

第四周

1. 部署 jenkins master 及多 slave 环境
2. 基于 jenkins 视图对 jenkins job 进行分类
3. 总结 jenkins pipeline 基本语法
4. 部署代码质量检测服务 sonarqube
5. 基于命令、shell 脚本和 pipeline 实现代码质量检测

扩展题（选做）：

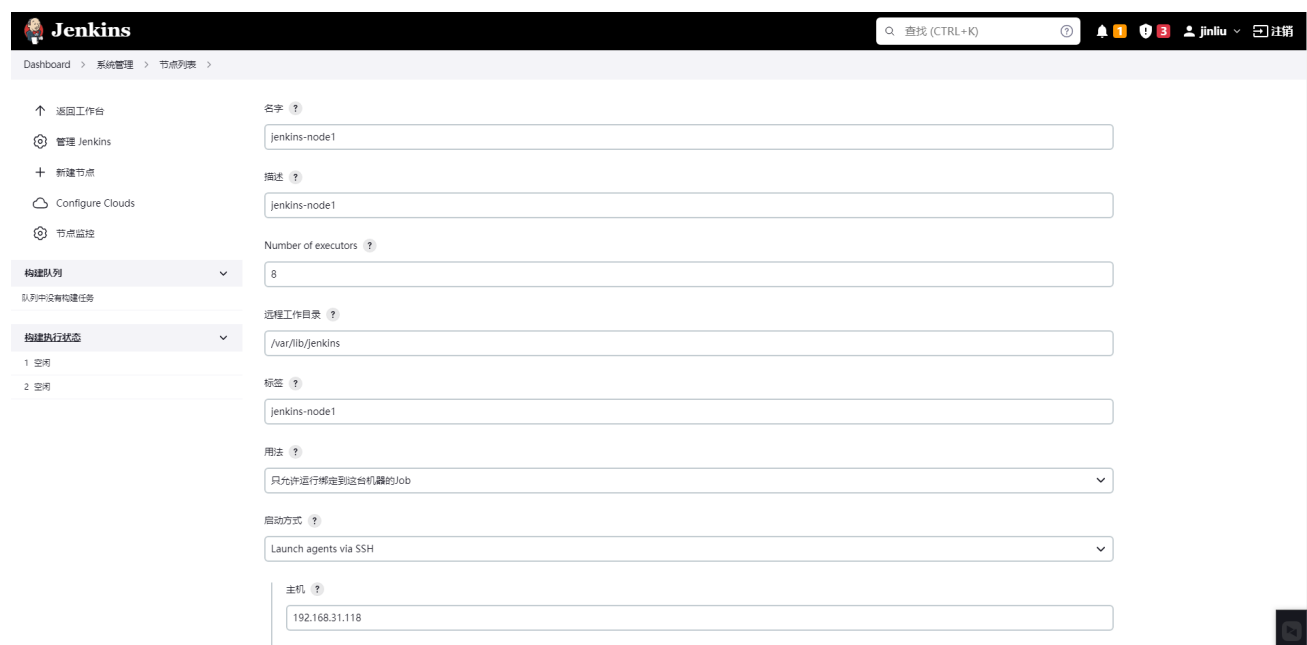
1. jenkins 安装 Sonarqube Scanner 插件、配置 sonarqube server 地址、基于 jenkins 配置代码扫描参数实现代码质量扫描
2. Execute SonarQube Scanner

1. 部署 jenkins master 及多 slave 环境

1.1. 创建准备

1 #JDK、工作路径、安装git、mvn等必要工具

1.2. 新加节点



The screenshot displays the Jenkins 'Add Node' configuration interface. The top navigation bar includes the Jenkins logo, a search bar, and user information. The left sidebar contains navigation links: 'Return to workspace', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. The main content area is titled 'Add Node' and contains the following fields:

- Name:** jenkins-node1
- Description:** jenkins-node1
- Number of executors:** 8
- Remote workspace:** /var/lib/jenkins
- Label:** jenkins-node1
- Usage:** Only allow jobs that run on this machine
- Startup mode:** Launch agents via SSH
- Host:** 192.168.31.118

1.3. 配置凭证

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 ?

