pipeline/steps/
Pipeline Examples：https://jenkins.io/doc/pipeline/examples/

一个完整的Jenkinsfile示例

```
#!/usr/bin/env groovy

// Jenkinsfile (Scripted Pipeline)
node('docker') {
    checkout scm

    stage('发布信息') {
        def userInput = input(id: 'userInput', message: '请提供待测试和发布的服务名和版本号', parameters: [[$class:
'TextParameterDefinition', defaultValue: 'aixue-test', description: '待发布服务名?', name: 'service_name'],[$class:
'TextParameterDefinition', defaultValue: 'v1.0.0', description: '待发布服务版本号?', name: 'tag_version']])
        env.SERVICE_NAME = userInput['service_name']
        env.TAG_VERSION = userInput['tag_version']
        env.RUNNING_ENVIRONMENT = "test"
        echo ("Service_name: "+env.SERVICE_NAME)
        echo ("Tag_version: "+env.TAG_VERSION)
    }

    stage('镜像构建') {
        echo 'git clone'
        echo 'docker build'
        echo 'docker push'
    }

    stage('测试环境部署发布') {
        echo 'aliyun_docker_deploy'
        echo 'aliyun_docker_deploy_rollback'
    }

    stage('测试') {
        echo "notify test team by mail"
        emailext body: '''待测试服务: $SERVICE_NAME
        待测试版本: $TAG_VERSION
        $JOB_NAME - Build # $BUILD_NUMBER - ${currentBuild.currentResult}
        Check console output at $BUILD_URL or $JOB_URL to view the results.''', subject:"测试通知: 服务 $SERVICE_NAME ,版本
$TAG_VERSION 可以开始测试啦! ",from:"zabbix@wenba100.com",to:"jie.yu@wenba100.com"
        input '''测试是否通过? Proceed (通过) or Abort (不通过) '''
    }

    stage('生产集群确认') {
        env.RUNNING_ENVIRONMENT = input(id: 'deployEnv', message: '请提供待发布的生产环境集群', parameters: [[$class:
'TextParameterDefinition', defaultValue: 'prod_A', description: '待发布生产环境集群? prod_A or prod_B', name:
'runnning_environment']])
        echo ("Running_environment: "+env.RUNNING_ENVIRONMENT)
    }

    stage('生产环境部署发布') {
        echo 'aliyun_docker_deploy'
        echo 'aliyun_docker_deploy_rollback'
    }

    stage('回归测试验证') {
        input ("回归测试是否通过? \012 Proceed (通过) or Abort (不通过) ")
        if (currentBuild.currentResult == 'SUCCESS') {
            emailext(subject:"发布完成通知: $JOB_NAME - Build # $BUILD_NUMBER -
${currentBuild.currentResult}!",body:"$JOB_NAME - Build # $BUILD_NUMBER - ${currentBuild.currentResult}: Check console
output at $BUILD_URL to view the results.",from:"zabbix@wenba100.com",to:"jie.yu@wenba100.com")
        } else {
                emailext(subject: "发布失败通知: Job '${JOB_NAME}' (${BUILD_NUMBER}) is failed",body: "Please go to
${BUILD_URL} and verify the build",from:"zabbix@wenba100.com",to:"jie.yu@wenba100.com")
        }
    }
}
```

9.Pipeline语法

Pipeline Syntax：https://jenkins.io/doc/book/pipeline/syntax/
Using a Jenkinsfile：https://jenkins.io/doc/book/pipeline/jenkinsfile/
支持两种独立的语法格式：

- Declarative Pipeline（声明式）：限制了对用户可用的更严格和预定义的结构，使其成为更简单的连续交付管道的理想选择
- Scripted Pipeline（脚本式）：提供了很少的限制，更加灵活，适合高级用户和那些需求更复杂的用户

语法基本结构
Declarative Pipeline

- Sections（区块）：
  - agent

    - post
    - stages
    - steps
- Directives（指令）：
    - environment
    - options
    - parameters
    - triggers
    - stage
    - tools
    - when
    - parallel
- Steps（步骤）：
    - script

Scripted Pipeline

- Flow Control
- Steps

10.利用 Jenkins 编译测试 PHP 项目

Jenkins and PHP：https://jenkins.io/solutions/php/
Template for Jenkins Jobs for PHP Projects：http://jenkins-php.org/index.html

