



基于Jenkins Pipeline设计交付 流水线

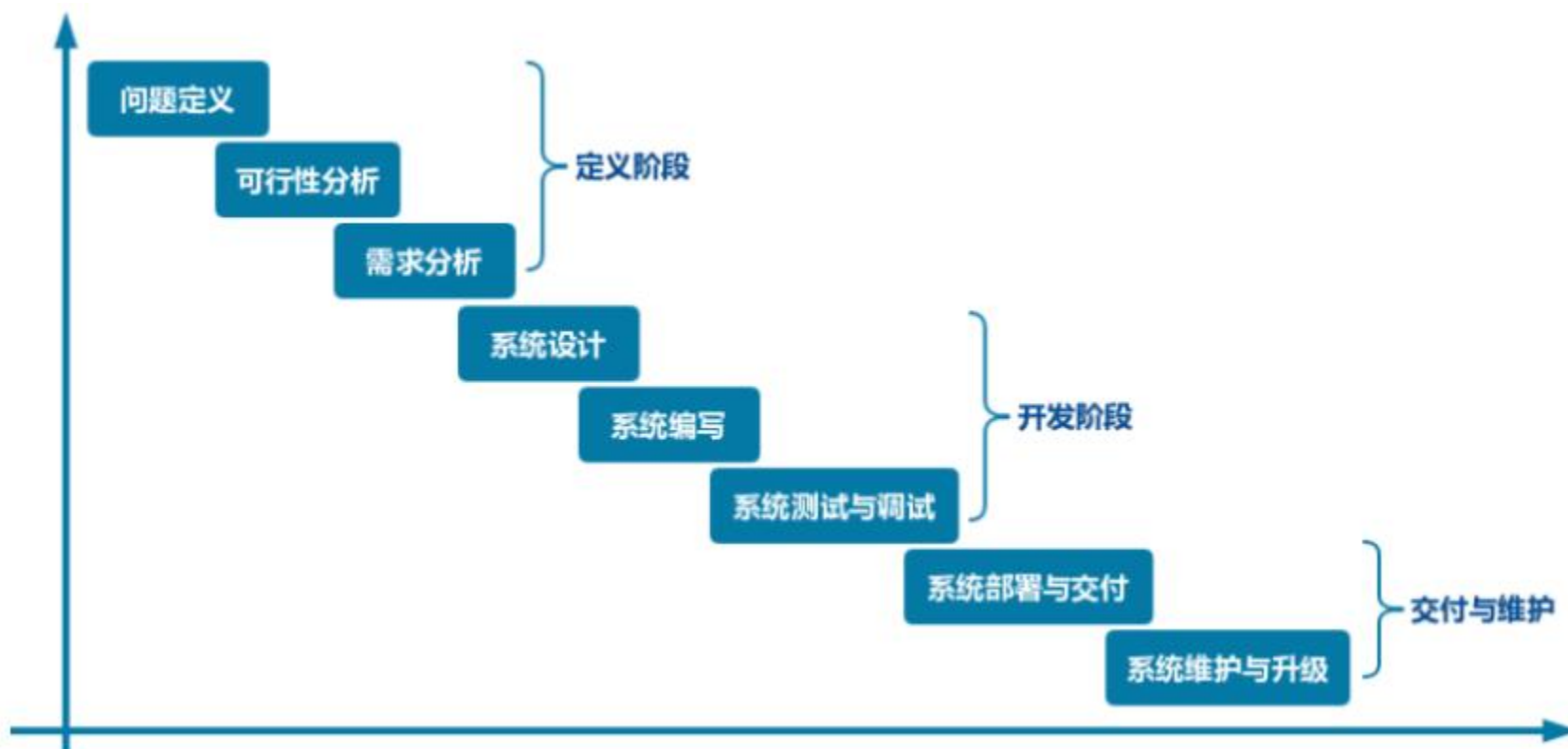
蒋刚毅 (Cay) @微医

目录

1. 流水线驱动CD/DevOps
2. Jenkins Pipeline关键特性
3. Jenkins Pipeline分阶段实践
4. Jenkins Pipeline高可用设计
5. 从交付流水线到研发协同平台

Part 1 流水线驱动CD/DevOps

传统瀑布流程



专职测试团队的困惑

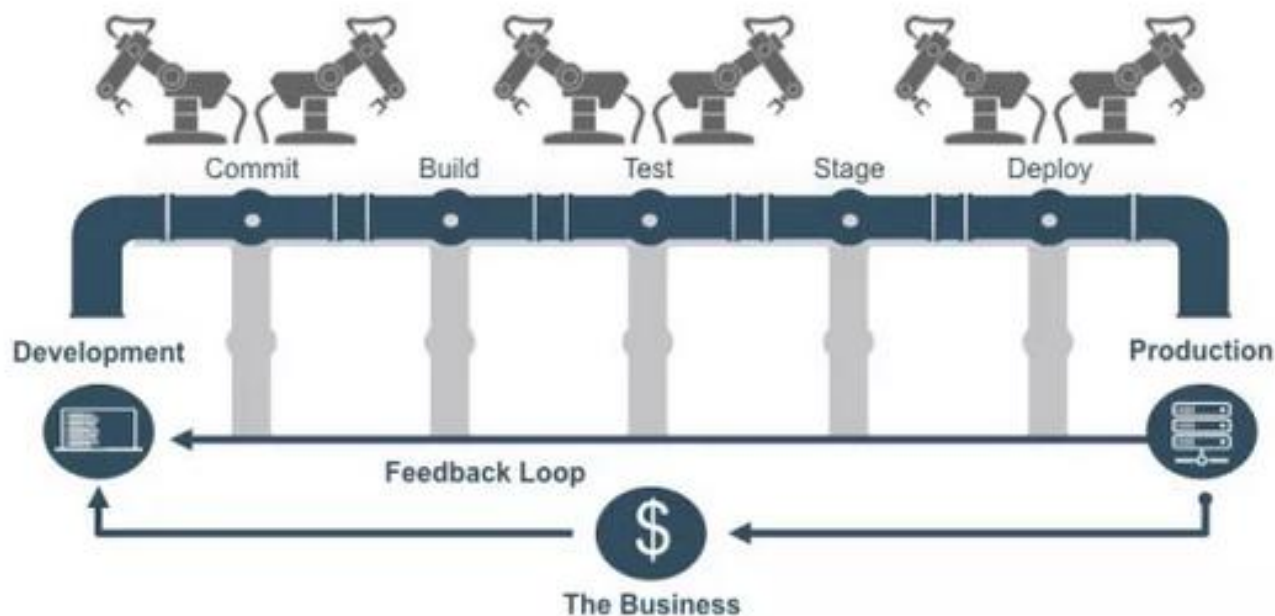


我们费劲心血建立并引以为豪的测试体系。

- 无法为其他技术团队所理解？
- 无法适应快速的迭代节奏？
- 线上仍然还是会出现各种各样的故障....
- 很多开发的测试平台变成了门面过程，用不用看测试工程师心情？
- 测试变成了背锅侠，测试真的只是测试团队的事情？
- 如何让测试变成整个研发团队的事情而不是只有测试团队自己跳舞？