root-ssh-passwd

添加 取消

1.4. 验证状态

Jenkins

Dashboard > 系统管理 > 节点列表

返回工作台

管理 Jenkins

新建节点

Configure Clouds

节点监控

构建队列

队列中没有构建任务

构建执行状态

master

1 空闲

2 空闲

jenkins-node1

1 空闲

2 空闲

3 空闲

4 空闲

5 空闲

6 空闲

7 空闲

8 空闲

Manage nodes and clouds

删除状态

\$	名称 ↓	架构	时钟差异	剩余磁盘空间	剩余交换空间	剩余临时空间	响应时间
	jenkins-node1	Linux (amd64)	已同步	41.17 GB	0 B	41.17 GB	4ms
	jenkins-node2	Linux (amd64)	已同步	41.08 GB	0 B	41.08 GB	3ms
	master	Linux (amd64)	已同步	39.89 GB	0 B	39.89 GB	0ms
	获取到的数据	15 毫秒	13 毫秒	13 毫秒	13 毫秒	11 毫秒	10 毫秒

2. 基于 jenkins 视图对 jenkins job 进行分类

2.1. 新建试图

Jenkins

Dashboard >

+ 新建任务

👤 用户列表

📁 构建历史

🔗 项目关系

🔍 检查文件指纹

⚙️ 系统管理

👁️ 我的视图

🌊 打开 Blue Ocean

构建队列

队列中没有构建任务

构建执行状态

master

- 1 空闲
- 2 空闲

jenkins-node1

- 1 空闲
- 2 空闲
- 3 空闲
- 4 空闲

jinliu-list

jinliu-nginx-list

jinliu-view

所有

+

S	W	名称 ↓	上次成功	上次失败	上次持续时间	收藏
✅	✖	dd	10 天 #1	无	81 毫秒	▶ ☆
❌	✖	jinliu-envtest	没有	无	无	▶ ☆
❌	✖	jinliu-nginx-001	没有	无	无	▶ ☆
❌	✖	jinliu-nginx-002	没有	无	无	▶ ☆
✅	✖	jinliutest	3 天 23 小时 #13	无	1.5 秒	▶ ☆
❌	✖	jinliutest-post	没有	无	无	▶ ☆
✅	✖	jinliutest01	3 天 23 小时 #4	无	4.6 秒	▶ ☆