11.多分支 Pipeline，实现指定代码仓库分支

Bracnches and Pull Requests：https://jenkins.io/doc/book/pipeline/multibranch/

12.利用 Docker 执行 Pipeline

Using Docker with Pipeline：https://jenkins.io/doc/book/pipeline/docker/

- 自定义 docker 执行环境
- 缓存容器数据，加速 pipeline 执行
- 使用多个不同类型容器（我们用不上）
- 使用 Dockerfile 构建运行环境

参考：
1）使用阿里云容器服务Jenkins 2.0实现持续集成之Pipeline篇：https://yq.aliyun.com/articles/64970
2）用aliyun容器搭建基于docker的jenkins动态agent节点：http://www.wantchalk.com/c/devops/docker/2017/07/03/deploy-jenkins-dynamic-agent-node-based-on-docker-with-aliyun-container.html

实现一：原生插件

```
node {
    docker.withServer('tcp://master4g3.cs-cn-hangzhou.aliyun.com:11506', 'aliyun-swarm-credentials') {
        docker.withRegistry('http://registry.cn-hangzhou.aliyuncs.com', 'docker-registry-credentials') {
            docker.image('wenba/nginx-php-status').inside {
```

实现二：第三方插件（Yet Another Docker）（推荐）

```
node('docker-jenkins-php') {
```

Yet Another Docker插件配置

```
Cloud Name: aliyun-docker-swarm-test
Docker URL: tcp://master4g3.cs-cn-hangzhou.aliyun.com:11506
Host Credentials: aliyun-swarm-credentials
Type: NETTY
Max Containers: 500
Images:
    Docker Template
        Docker Image Name: registry.aliyuncs.com/acs-sample/jenkins-slave-dind-java:latest
        Pull strategy: Pull never
        Registry Credentials: aliyun-registry-credentials
        Create Container Settings
            Volumes:
                /var/run/docker.sock:/var/run/docker.sock
                /home/jenkins/.m2:/root/.m2
        Jenkins Slave Config
            Remote Filing System Root: /home/jenkins
            Labels: docker
            用法: 只允许运行绑定到这台机器的Job
            Launch method: Docker SSH computer launcher
                Credentials: jenkins-slave-credentials
```

注：版本 0.1.0-rc46 与 aliyun registry的API有冲突，导致无法pull image，故通过本地拉取基础镜像来规避

各种错误汇总

错误1：Wrote authentication to /var/jenkins_home/.dockercfg
原因：未知

错误2：
/var/jenkins_home/workspace/kins-pipeline-test02_master-GC3UCLFOCPAAKVKUYBFISLHA2YZY6J2CNEZMIMRII7HQ3CWY524Q@tmp/durable-0cbfa714/script.sh: docker: not found
原因：jenkins master中没有 docker 命令，导致无法执行 docker 相关的操作
解决：重新安装最新的 Blue Ocean 的 Jenkins 镜像作为 master 节点
参考文档：https://stackoverflow.com/questions/42685676/jenkins-docker-pipelining-inside-docker

错误3：
com.github.kostyasha.yad_docker_java.com.github.dockerjava.api.exception.DockerClientException: Could not pull image: Pulling registry.cn-hangzhou.aliyuncs.com/wenba/jenkins-slave-golang...
    at com.github.kostyasha.yad_docker_java.com.github.dockerjava.core.command.PullImageResultCallback.awaitSuccess(PullImageResultCallback.java:50)
    at com.github.kostyasha.yad.commons.DockerPullImage.exec(DockerPullImage.java:135)
    at com.github.kostyasha.yad.DockerCloud.provisionWithWait(DockerCloud.java:229)
    at com.github.kostyasha.yad.DockerCloud.lambda$provision$0(DockerCloud.java:135)
    at jenkins.util.ContextResettingExecutorService$2.call(ContextResettingExecutorService.java:46)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)
原因：插件 Yet Another Docker 与 aliyun registry冲突导致无法拉取镜像
解决：配置 Pull strategy 为 never，并将需要的基础镜像手动拉取到集群的每一台 ECS 上，每次生成 slave 都是从本地获取基础镜像

错误4：
Will provision 'registry.cn-hangzhou.aliyuncs.com/wenba/jenkins-slave-golang', for label: 'docker-jenkins-slave-go', in cloud: 'aliyun-docker-swarm-test'
Dec 19, 2017 8:32:32 AM INFO com.nirima.jenkins.plugins.docker.DockerCloud addProvisionedSlave
Not Provisioning 'registry.cn-hangzhou.aliyuncs.com/wenba/jenkins-slave-golang'; Server 'aliyun-docker-swarm-test' full with '100' container(s)
原因：在 Cloud Docker 中默认 Container Cap 的值是100，故无法创建新的 docker 容器
解决：将 Container Cap 修改为 1000 即可