持续交付的流水线是答案



DevOps模式对测试提出的新要求

- 测试左移，在开发阶段保障编码质量，定义测试方法和检查点。
- 测试右移，在生产环境保障发布质量，参与和实施线上监控，验证可靠性。

流水线提升效率



Time to Assemble a Model-T

12.5 hrs

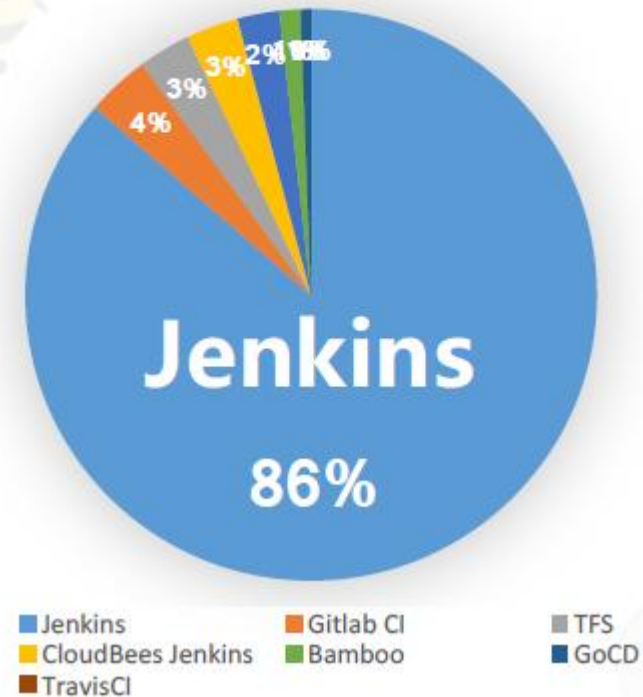


1.5 hrs

Why Jenkins?

- **65%**以上公司实现了一周一次以上的部署，微服务的时代快速交付成为常态。
- **64%**的公司已经引入持续交付流水线，其中**86%**都在使用Jenkins。
- 包括**1350**个Jenkins插件在内的活跃开源社区，完善的工具生态。
- Pipeline+BlueOcean=Future

持续交付流水线工具



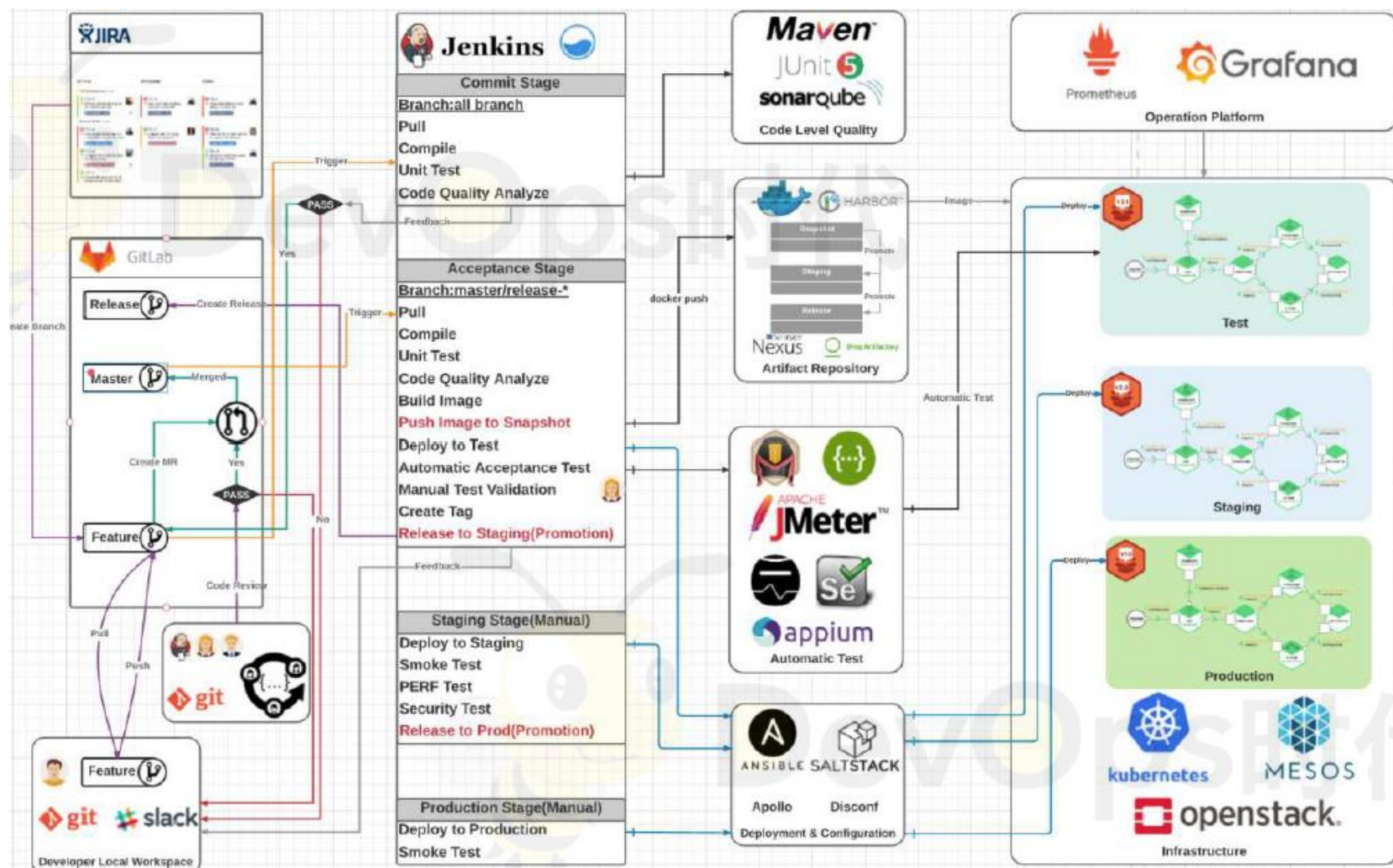
Why Pipeline?

- Pipeline as Code: 任何流程都可以表述为一段Jenkinsfile脚本, 并且Jenkins支持从代码库直接读取脚本。
- 以前用N个freestyle的job串联的工作流, 现在只需要用1个pipeline即可实现。
- 可通过groovy脚本无限扩展pipeline的能力。
- 可通过共享库方式抽象公共逻辑。
- 支持大量devops开源工具链集成。



From KK: Why, What, and How of Continuous Delivery

交付流水线工具链



Part 2 Jenkins Pipeline关键特性

Docker方式部署Jenkins

1. 下载Jenkins的docker镜像:

```
docker pull hub.c.163.com/library/jenkins:latest
```

2. 启动Jenkins镜像:

```
docker run -itd -u root --name jenkins -p 8080:8080 -p 50000:50000 -v /var/jenkins_home:/var/jenkins_home hub.c.163.com/library/jenkins
```

3. 进入Jenkins容器

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

4. 访问Jenkins服务:

```
http://IP:8080/jenkins
```

Jenkins Pipeline定义

- 使用条件

- ✓ Jenkins 2.x或更高版本
- ✓ Pipeline插件

- Pipeline定义

- ✓ 任何Pipeline交付流程都可以表述为一段Groovy脚本，并且Jenkins支持从代码库直接读取脚本，从而实现了Pipeline as Code的理念。
- ✓ Pipeline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发布流程。

Pipeline DSL Syntax

- **Declarative Pipeline**

对用户来说，语法更严格，有固定的组织结构，更容易生成代码段，同时可支持BlueOcen图形化脚本操作，使其成为用户更理想的选择。

- **Scripted Pipeline**

更加灵活，因为Groovy本身只能对结构和语法进行限制，对于更复杂的Pipeline来说，用户可以根据自己的业务进行灵活的实现和扩展。

Jenkinsfile (Declarative Pipeline)