图标: 小 中 大

图例

Atom feed 全部

Atom feed 失败

Atom feed 最新的构建




Jenkins

Dashboard


>

+


新建任务




用户列表




构建历史




项目关系




检查文件接收



系统管理



我的视图



打开 Blue Ocean


构建队列

▼

队列中没有任何构建任务

构建执行状态


▼



master

1 空闲

2 空闲



jenkins-node1

1 空闲

2 空闲

3 空闲

4 空闲

新建视图

视图名称

jinliu-app-list

Type

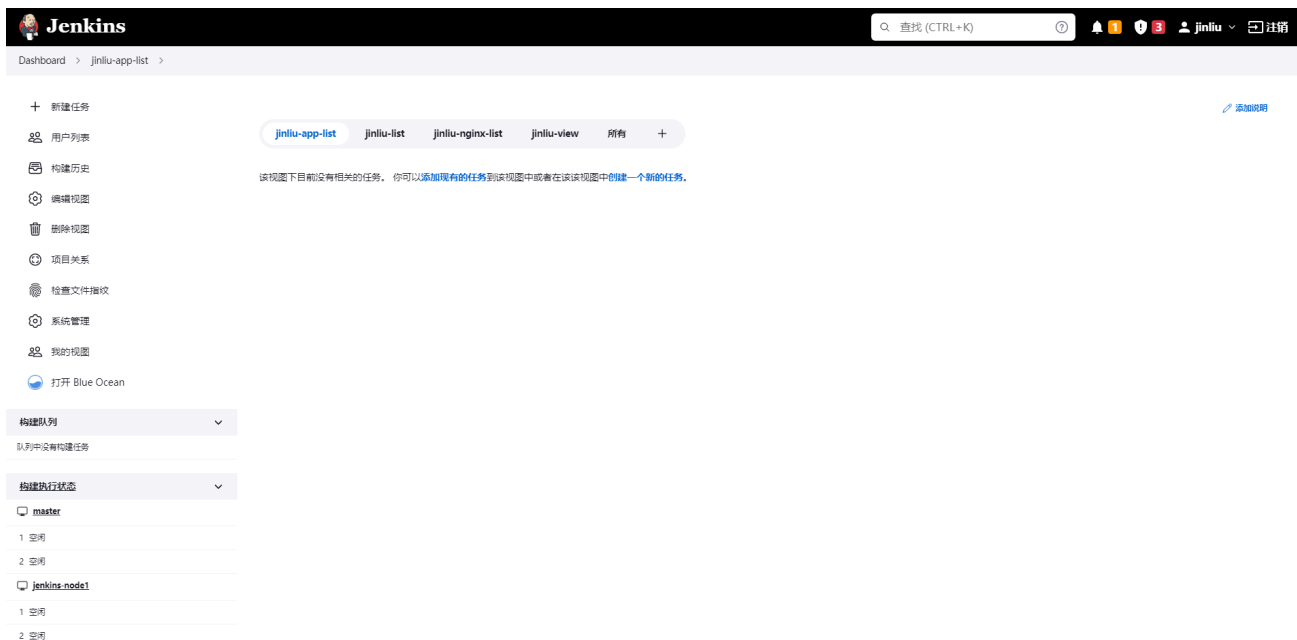
☒ 列表视图

显示简单列表。你可以从中选择任务来显示在某个视图中

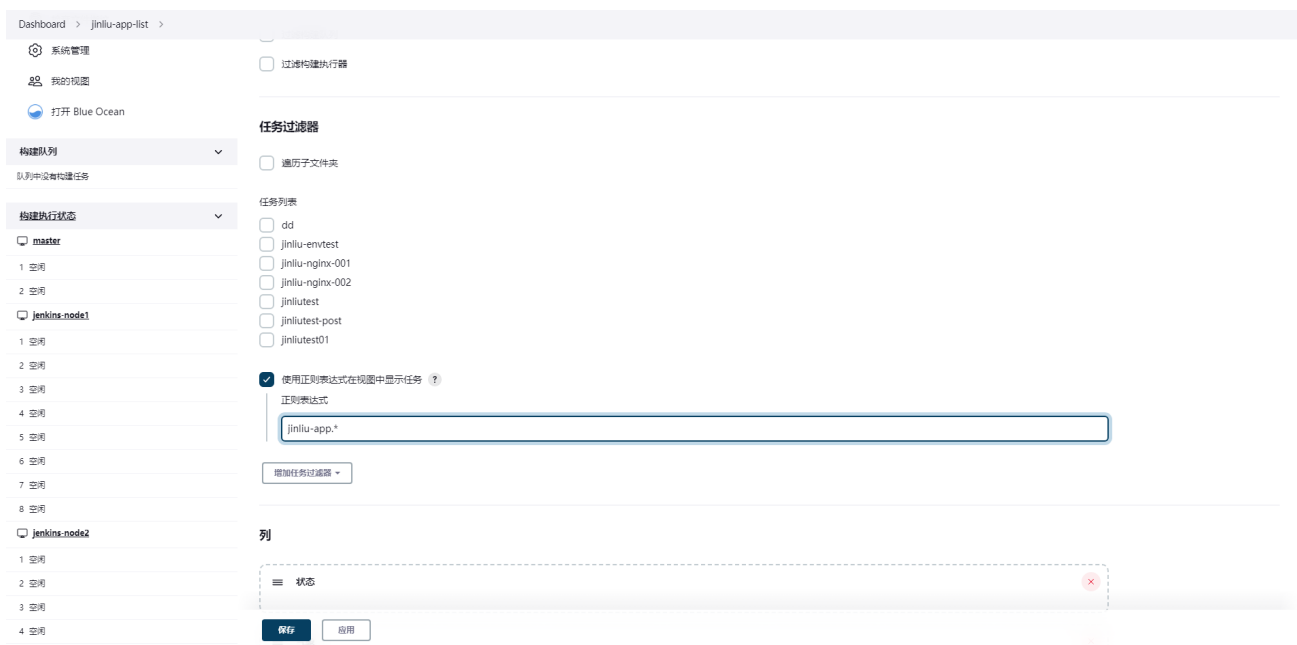
☐ 我的视图

该视图自动显示当前用户有权访问的任务

Create

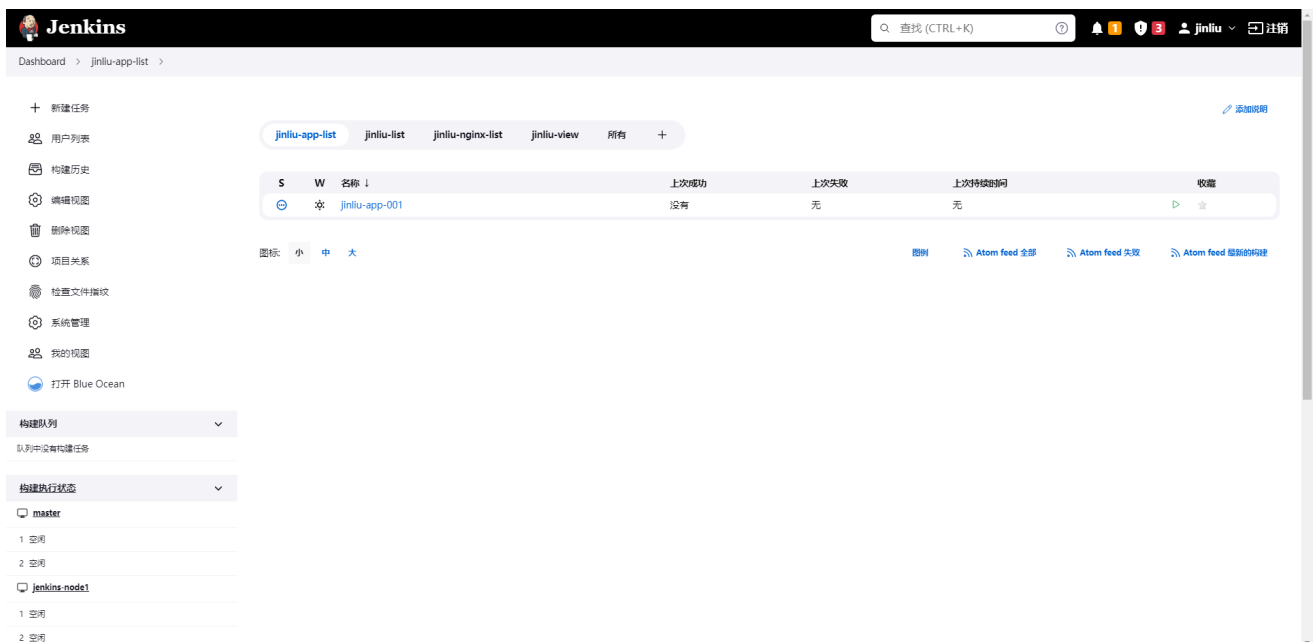
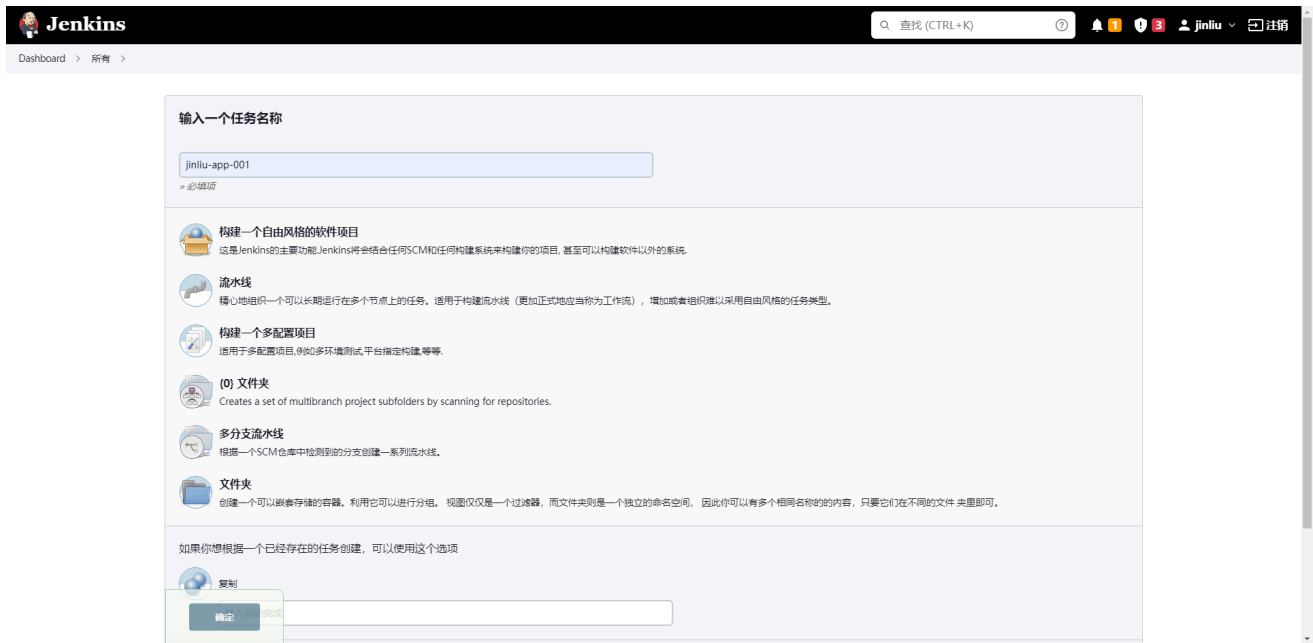