错误5：
java.io.IOException: SSH service didn't started after 60s.
    at io.jenkins.docker.connector.DockerComputerSSHConnector.createLauncher(DockerComputerSSHConnector.java:238)
    at io.jenkins.docker.connector.DockerComputerConnector.createLauncher(DockerComputerConnector.java:101)
    at com.nirima.jenkins.plugins.docker.DockerTemplate.provisionNode(DockerTemplate.java:442)
    at com.nirima.jenkins.plugins.docker.DockerCloud$1.run(DockerCloud.java:268)
    at jenkins.util.ContextResettingExecutorService$1.run(ContextResettingExecutorService.java:28)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)
原因：使用的slave的镜像中没有启动 SSHD 服务，导致 master 无法与 slave 通过 SSH 通信
解决：使用带 SSHD 服务，并开机启动服务的镜像来启动 slave，如 registry.aliyuncs.com/acs-sample/jenkins-slave-dind-java:latest

完整的成功日志
Dec 20, 2017 2:55:54 PM INFO com.github.kostyasha.yad.DockerCloud provision
Asked to provision load: '1', for: 'docker' label

Dec 20, 2017 2:55:54 PM INFO com.github.kostyasha.yad.DockerCloud provision
Will provision 'registry.aliyuncs.com/acs-sample/jenkins-slave-dind-java:latest', for label: 'docker', in cloud: 'aliyun-docker-swarm-test'

Dec 20, 2017 2:55:54 PM INFO com.github.kostyasha.yad.DockerCloud addProvisionedSlave
Provisioning 'registry.aliyuncs.com/acs-sample/jenkins-slave-dind-java:latest' number '0' on 'aliyun-docker-swarm-test'; Total containers: '0'

Dec 20, 2017 2:55:56 PM INFO com.github.kostyasha.yad.utils.HostAndPortChecker bySshWithEveryRetryWaitFor
SSH port is open on 10.28.0.2:32776

Dec 20, 2017 2:55:56 PM INFO com.github.kostyasha.yad.launcher.DockerComputerSSHLauncher getSSHLauncher
Creating slave SSH launcher for '10.28.0.2:32776'. Cloud: 'aliyun-docker-swarm-test'. Template: 'registry.aliyuncs.com/acs-sample/jenkins-slave-dind-java:latest'

Dec 20, 2017 2:56:04 PM INFO hudson.slaves.NodeProvisioner$2 run
registry.aliyuncs.com/acs-sample/jenkins-slave-dind-java:latest provisioning successfully completed. We have now 2 computer(s)

Dec 20, 2017 2:58:56 PM INFO com.github.kostyasha.yad.DockerSlave _terminate
Requesting disconnect for computer: 'aliyun-docker-swarm-test-ca6c0b413f1e'

Dec 20, 2017 2:59:08 PM INFO com.github.kostyasha.yad.DockerSlave _terminate
Stopped container ca6c0b413f1ed66c8ff5cb0880e5075d3b41a22e1e404e55bbc731e1f48bed57

Dec 20, 2017 2:59:08 PM INFO com.github.kostyasha.yad.DockerSlave _terminate

```
Removed container ca6c0b413f1ed66c8ff5cb0880e5075d3b41a22e1e404e55bbc731e1f48bed57

Dec 20, 2017 3:01:10 PM INFO org.jenkinsci.plugins.workflow.job.WorkflowRun finish
jenkins-pipeline-test02/master #9 completed: SUCCESS
```

13.利用共享库（Shared Libraries）减少不同项目之间的Pipeline的冗余
Extending with Shared Libraries：https://jenkins.io/doc/book/pipeline/shared-libraries/

14.Pipeline 编辑器
Pipeline Editor：https://jenkins.io/doc/book/blueocean/pipeline-editor/
可用于创建和编辑 Declarative Pipeline（声明式），