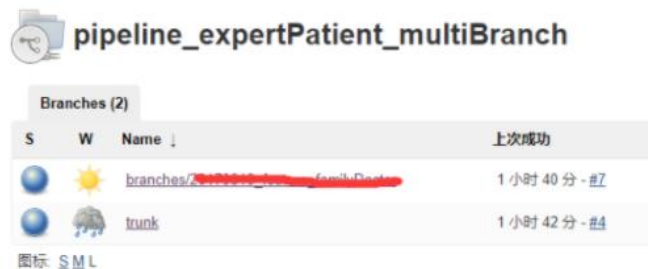
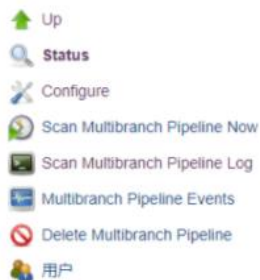
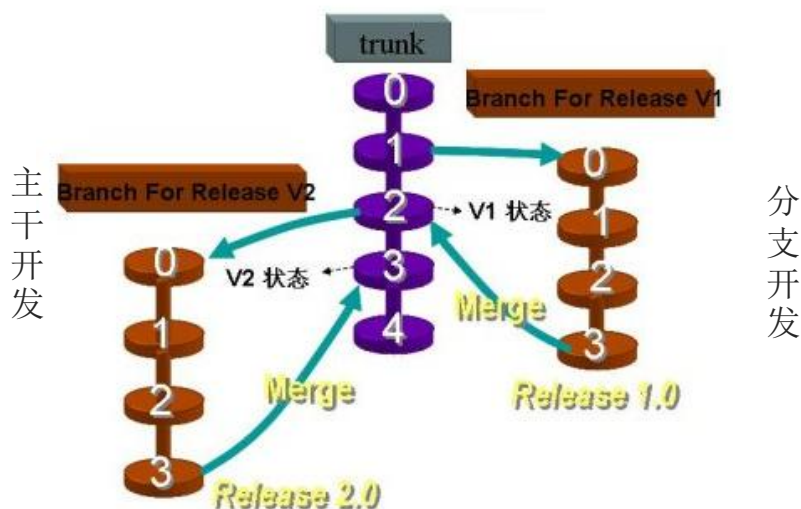
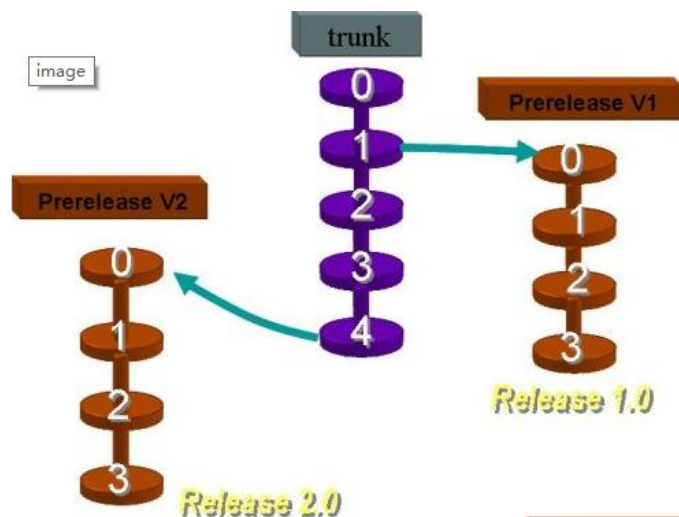
```
pipeline {  
    agent any ❶  
  
    stages {  
        stage('Build') { ❷  
            steps { ❸  
                sh 'make' ❹  
            }  
        }  
        stage('Test'){  
            steps {  
                sh 'make check'  
                junit 'reports/**/*.xml' ❺  
            }  
        }  
        stage('Deploy') {  
            steps {  
                sh 'make publish'  
            }  
        }  
    }  
}
```

Jenkinsfile (Scripted Pipeline)

```
node {  
    stage('Example') {  
        if (env.BRANCH_NAME == 'master') {  
            echo 'I only execute on the master branch'  
        } else {  
            echo 'I execute elsewhere'  
        }  
    }  
}
```

MultiBranch Pipeline

- 主干开发模式（或者分支少），推荐采用普通Pipeline模式
- 多分支开发模式，推荐采用MultiBranch Pipeline模式。



Shared Library

- 共享库的方式抽象了各种项目之间共享的代码（甚至整条完整的pipeline），有效降低了业务工程师使用pipeline脚本的复杂度。
- 同时通过外部源代码控制（SCM）的方式，可以保证最新提供的库代码功能可被所有pipeline项目即时使用，在大团队推广和协作过程中可起到非常重要的作用。

```
#!groovy
library 'weiyi-pipeline-library' //调用一个名为weiyi-pipeline-library的library，目前只有weiyi-pipeline-library这个库，所以此处照抄就行
def map = [:] //定义一个名叫map的hash表，下面的所有map.put都是把运行相关参数加入到这个hash表数据结构中

/*参数化变量,运行时可选择*/
map.put('repoBranch','master') //git分支名称
map.put('server','192.168.1.100') //测试服务器列表选择,非必须,根据项目,务必填入完整绝对路径,autoconfig项目
map.put('dubboPort','20880') //测试服务器的dubbo服务端口
map.put('lineCoverage','0') //单元测试代码覆盖率要求,各项目视要求调整参数

/*环境变量,初始确定后一般不需更改*/
map.put('maven','maven3') //maven版本,manvn项目需要该参数,该参数非必须
// map.put('gradle','Gradle2.14.1') //gradle版本,gradle项目需要该参数,该参数非必须
map.put('jdk','jdk8') //jdk版本

/*常量参数,初始确定后一般不需更改*/
map.put('DOCKER_NAME','gconfig') //docker容器名称,仅在选择docker部署需要,该参数非必须
map.put('DOCKER_IMAGE','jdk8') //基础镜像,仅在选择docker部署需要,该参数非必须
map.put('REPO_URL','git@github.com:weiyi-pipeline-library.git') //项目gitlab代码地址
map.put('CRED_ID','123456789') //git服务全系统只读账号,通常情况下该值无需修改
map.put('POM_PATH','pom.xml') //pom.xml的相对路径
map.put('WAR_PATH','gconfig-service/gconfig-service-war/target/*.war') //生成war包的相对路径
map.put('TEMPLATES_PATH','gconfig-gops/gconfig-gops-deploy/templates/') //template文件夹目录,该参数非必须
map.put('QA_EMAIL','test@weiyi.com') //测试人员邮箱地址
map.put('ITEST_JOBNAME','Guahao_InterfaceTest_kano') //接口测试job名称
// map.put('JETTY_VERSION','8') //jetty 版本,默认为8,该参数非必须
// map.put('SHELL_EXEC_BEFORE_PACKAGE','xxx/xxx/xxx.sh') //某些应用在编译打包之前,需要执行一些操作比如替换微信公众号及定时任务相关配置,包含这些操作的shell文件路径在此处传入,该参数非必须

pipelineCall('maven',map) //调用pipelineCall这个方法,这个方法是在weiyi-pipeline-library库中定义的,传入项目构建类型(目前支持maven和gradle两种方式)及前几步定义好的hash表
```