2.2. 设置正则



2.3. 验证任务自动归类

创建pipeline, "jiniu-app-001"



3. 总结 jenkins pipeline 基本语法

3.1. 简介

Pipeline支持两种语法： Declarative Pipeline（声明式pipeline，在pipeline2.5中引入，结构化方式）和Scripted Pipeline（脚本式pipeline），两者都支持建立连续输送的Pipeline。

相关资料：<https://stackoverflow.com/questions/43484979/jenkins-scripted-pipeline-or-declarative-pipeline>
<http://jenkins-ci.361315.n4.nabble.com/Declarative-pipelines-vs-scripted-td4891792.html> 声明式Pipeline是后续Open Blue Ocean所支持类型，建议使用声明式Pipeline的方式进行编写，从jenkins社区动向看，很明显这种语法结构会是未来的趋势。

- 声明式pipeline可以内嵌脚本式pipeline
- 声明式pipeline必须包含在固定格式的pipeline{}内
- 块（Block{}）： 只能包含章节Sections，指令Directives，步骤Steps或者赋值语句
- 章节（Sections）： 通常包括一个或多个指令或步骤，如agent, post, stages, steps

- 指令 (Directives) : environment, options, parameters, triggers, stage, tools, when
- 步骤 (steps) : 执行脚本式pipeline, 如script{}

3.2. Declarative Pipeline (声明式pipeline)

3.2.1. agent

Pipeline或特定阶段将在Jenkins环境中执行的位置，具体取决于该agent 部分的放置位置；必须在pipeline顶层定义。

3.2.1.1. 参数:

- any: 在任何可用的agent 上执行Pipeline或stage。例如: agent any
- none: 当在pipeline块的顶层使用none时，将不会为整个Pipeline运行分配全局agent，每个stage部分将需要包含其自己的agent部分。
- label: 使用有label标签的agent，例如: agent { label 'my-defined-label' }
- node: agent { node { label 'labelName' } }，等同于 agent { label 'labelName' }，但node允许其他选项（如 customWorkspace）。
- docker: 动态供应一个docker节点去执行pipeline或stage，docker还可以接受一个args，直接传递给docker run调用。

```

1  agent {
2      docker {
3          image 'maven:3-alpine'
4          label 'my-defined-label'
5          args '-v /tmp:/tmp'
6      }
7  }
8

```

- dockerfile: Dockerfile源存储库中包含的容器来构建执行Pipeline或stage。使用此参数，jenkinsfile必须从代码中加载使用“pipeline from SCM”或者“Multibranch Pipeline”加载

默认是Dockerfile在根目录: agent { dockerfile true } 如果Dockerfile在另一个目录，使用dir参数: agent { dockerfile { dir 'someSubDir' } } 可以使用docker build添加参数: agent { dockerfile { additionalBuildArgs '--build-arg foo=bar' } }

```

1  pipeline {
2      agent { dockerfile true }
3      stages {
4          stage('Test') {
5              steps {
6                  sh 'node --version'
7                  sh 'svn --version'
8              }
9          }
10     }
11 }
12