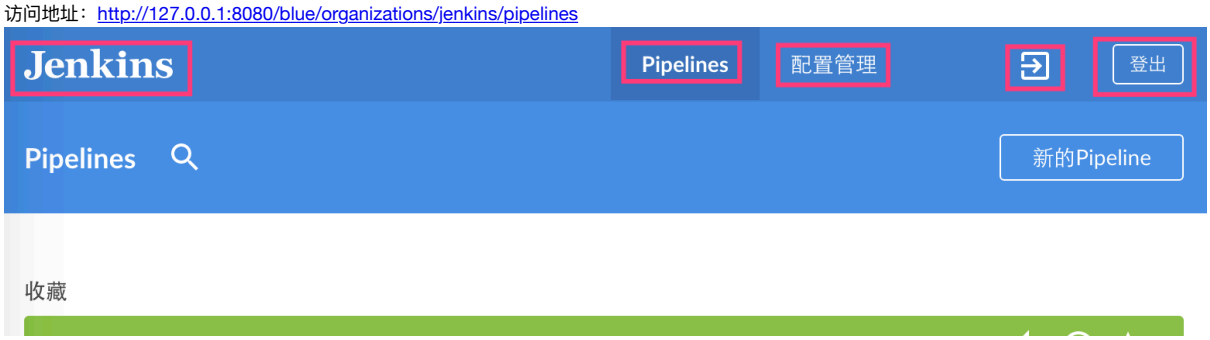
操作地址：http://127.0.0.1:8080/blue/organizations/jenkins/pipeline-editor/
流程展示：http://127.0.0.1:8080/blue/organizations/jenkins/jenkins-pipeline-test/detail/master/5/pipeline/

15.Pipeline 辅助工具
Pipeline Development Tools：https://jenkins.io/doc/book/pipeline/development/

- Blue Ocean Editor：Web 编辑器
- Command-line Pipeline Linter：命令行验证 Jenkinsfile 的有效性
- "Replay" Pipeline Runs with Modifications：重放功能，达到快速修改 Pipeline 流程的作用

16.如何修改 Blue Ocean 的 Web UI 展示界面
Blue Ocean源码：https://github.com/jenkinsci/blueocean-plugin

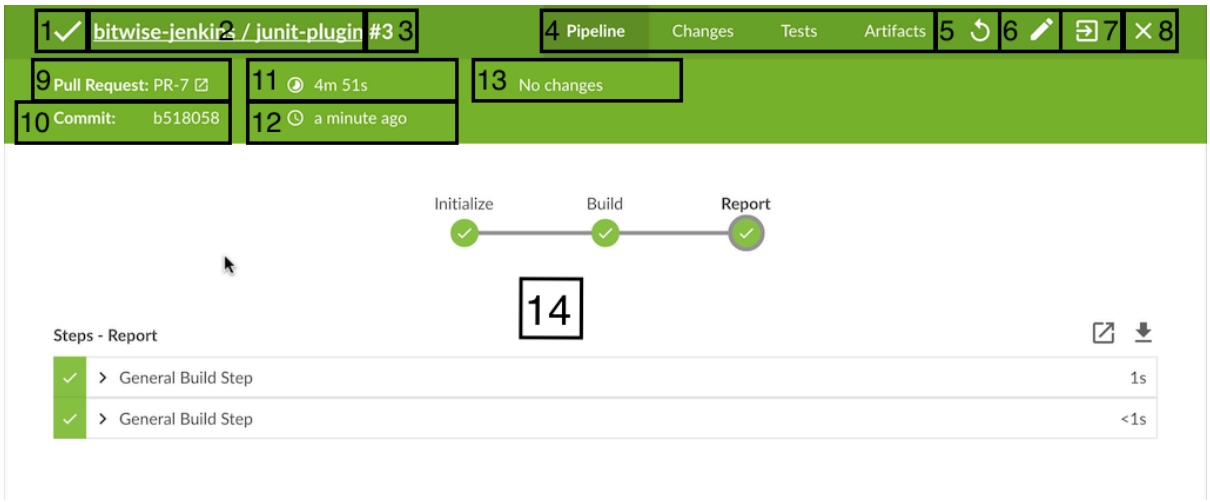
17.Blean Ocean 导航栏
访问地址：http://127.0.0.1:8080/blue/organizations/jenkins/pipelines



- Jenkins - 导航到仪表板（重新加载）
- Pipeline - 导航到仪表板（do nothing）
- 配置管理 - 导航到 Jenkins 系统管理（使用经典UI）
- 切换到“经典”用户界面 - 切换到“经典” Jenkins 用户界面
- 登出 - 注销当前用户，返回到Jenkins登录页面