Pipeline work in parallel

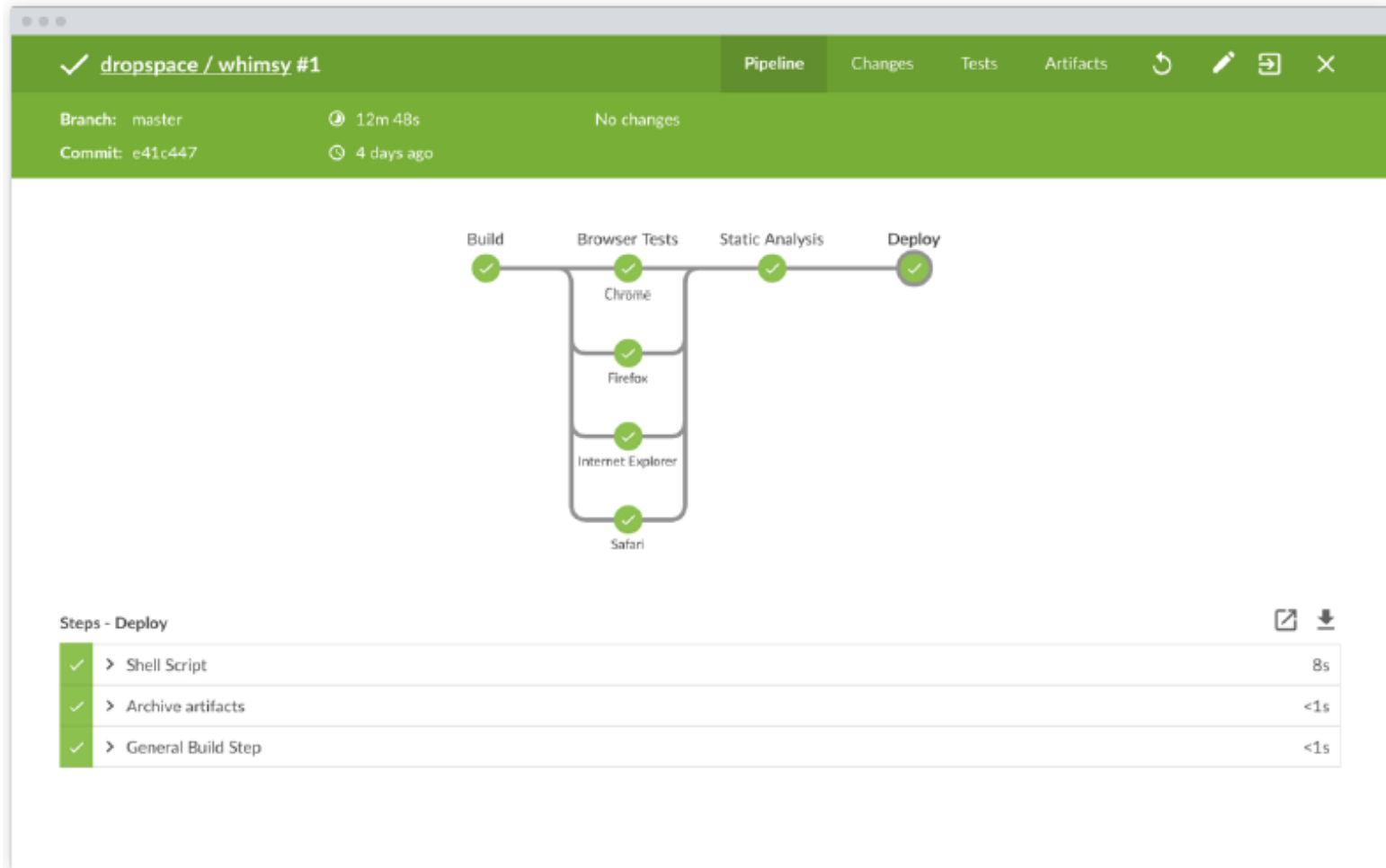
```
//Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Non-Parallel Stage') {
            steps {
                echo 'This stage will be executed first.'
            }
        }
        stage('Parallel Stage') {
            when {
                branch 'master'
            }
            failFast true
            parallel {
                stage('Branch A') {
                    agent {
                        label "for-branch-a"
                    }
                    steps {
                        echo "On Branch A"
                    }
                }
                stage('Branch B') {
                    agent {
                        label "for-branch-b"
                    }
                    steps {
                        echo "On Branch B"
                    }
                }
            }
        }
    }
}
```

对相互不存在依赖的stage可以通过parallel的方式执行，以提升pipeline的运行效率。



Blue Ocean

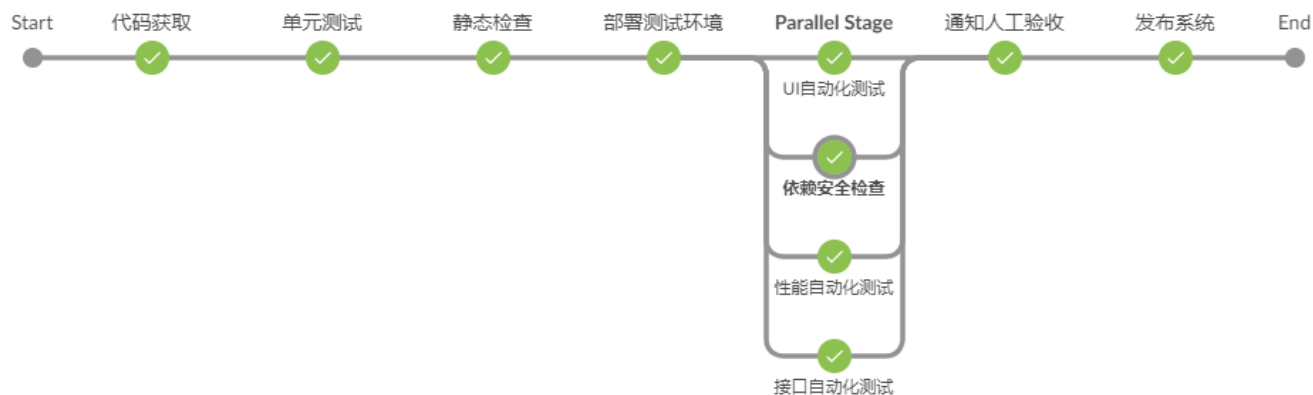
未来是否可以替换目前的Jenkins界面？



Part 3

Jenkins Pipeline分阶段实践

持续交付流水线样例



Steps 依赖安全检查



✓	> maven3 — Use a tool from a predefined Tool Installation	<1s
✓	> Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step.	<1s
✓	> jdk8 — Use a tool from a predefined Tool Installation	<1s
✓	> Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step.	<1s
✓	> Invoke OWASP Dependency-Check analysis	39s

持续交付流水线样例

▶ Run

状态	运行	提交	消息
✓	524	-	add -Da
✓	523	-	Started b
✓	522	-	Started b
✓	521	-	Started b
✓	520	-	Started b
✓	519	-	Started b
✓	518	-	Started b
✗	517	-	Started b
✗	516	-	modify p
✓	515	-	Started by user 薛小冬

需要输入

场景选择, 默认运行完整流水线, 如果只做开发自测可选择代码检查, 如果只做环境部署可选择测试部署

☒ scene1:完整流水线

☐ scene2:代码检查

☐ scene3:测试部署

git分支名称

release_expert-patient_2

测试服务器列表选择
(IP,JettyPort,Name,Passwd,autoconfigPath)

☒ 192.168.1.107,9090,

☐ 192.168.1.60,9090,

单元测试代码覆盖率要求(%), 小于此值pipeline将会失败!

20

运行

取消

持续时间	完成
22s	6 hours ago
3m 26s	3 days ago
36s	3 days ago
35s	3 days ago
5m 9s	6 days ago
3m 34s	6 days ago
21s	6 days ago
2m 26s	7 days ago
2m 57s	7 days ago
37s	7 days ago

GitLab 自动触发



GitLab

Settings

Webhooks

Webhooks can be used for binding events when something is happening within the project.