```

3.2.1.2. 常用选项:

- label: 一个字符串, 选择哪个特定的label标签, 此选项适用于node, docker和dockerfile, 并且 node是必需的。
- customWorkspace: 一个字符串, 自定义工作空间, 可以使相对路径, 也可以是绝对路径。

```

1 agent {
2     node {
3         label 'my-defined-label'
4         customWorkspace '/some/other/path'
5     }
6 }
7

```

- reuseNode: 一个布尔值, 默认false, 如果为true, 在同一工作空间中, 适用于docker和dockerfile, 并且仅在 单个的stage中使用agent才有效。

```

1 pipeline {
2     //Execute all the steps defined in this Pipeline within a newly created container
    of the given name and tag (maven:3-alpine).
3     agent { docker 'maven:3-alpine' }
4     stages {
5         stage('Example Build') {
6             steps {
7                 sh 'mvn -B clean verify'
8             }
9         }
10    }
11 }
12

```

```

1 pipeline {
2     //使用多个代理, pipeline顶层agent none, 每个stage有各自的agent代理
3     agent none
4     stages {
5         stage('Example Build') {
6             agent { docker 'maven:3-alpine' }
7             steps {
8                 echo 'Hello, Maven'
9                 sh 'mvn --version'
10            }
11        }
12        stage('Example Test') {
13            agent { docker 'openjdk:8-jre' }
14            steps {
15                echo 'Hello, JDK'
16                sh 'java -version'
17            }
18        }
19    }
20 }
21

```

3.2.2. post

定义Pipeline或stage运行结束时的操作。

- always: 运行, 无论Pipeline运行的完成状态如何。
- changed: 只有当前Pipeline运行的状态与先前完成的Pipeline的状态不同时, 才能运行。
- failure: 仅当当前Pipeline处于“失败”状态时才运行, 通常在Web UI中用红色指示表示。
- success: 仅当当前Pipeline具有“成功”状态时才运行, 通常在具有蓝色或绿色指示的Web UI中表示。
- unstable: 只有当前Pipeline具有“不稳定”状态, 通常由测试失败, 代码违例等引起, 才能运行。通常在具有黄色指示的Web UI中表示。
- aborted: 只有当前Pipeline处于“中止”状态时, 才会运行, 通常是由于Pipeline被手动中止。通常在具有灰色指示的Web UI中表示。

```
1 pipeline {
2     environment {
3         CRDE_EMAIL='xxx@163.com'
4     }
5     post {
6         success {
7             script {
8                 //使用wrap([$class: 'BuildUser'])需要安装user build vars plugin插件
9                 // JOB_NAME,BUILD_NUMBER,BUILD_USER,env.BUILD_URL是jenkins pipeline内部变量
10                wrap([$class: 'BuildUser']) {
11                    mail to: "${CRDE_EMAIL}",
12                    subject: "pipeline '${JOB_NAME}' (${BUILD_NUMBER})
13result",
14                    body: "${BUILD_USER}'s pipeline '${JOB_NAME}'
15(${BUILD_NUMBER}) run success\n请及时前往${env.BUILD_URL}进行查看."
16                }
17            }
18        }
19        failure {
20            script {
21                wrap([$class: 'BuildUser']) {
22                    mail to: "${CRDE_EMAIL}",
23                    subject: "pipeline '${JOB_NAME}' (${BUILD_NUMBER})
24result",
25                    body: "${BUILD_USER}'s pipeline '${JOB_NAME}'
26(${BUILD_NUMBER}) run failure\n请及时前往${env.BUILD_URL}进行查看."
27                }
28            }
29        }
30        unstable {
31            script {
32                wrap([$class: 'BuildUser']) {
33                    mail to: "${CRDE_EMAIL}",
34                    subject: "pipeline '${JOB_NAME}' (${BUILD_NUMBER})
35result",
36                    body: "${BUILD_USER}'s pipeline '${JOB_NAME}'
37(${BUILD_NUMBER}) run unstable\n请及时前往${env.BUILD_URL}进行查看."
38                }
39            }
40        }
41    }
42 }
```



```
35     }
36 }
37
```

3.2.3. stages

包含一个或多个stage的序列，Pipeline的大部分工作在此执行。建议stages至少包含至少一个stage指令，用于连接各个交付过程，如构建，测试和部署等。