18.Blue Ocean 活动视图（Activity View）
活动（Activity）：显示最近完成或正在运行的运行任务（Run），可以通过分支或拉取请求来过滤显示；
分支（Branches）：显示当前 Pipeline 中已完成或正在进行的所有分支的列表；可以运行、停止、再次运行、编辑、收藏运行任务；
Pull Requests：显示当前 Pipeline 的已完成或正在进行的运行的所有合并请求的列表；

19.Blue Ocean 选项卡



区域4是选项卡选择器（Tab Seletor），分为 Pipeline（默认显示）、Changes（改变）、Tests（测试）、Artifacts（存档工件）

- Pipeline：显示了此 Pipeline 运行流程的整体视图，显示每个阶段和并行分支，这些阶段的步骤，以及这些步骤的控制台输出；
- Changes：显示此次运行涉及的变更（如代码变更）；
- Tests：显示此 Pipeline 的测试结果；
- Artifacts：显示使用"Archive Artifacts"（archive）步骤（step）保存的任何工件的列表；完整的运行输出日志也可以在此下载；

20.Pipeline 常用step详解

文档：https://jenkins.io/doc/pipeline/steps/

代码提交：Gitlab Plugin

- acceptGitLabMR: Accept GitLab Merge Request
- addGitLabMRComment: Add comment on GitLab Merge Request
- gitlabBuilds: Notify gitlab about pending builds
- gitlabCommitStatus: Update the commit status in GitLab depending on the build status
- updateGitlabCommitStatus: Update the commit status in GitLab

编译

docker镜像推送
Artifactory Plugin：

- dockerPullStep: Artifactory docker pull
- dockerPushStep: Artifactory docker push

Docker Pipeline

- dockerFingerprintFrom: Record trace of a Docker image used in FROM
- dockerFingerprintRun: Record trace of a Docker image run in a container
- withDockerContainer: Run build steps inside a Docker container
- withDockerRegistry: Sets up Docker registry endpoint
- withDockerServer: Sets up Docker server endpoint

参考文档：https://wiki.jenkins.io/display/JENKINS/Docker+build+step+plugin
Docker Slaves Plugin

- dockerNode: Allocate a docker node

部署至aliyun_docker
Pipeline: Nodes and Processes

- node: Allocate node
- sh: Shell Script

交互输入：
Pipeline: Input Step

- input: Wait for interactive input

注1：Input 插件的parameters功能，只能在 Scripted Pipeline 中使用，而不能在 Declarative Pipeline 中使用

注2：Input 插件没有超时时间，但可以通过 timeout 设置超时时间（https://support.cloudbees.com/hc/en-us/articles/226554067-Pipeline-How-to-add-an-input-step-with-timeout-that-continues-if-timeout-is-reached-using-a-default-value）
参考：https://jenkins.io/doc/pipeline/steps/pipeline-input-step/#input-wait-for-interactive-input

21.去除一般用户的 Abort 权限：
https://support.cloudbees.com/hc/en-us/articles/115000633632-How-can-I-prevent-users-from-aborting-a-Pipeline-Input-Step-using-RBAC-

22.容错处理
Declarative Pipeline：支持 post 来处理错误

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
            }
        }
    }
    post {
        always {
            echo 'I will always say Hello again!'
        }
    }
}

# post必须放在 Declarative Pipeline的最后
```

Scripted Pipeline：只能通过 try-catch-finally 来处理（配合currentBuild.currentResult使用）

```
示例1
Jenkinsfile (Scripted Pipeline)
node {
    stage('Example') {
        try {
            sh 'exit 1'
        }
        catch (exc) {
            echo 'Something failed, I should sound the klaxons!'
            throw
        }
    }
}

示例2
#!/usr/bin/env groovy

// Jenkinsfile (Scripted Pipeline)
node('docker') {
    checkout scm

    try {
        stage('发布信息') {

        }
    } catch (exc) {
        def build_response = "服务：${SERVICE_NAME} 版本：${TAG_VERSION} 测试环境构建发布失败，需要修正后再发布！"
        def response = httpRequest ...
        println("Build_response: "+build_response)
        println("Status: "+response.status)
    } finally {
        if (currentBuild.currentResult == 'UNSTABLE') {
            echo 'I am unstable :/'
         } else if (currentBuild.currentResult == 'SUCCESS') {
            def build_response = "服务：${SERVICE_NAME} 版本：${TAG_VERSION} 测试环境构建发布成功，可以测试啦！"
            def response = httpRequest ...
            println("Build_response: "+build_response)
            println("Status: "+response.status)
        }
    }
}