Add new webhook

URL

`http://103.10.86.5:8080/jenkins/project/`

Secret Token

Use this token to validate received payloads

Trigger

- ☒ Push events
This url will be triggered by a push to the repository
- ☐ Tag push events
This url will be triggered when a new tag is pushed to the repository
- ☐ Comments
This url will be triggered when someone adds a comment
- ☐ Issues events
This url will be triggered when an issue is created/updated/merged
- ☐ Merge Request events
This url will be triggered when a merge request is created/updated/merged
- ☐ Build events
This url will be triggered when the build status changes

SSL verification

- ☒ Enable SSL verification

Add Webhook

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 CI Service URL: `http://103.10.86.5:8080/jenkins/project/`

Enabled GitLab triggers

Push Events	<input checked="" type="checkbox"/>
Merge Request Events	<input checked="" type="checkbox"/>
Rebuild open Merge Requests	Never
Comments	<input checked="" type="checkbox"/>
Comment for triggering a build	Jenkins please retry a build

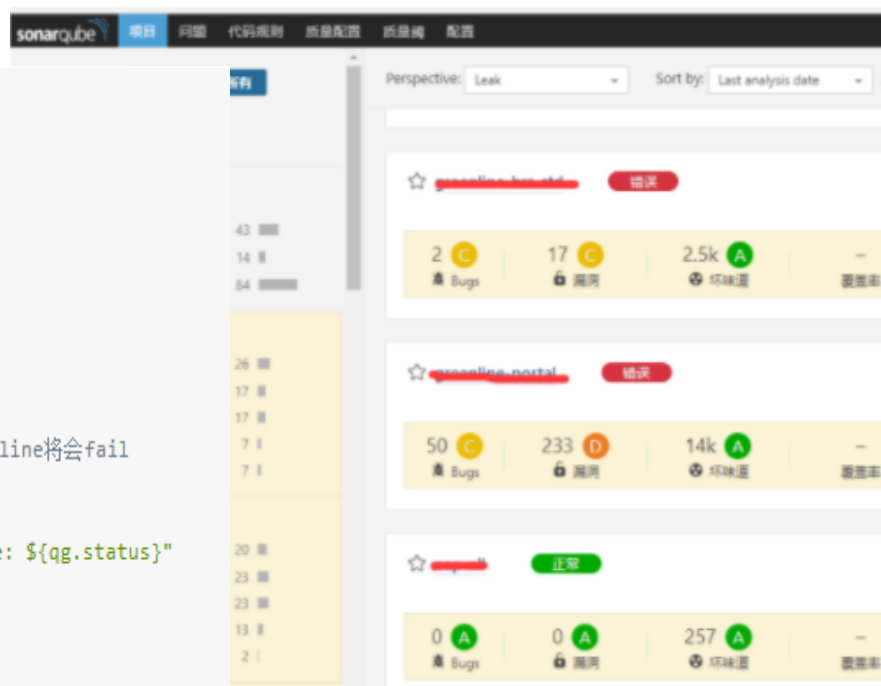
☐ Poll SCM

代码静态检查

Sonar6支持webhook功能，可与Jenkins等持续集成平台完美对接，在检查完后通过webhook的方式实时反馈检查结果，控制pipeline状态。

Sonar质量阈未通过时中止pipeline

```
stage('静态检查') {  
    steps {  
        echo "starting codeAnalyze with SonarQube....."  
        //sonar:sonar.QualityGate should pass  
        withSonarQubeEnv('SonarQube') {  
            //固定使用项目根目录${basedir}下的pom.xml进行代码检查  
            sh "mvn -f pom.xml clean compile sonar:sonar"  
        }  
        script {  
            timeout(10) {  
                //利用 sonar webhook功能通知pipeline代码检测结果，未通过质量阈，pipeline将会fail  
                def qg = waitForQualityGate()  
                if (qg.status != 'OK') {  
                    error "未通过Sonarqube的代码质量阈检查，请及时修改! failure: ${qg.status}"  
                }  
            }  
        }  
    }  
}
```



单元测试和覆盖率统计






















单测代码覆盖率低于70%时中止pipeline

```
stage('单元测试') {  
  steps {  
    echo "starting unitTest....."  
    //clean test. All tests should pass.  
    sh "mvn org.jacoco:jacoco-maven-plugin:prepare-agent -f pom.xml clean test -Dautoconfig.skip=true -Dmaven.test.skip=false -Dmaven.test.failure.ignore=true"  
    junit '**/target/surefire-reports/*.xml'  
    //code coverage.LineCoverage>70%.  
    jacoco changeBuildStatus: true, maximumLineCoverage:70  
  }  
}
```

Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all classes	66%  M: 8001 C: 15818	47%  M: 728 C: 651	60%  M: 1021 C: 1555	67%  M: 2010 C: 4061	70%  M: 568 C: 1315	75%  M: 45 C: 134

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
	M: 35 C: 113 76% 	M: 4 C: 10 71% 	M: 5 C: 9 64% 	M: 11 C: 20 65% 	M: 3 C: 4 57% 	M: 1 C: 1 50% 
	M: 10 C: 15 60% 	M: 0 C: 0 100% 	M: 2 C: 2 50% 	M: 2 C: 5 71% 	M: 2 C: 2 50% 	M: 0 C: 1 100% 
	M: 243 C: 651 73% 	M: 30 C: 64 68% 	M: 31 C: 31 50% 	M: 63 C: 147 70% 	M: 4 C: 11 73% 	M: 0 C: 1 100% 

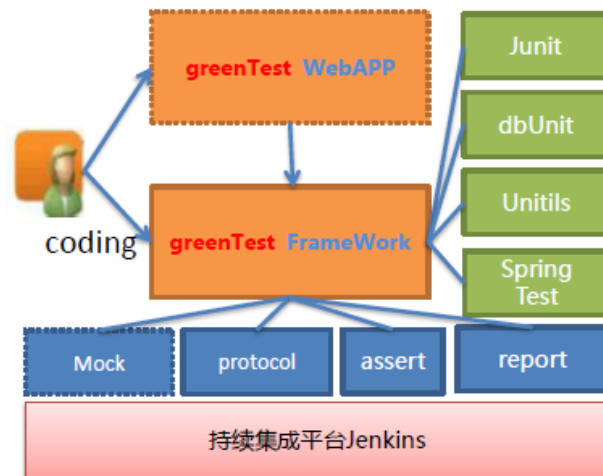
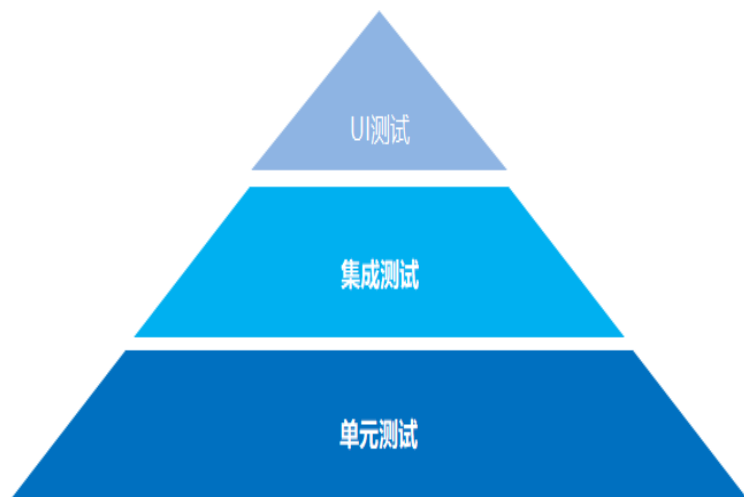
自动部署和发布