3.2.4. steps

steps包含一个或多个在stage块中执行的step序列。

3.2.5. Directives (指令)

3.2.5.1. environment

environment指令指定一系列键值对，这些键值对将被定义为所有step或stage-specific step的环境变量，具体取决于environment指令在Pipeline中的位置。该指令支持一种特殊的方法credentials()，可以通过其在Jenkins环境中的标识符来访问预定义的凭据。对于类型为“Secret Text”的凭据，该credentials()方法将确保指定的环境变量包含Secret Text内容；对于“标准用户名和密码”类型的凭证，指定的环境变量将被设置为username:password。每个stage可以有独自的environment块

```
1 pipeline {
2     agent any
3     environment {
4         CC = 'clang'
5     }
6     stages {
7         stage('Example') {
8             environment {
9                 AN_ACCESS_KEY = credentials('my-prefined-secret-text')
10            }
11            steps {
12                sh 'printenv'
13            }
14        }
15    }
16 }
17
```

3.2.5.2. options

允许在Pipeline本身内配置Pipeline专用选项。Pipeline本身提供了许多选项，例如buildDiscarder，但它们也可能由插件提供，例如 timestamps。

3.2.5.2.1. 常用选项

- buildDiscarder: pipeline保持构建的最大个数。例如：options { buildDiscarder(logRotator(numToKeepStr: '1')) }
- disableConcurrentBuilds: 不允许并行执行Pipeline,可用于防止同时访问共享资源等。例如：options { disableConcurrentBuilds() }
- skipDefaultCheckout: 默认跳过来自源代码控制的代码。例如：options { skipDefaultCheckout() }

- skipStagesAfterUnstable: 一旦构建状态进入了“Unstable”状态，就跳过此stage。例如：options { skipStagesAfterUnstable() }
- timeout: 设置Pipeline运行的超时时间。例如：options { timeout(time: 1, unit: 'HOURS') }
- retry: 失败后，重试整个Pipeline的次数。例如：options { retry(3) }
- timestamps: 预定义由Pipeline生成的所有控制台输出时间。例如：options { timestamps() }

```

1 pipeline {
2     agent any
3     options {
4         timeout(time: 1, unit: 'HOURS')
5     }
6     stages {
7         stage('Example') {
8             steps {
9                 echo 'Hello World'
10            }
11        }
12    }
13 }
14

```

3.2.5.3. parameters

parameters指令提供用户在触发Pipeline时的参数列表。这些参数值通过该params对象可用于Pipeline步骤。

3.2.5.3.1. 常用参数

string, choice, booleanParam等

```

1 pipeline {
2     agent any
3     environment {
4         CRDE_EMAIL='xxx@163.com'
5     }
6     parameters {
7         string(name: 'PERSON', defaultValue: 'Mr Jenkins', description: 'Who should I
8 say hello to?')
9         choice(name: 'server', choices: '192.168.1.1,22,vito,vito111', description:
10 '测试服务器列表选择 (IP,SshPort,Name,Passwd) ')
11         booleanParam(name: 'isCommit', description: '是否邮件通知部署人员', defaultValue:
12 false)
13     }
14     stages {
15         stage('Example') {
16             steps {
17                 echo "Hello ${params.PERSON}"
18             }
19             script {
20                 def split = ${params.server.split(",")}
21                 serverIP = split[0]
22                 sshport = split[1]
23                 username = split[2]
24                 password = split[3]
25                 echo

```

```

"serverIP:${serverIP},sshport:${sshport},username:${username},password:${password}"
22     }
23     }
24     }
25     stage('通知人工验收') {
26         steps {
27             script {
28                 wrap([$class: 'BuildUser']) {
29                     if(params.isCommit==false){
30                         echo "不需要通知部署人员人工验收"
31                     }
32                     else{
33                         //邮件通知测试人员人工验收
34                         mail to: "${CRDE_EMAIL}",
35                             subject: "pipeline '${JOB_NAME}'
36                             (${BUILD_NUMBER}) 人工验收通知",
37                             body: "${BUILD_USER}提交的PineLine '${JOB_NAME}'
38                             (${BUILD_NUMBER}) 进入人工验收环节\n请及时前往${env.BUILD_URL}进行测试验收"
39                     }
40                 }
41             }
42         }
43     }
44