注：使用 if/else if 避免出现异常时，还会执行finally中的输出成功信息，故需要限定当前构建的 currentResult
```

参考：https://jenkins.io/doc/pipeline/tour/post/

23.邮箱配置-Jenkins系统配置
Jenkins Location
Jenkins URL：http://127.0.0.1:8080/
系统管理员邮件地址：zabbix@wenba100.com （需要与认证用户一致）

Extended E-mail Notification
众所周知，Jenkins默认提供了一个邮件通知，能在构建失败、构建不稳定等状态后发送邮件。但是它本身有很多局限性，比如它的邮件通知无法

提供详细的邮件内容、无法定义发送邮件的格式、无法定义灵活的邮件接收配置等等。在这样的情况下，我们找到了Jenkins Email Extension Plugin。该插件能允许你自定义邮件通知的方方面面，比如在发送邮件时你可以自定义发送给谁，发送具体什么内容等等。

SMTP server：smtp.exmail.qq.com
Default user E-mail suffix：wenba100.com
勾选 Use SMTP Authentication
User Name：
Password：
勾选 Use SSL
SMTP port：465
Charset：UTF-8
Default Content Type：HTML（text/html）
Default Recipients：devops@wenba100.com
Default Subject：（默认邮件主题，可忽略）构建通知：$PROJECT_NAME - Build # $BUILD_NUMBER - $BUILD_STATUS!
Default Content：（默认邮件正文，可忽略）

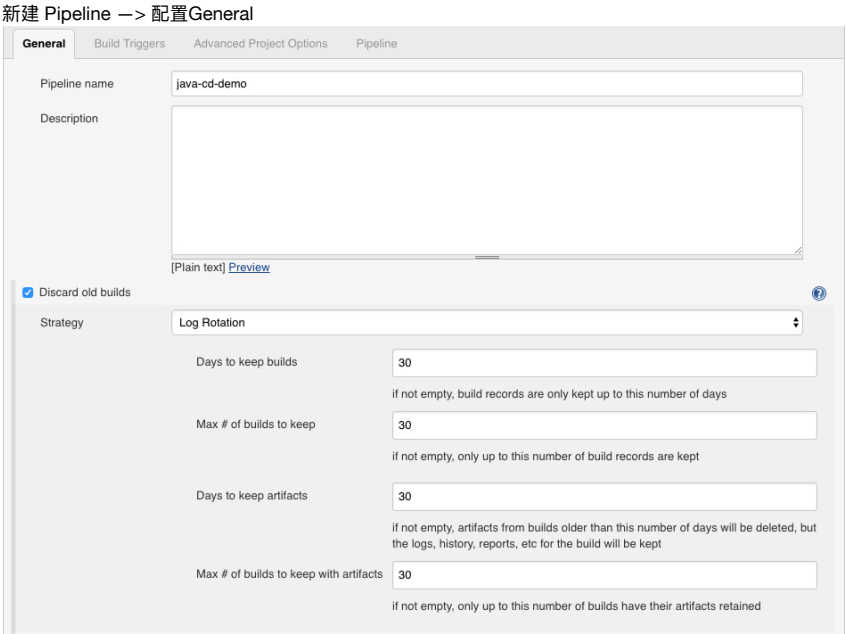
```
<hr/>
(本邮件是程序自动下发的，请勿回复！)<br/><hr/>
项目名称：$PROJECT_NAME<br/><hr/>
构建编号：$BUILD_NUMBER<br/><hr/>
git版本号：${TAG_VERSION}<br/><hr/>
构建状态：$BUILD_STATUS<br/><hr/>
触发原因：${CAUSE}<br/><hr/>
构建日志地址：<a href="${BUILD_URL}console">${BUILD_URL}console</a><br/><hr/>
构建地址：<a href="$BUILD_URL">$BUILD_URL</a><br/><hr/>
变更集:${JELLY_SCRIPT,template="html"}<br/><hr/>
```

同时，在构建过程中，发送邮件时的 From 都必须是 zabbxi@wenba100.com（与认证用户一致）

参考：Jenkins 邮件设置：http://www.cnblogs.com/zz0412/p/jenkins_jj_01.html

24.配置General，选择Discard old builds来减少存储空间占用

新建 Pipeline —> 配置General



25.Jenkins与钉钉通知

1）安装插件：HTTP Request Plugin
2）创建钉钉机器人，获得webhook：https://oapi.dingtalk.com/robot/send?access_token=xxx
3）在Jenkinsfile中配置 httpRequest 发送 json 格式的通知信息至钉钉即可，完整示例如下：