使用共享库屏蔽部署脚本复杂性，适配虚拟机/docker等不同环境。

```
stage('部署测试环境') {
    when {
        expression {
            return isDP
        }
    }
    steps {
        script {
            if ("${SHELL_EXEC_BEFORE_PACKAGE}" != "null") {
                sh "sh ${SHELL_EXEC_BEFORE_PACKAGE}"
            }
            sh "sshpass -p ${serverPasswd} ssh ${serverName}@${serverIP} rm -rvf htdocs/war/*.war"
            sh "sshpass -p ${serverPasswd} ssh ${serverName}@${serverIP} rm -rvf htdocs/templates/*"
            echo "starting deploy to ${serverIP}....."
            if ("${autoconfig_path}" != "null") {
                sh "mvn -f ${POM_PATH} clean package -Dmaven.test.skip=true -Dmaven.javadoc.skip=true -Dautoconfig.userProper"
            } else {
                sh "mvn -f ${POM_PATH} clean package -Dautoconfig.skip=true -Dmaven.test.skip=true -Dmaven.javadoc.skip=true"
            }
            archiveArtifacts WAR_PATH
            //deploy war
            sh "sshpass -p ${serverPasswd} scp ${WAR_PATH} ${serverName}@${serverIP}:htdocs/war"
            try {
                sh "sshpass -p ${serverPasswd} scp -r ${TEMPLATES_PATH} ${serverName}@${serverIP}:htdocs/"
                echo "templates copy successful"
            } catch (e) {
                echo "templates copy unsuccessful"
            }
            Date date = new Date()
            def deploylog = "${date.toString()},${user} use ${JOB_NAME}_${BUILD_NUMBER} deploy branch ${params.repoBranch} to"
            println deploylog

            // 追加部署日志到远程服务器，方便后续查询
            sh "sshpass -p ${serverPasswd} ssh ${serverName}@${serverIP} 'echo ${deploylog} >>htdocs/war/deploy.log'"
            // 从远程服务器复制，存档部署日志。
            sh "sshpass -p ${serverPasswd} scp -r ${serverName}@${serverIP}:htdocs/war/deploy.log ${WORKSPACE}"
            if ("${DOCKER_NAME}" == "null") {
                if ("${JETTY_VERSION}" == "9") {
                    sh "sshpass -p ${serverPasswd} ssh ${serverName}@${serverIP} 'bin/jettyrestart9.sh'"
                } else {
                    sh "sshpass -p ${serverPasswd} ssh ${serverName}@${serverIP} 'bin/jettyrestart.sh'"
                }
            } else {
                sh "sshpass -p ${serverPasswd} ssh ${serverName}@${serverIP} './execution.sh ${jettyPort} ${DOCKER_NAME} ${DOC"
            }
            archiveArtifacts 'deploy.log'
        }
    }
}
```

UI/接口自动化测试



```
stage('接口自动化测试') {
    steps{
        echo "starting interfaceTest....."
        script {
            //为确保jetty启动完成，加了一个判断，确保jetty服务器启动可以访问后再执行接口层测试。
            timeout(5) {
                waitUntil {
                    try {
                        //确保jetty服务的端口启动成功
                        sh "nc -z ${serverIP} ${jettyPort}"
                        //sh "wget -q http://${serverIP}:${jettyPort} -O /dev/null"
                        return true
                    } catch (exception) {
                        return false
                    }
                }
            }
        }
        //将参数IP和Port传入到接口测试的job，需要确保接口测试的job参数可注入
        build job: ITEST_JOBNAME, parameters: [string(name: "dubbourl", value: "${serverIP}:${params.dubboPort}")]
    }
}
```

安全自动化测试

- 代码安全检查
- 依赖组件安全检查
- 安全渗透扫描

Dependency-Check Results

Vulnerability Trend

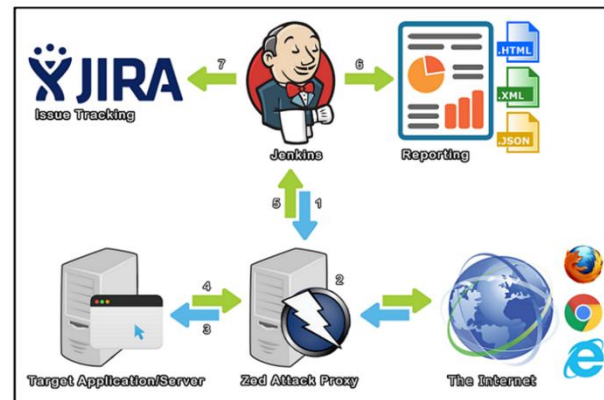
All Vulnerabilities	New Vulnerabilities
415	6

Summary

Total	High Priority	Normal Priority
415	17	280

Details

Source Folder	Total	Distribution
SNAPSHOT\WEB-INF\lib	409	
SNAPSHOT\WEB-INF\lib\ehcache-2.9.1.jar\rest-management-private-classpath\META-INF\maven\org.eclipse.jetty\jetty-continuation	1	



```
stage('依赖安全检查') {
    when {
        expression {
            return isDC
        }
    }
    steps {
        //指定检测**/lib/*.jar的组件
        dependencyCheckAnalyzer datadir: '', hintsFile: '', includeCsvReports: false, includeHtmlReports: false, includeJsonReports: false, isAutoupdateDisabled: false, outdir: '', scanpath: '**/lib/*.jar', skipOnScmChange: false, skipOnUpstreamChange: false, suppressionFile: '', zipExtensions: ''
        //有高级别组件漏洞时，fail掉pipeline
        dependencyCheckPublisher canComputeNew: false, defaultEncoding: '', failedTotalHigh: '0', healthy: '', pattern: '', unhealthy: ''
    }
}
```

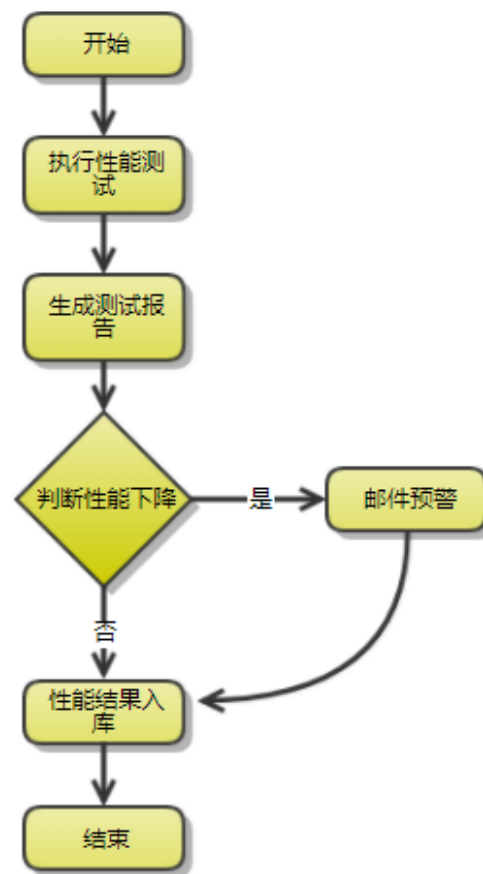
更多自动化环节的集成

- APP自动打包分发
- APP专项测试
- 性能自动化压测
- 分级部署和发布
-

[rch-autotest](#) | 1512583200951 |

Test Result

		TPS	Response Time(avg)	Response Time(90%)	错误率	备注
Total	历史平均	1007.12	41.15 ms	119.13 ms		
	本次结果	1173.57	36.55 ms	76.0 ms	0.0%	
hospital_search	历史平均	1165.37	30.75 ms	48.00 ms		
	本次结果	1248.44	31.22 ms	51.0 ms	0.0%	
tag_service	历史平均	2606.74	18.11 ms	28.75 ms		TPS下降 : 0.32
	本次结果	1762.43	27.39 ms	32.0 ms	0.0%	



核心接口压测自动化

交付过程度量

- 收集交付过程中产生的数据：如代码检查数据，研发Pipeline数据、发布Pipeline数据、线上故障数据等，可有效分析和改进团队质效问题。

WCP

应用管理

资源管理

流水线管理

发布管理

监控管理

质效管理

质效看板

质效数据

团队：

全部

应用：

全部

查询

场景			开发阶段				测试阶段		发布阶段		运营阶段
编号	应用	团队	阻断违规数	严重违规数	代码重复率(%)	单元测试覆盖率(%)	Pipeline执行数	Pipeline成功率(%)	发布执行数	发布成功率(%)	外部缺陷数
1	wcp	质量中心									
2	guser	应用平台									