```
#!/usr/bin/env groovy
// Jenkinsfile (Scripted Pipeline)
node('docker') {
    checkout scm
    env.RUNNING_ENVIRONMENT = "docker-test"
    env.SERVICE_NAME = "aixue-activity-back"
    env.TAG_VERSION = "latest"
    env.DINGDING_WEBHOOK = "https://oapi.dingtalk.com/robot/send?access_token=fa01ebc942294adbb02067e4b9cc0a3aad5342ce10680
    try {
```

```
        stage('镜像构建') {
            sh 'chmod +x  build.sh'
            sh './build.sh'
        }
        stage('测试环境部署') {
            sh 'chmod +x deploy.py'
            sh 'python ./deploy.py'
        }
        stage('发布进行时') {
            sleep 30
        }
    } catch (exc) {
        def build_response = """
            {
                "msgtype": "text",
                "text": {
                    "content": "[测试环境]-构建通知 \n 服务：${SERVICE_NAME} \n 版本：${TAG_VERSION} \n 测试环境构建发布失败，需要修正
                },
                "at": {
                    "atMobiles": [
                        "13651694290"
                    ],
                "isAtAll": true
                }
            }"""
        def response = httpRequest acceptType: 'APPLICATION_JSON_UTF8', contentType: 'APPLICATION_JSON_UTF8', httpMode: 'POS
        println("Build_response: "+build_response)
        println("Status: "+response.status)
    } finally {
        if (currentBuild.currentResult == 'UNSTABLE') {
            echo 'I am unstable :/'
        } else if (currentBuild.currentResult == 'SUCCESS') {
            def build_response = """
                {
                    "msgtype": "text",
                    "text": {
                        "content": "[测试环境]-构建通知 \n 服务：${SERVICE_NAME} \n 版本：${TAG_VERSION} \n 测试环境构建发布成功，可
                    },
                    "at": {
                        "atMobiles": [
                            "13651694290"
                        ],
                    "isAtAll": true
                    }
                }"""
            def response = httpRequest acceptType: 'APPLICATION_JSON_UTF8', contentType: 'APPLICATION_JSON_UTF8', httpMode:
            println("Build_response: "+build_response)
            println("Status: "+response.status)
        }
    }
}
```

其中：build_response 必须是 json 格式的，且 httpRequest 的 acceptType 和 contentType HTTP Header 都设置为 APPLICATION_JSON_UTF8，才能正常显示中文字符！

参考文档：
自定义机器人：https://open-doc.dingtalk.com/docs/doc.htm?treeId=257&articleId=105735&docType=1
HTTP Request Plugin：https://github.com/jenkinsci/http-request-plugin | https://jenkins.io/doc/pipeline/steps/http_request/

钉钉自定义机器人

- 获取到Webhook地址后，用户可以使用任何方式向这个地址发起HTTP POST 请求，即可实现给该群组发送消息。注意，发起POST请求时，必须将字符集编码设置成UTF-8。
- 当前自定义机器人支持文本（text）、连接（link）、markdown（markdown）三种消息类型，大家可以根据自己的使用场景选择合适的消息类型，达到最好的展示样式。具体的消息类型参考下一节内容。
- 自定义机器人发送消息时，可以通过手机号码指定“被@人列表”。在“被@人列表”里面的人员，在收到该消息时，会有@消息提醒（免打扰会话仍然通知提醒，首屏出现“有人@你”）

26.

30.报错处理
报错1：groovy.lang.MissingPropertyException: No such property: test for class: groovy.lang.Binding
原因：缺少 test 这个类，其实变量赋值应该用引号来标识字符串的
解决方案：env.RUNNING_ENVIRONMENT = "test"

报错2：Scripts not permitted to use staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods getAt java.lang.Object java.lang.String
原因：Scripted Pipeline 中无法使用静态方法(如列表的 GetAt 方法)，如 def deployEnv = input(id: 'deployEnv',
解决方案：env.RUNNING_ENVIRONMENT = input(id: 'deployEnv',