Part 4

交付流水线的高可用设计

Jenkins性能调优实践

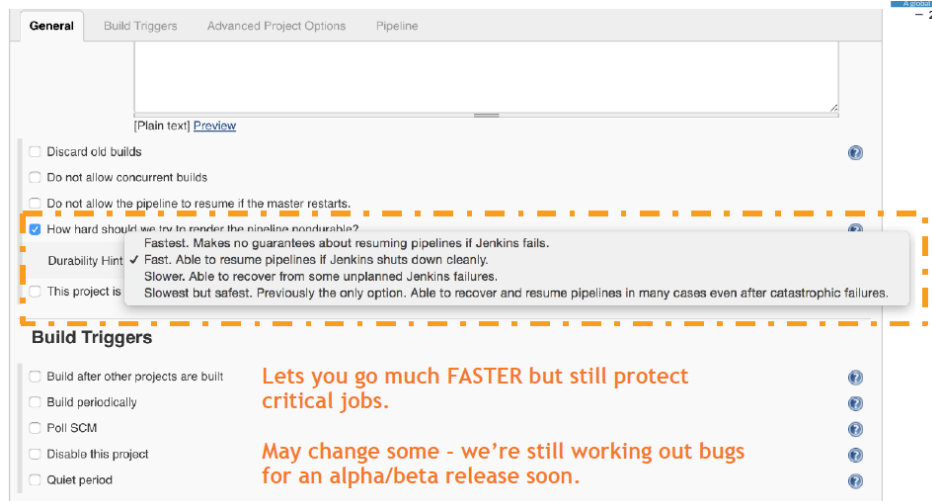
- 使用Pipeline方式比配置方式运行更快。
- 一个代码库，一条流水线。
- 让Pipeline做中间组织的工作，而不是取代其他工具。
- 能用脚本实现的，就不要用插件。
- 根据资源情况限制并发执行的job数量。
- 使用共享库抽象公共的代码并持续优化。
- 不要写太复杂的Pipeline脚本（>1000行），包括共享库代码在内。
- 不要对外网环境有强依赖（万恶的墙）。
- 使用“@NonCPS”注解高耗代码方法。
- 使用SSD硬盘等高性能硬件。



API CLIENTS

BUILD TOOLS

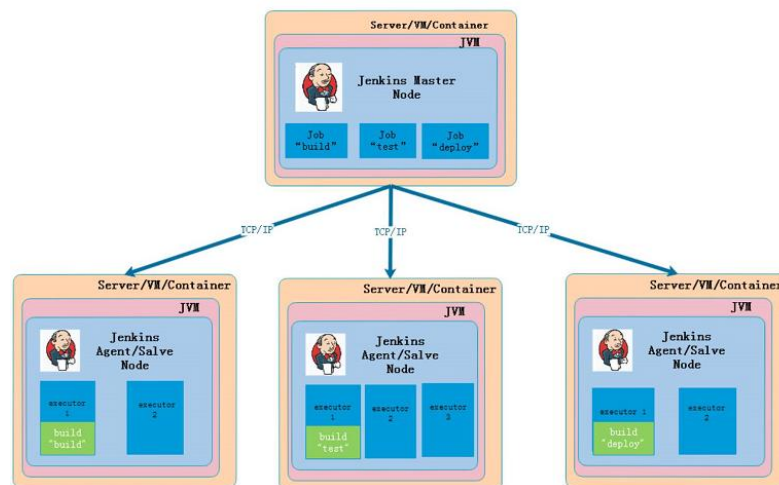
COMING SOON:
Durability/Speed Options



Jenkins构建集群

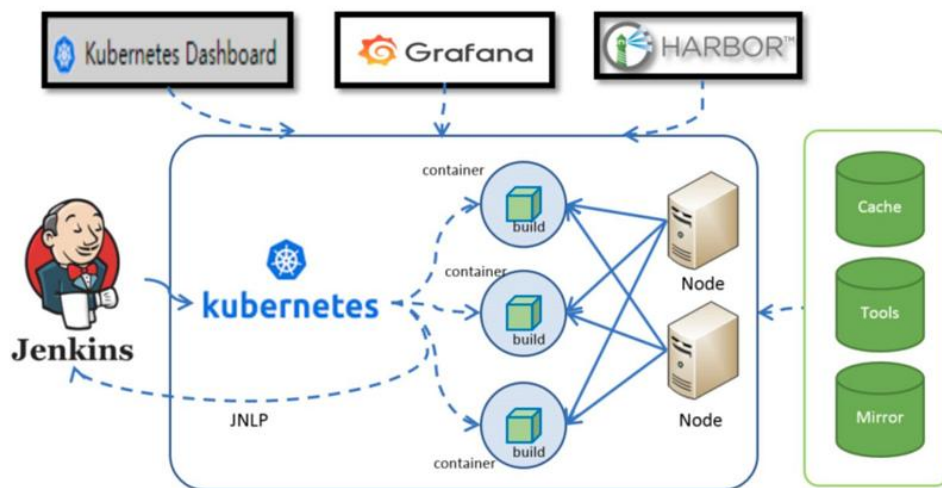
- Jenkins Master-Slave集群

- Slave编译环境不统一
- Slave资源利用不均衡
- Slave的workspace空间浪费
- 节点稳定性无法保证



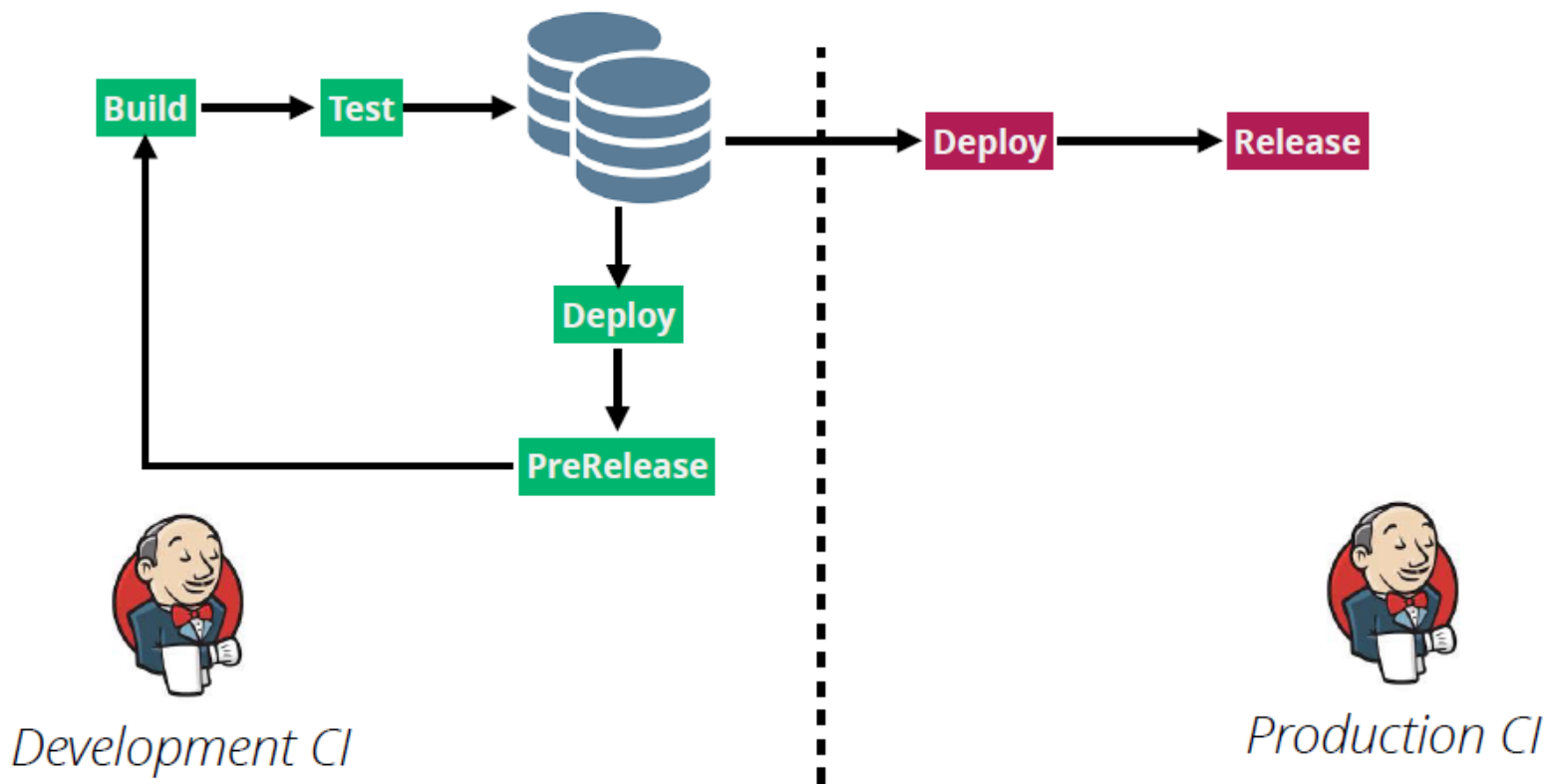
- 基于K8s的统一调度集群

- 使用Docker image标准化Jenkins环境
- 创建pod 挂载容器 slave
- 容器Slave按需弹性收缩



研发CI和生产CI分离

- 大团队协作时，有了持续交付流水线，并不代表一定要把发布环节也放到研发CI里去。
- 发布需要有一套独立的CI，让研发CI不断在自己的测试环境上去做测试，最后由发布人员从生产CI去取得发布镜像进行发布。

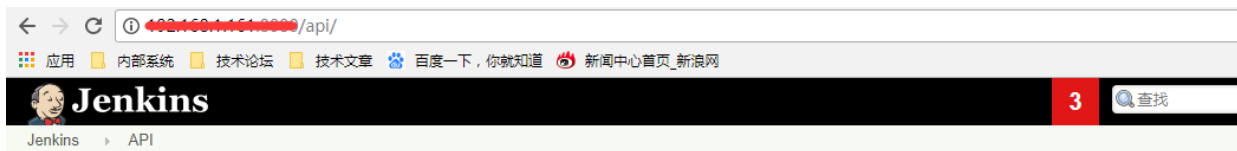


Part 5

从交付流水线到研发协同平台

从交付流水线到研发协同平台

Jenkins Pipeline有效的解决了我们对代码交付的流水线需求，但代码交付并不是研发工作的全部。



REST API

Many objects of Jenkins provide the remote access API. They are available at `.../api/` where `...` portion is the object for which you'd like to access.

XML API

Access data exposed in [HTML](#) as XML for machine consumption. [Schema](#) is also available.

You can also specify optional XPath to control the fragment you'd like to obtain (but see [below](#)). For example, `.../api/xml?xpath=/**/*[0]`.

For XPath that matches multiple nodes, you need to also specify the "wrapper" query parameter to specify the name of the root XML element to be create so that the resulting

Similarly `exclude` query parameter can be used to exclude nodes that match the given XPath from the result. This is useful for trimming down the amount of data you fetch (but be specified multiple times).

XPath filtering is powerful, and you can have it only return a very small data, but note that the server still has to build a full DOM of the raw data, which could cause a large memory consideration using the `tree` parameter, or use the `xpath` parameter in conjunction with the `tree` parameter. When used together, the result of the `tree` parameter filtering is built in the final return value. In this way, you can often substantially reduce the size of DOM built in memory.

JSON API

Access the same data as JSON for JavaScript-based access. `tree` may be used.

Python API

Access the same data as Python for Python clients. This can be parsed into Python object as `eval(urllib.urlopen(...).read())` and the resulting object tree is identical to the XML output. Beware of the security implication. If you are connecting to a non-trusted Jenkins, the server can send you malicious Python programs.

In Python 2.6 or later you can safely parse this output using `ast.literal_eval(urllib.urlopen(...).read())`

For more information about remote API in Jenkins, see [the documentation](#).

Controlling the amount of data you fetch

Jenkins Pipeline的扩展能力:

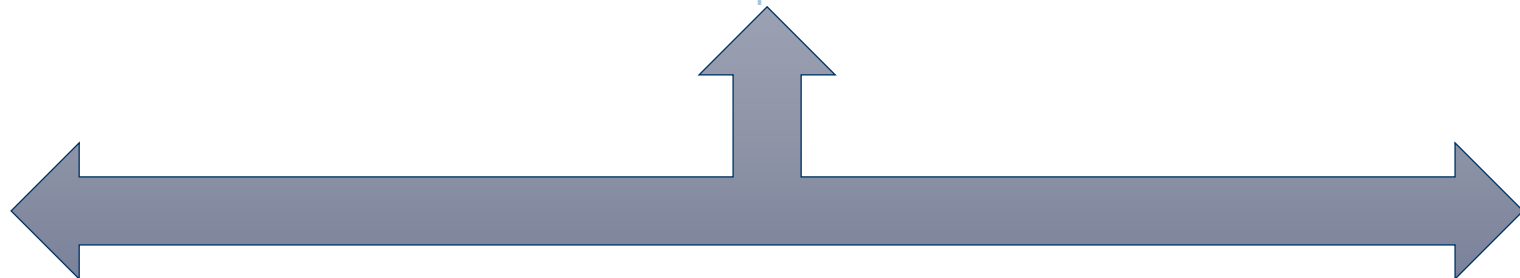
1. Jenkins支持丰富的API接口，通过调用接口可以封装和实现几乎所有需要的功能。
2. Pipeline as code的特性，让Pipeline可以对外输出数据和能力。

一站式智能研发协作平台



微医研发协作平台（WCP）

推动开发、测试、运维进行一站式研发协作的主系统



资源管控平台
K8s为核心的云资源平台



质量基础平台
各配套测试平台



Development CI

持续集成平台

Jenkins为核心的CI流水线



监控平台

全链路/拨测/运维/安全



项目管理平台

Redmine



发布管理平台

筋斗云+发布CI

一站式智能研发协作平台

应用申
请

资源申
请

交付流
水线

发布管
理

监控管
理

故障管
理

质效看
板

WCP



应用管理

资源管理

流水线管理

发布管理

监控管理

质效管理

质效看板

性能曲线 (APP)

Crash率 (APP)

问题管理

技术工具箱

质效数据

团队：全部

应用：全部

Q查询

场景			开发阶段				测试阶段		发布阶段		运营阶段
编号	应用	团队	阻断违规数	严重违规数	代码重复率(%)	单测覆盖率(%)	Pipeline执行数	Pipeline成功率(%)	发布执行数	发布成功率(%)	外部缺陷数
1	wcp	质量中心	5	7	30.5%	10.5%	10	90.32%	10	95.3%	1
2	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
3	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
4	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
5	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
6	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
7	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
8	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
9	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5
10	guser	应用平台	1	2	20.98%	0.2%	9	60.2%	25	90.2%	5

Thank You

欢迎加入微医

Email:jianggy@guahao.com