```

3.2.5.4. triggers

triggers指令定义了Pipeline自动化触发的方式。对于与源代码集成的Pipeline，如GitHub或BitBucket，triggers可能不需要基于webhook的集成也已经存在。目前只有两个可用的触发器：cron和pollSCM。

- cron: 接受一个cron风格的字符串来定义Pipeline触发的常规间隔，例如：triggers { cron('H 4/* 0 0 1-5') }
- pollSCM: 接受一个cron风格的字符串来定义Jenkins检查SCM源更改的常规间隔。如果存在新的更改，则Pipeline将被重新触发。例如：triggers { pollSCM('H 4/* 0 0 1-5') }

```

1 pipeline {
2     agent any
3     triggers {
4         cron('H 4/* 0 0 1-5')
5     }
6     stages {
7         stage('Example') {
8             steps {
9                 echo 'Hello World'
10            }
11        }
12    }
13 }
14

```

3.2.5.5. stage

stage指令在stages部分中，应包含stop部分，可选agent部分或其他特定于stage的指令。实际上，Pipeline完成的所有实际工作都将包含在一个或多个stage指令中。

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

3.2.5.6. tools

```
1 #通过tools可自动安装工具，并放置环境变量到PATH。如果agent none，这将被忽略。
2 #maven
3 #jdk
4 #gradle
```

```
1 pipeline {
2     agent any
3     tools {
4         //工具名称必须在Jenkins 管理Jenkins → 全局工具配置中预配置。
5         maven 'apache-maven-3.0.1'
6     }
7     stages {
8         stage('Example') {
9             steps {
10                 sh 'mvn --version'
11             }
12         }
13     }
14 }
15
```

3.2.5.7. when

```
1 #when指令允许Pipeline根据给定的条件确定是否执行该阶段。该when指令必须至少包含一个条件。如果when指令包含多个条件，则所有子条件必须为stage执行返回true。这与子条件嵌套在一个allOf条件中相同。
```

3.2.5.7.1. 内置条件

- branch: 当正在构建的分支与给出的分支模式匹配时执行，例如：when { branch 'master' }。请注意，这仅适用于多分支Pipeline。

- environment: 当指定的环境变量设置为给定值时执行，例如：when { environment name: 'DEPLOY_TO', value: 'production' }
- expression: 当指定的Groovy表达式求值为true时执行，例如：when { expression { return params.DEBUG_BUILD } }
- not: 当嵌套条件为false时执行。必须包含一个条件。例如：when { not { branch 'master' } }
- allOf: 当所有嵌套条件都为真时执行。必须至少包含一个条件。例如：when { allOf { branch 'master'; environment name: 'DEPLOY_TO', value: 'production' } }
- anyOf: 当至少一个嵌套条件为真时执行。必须至少包含一个条件。例如：when { anyOf { branch 'master'; branch 'staging' } }

```

1  pipeline {
2      agent any
3      stages {
4          stage('Example Build') {
5              steps {
6                  echo 'Hello World'
7              }
8          }
9          stage('Example Deploy') {
10             when {
11                 allOf {
12                     branch 'production'
13                     environment name: 'DEPLOY_TO', value: 'production'
14                 }
15             }
16             steps {
17                 echo 'Deploying'
18             }
19         }
20     }
21 }
22

```

3.2.6. Parallel（并行）

Declarative Pipeline近期新增了对并行嵌套stage的支持，对耗时长，相互不存在依赖的stage可以使用此方式提升运行效率。除了parallel stage，单个parallel里的多个step也可以使用并行的方式运行。

```

1  pipeline {
2      agent any
3      stages {
4          stage('Non-Parallel Stage') {
5              steps {
6                  echo 'This stage will be executed first.'
7              }
8          }
9          stage('Parallel Stage') {
10             when {
11                 branch 'master'
12             }
13             parallel {

```

```

14         stage('Branch A') {
15             agent {
16                 label "for-branch-a"
17             }
18             steps {
19                 echo "On Branch A"
20             }
21         }
22         stage('Branch B') {
23             agent {
24                 label "for-branch-b"
25             }
26             steps {
27                 echo "On Branch B"
28             }
29         }
30     }
31 }
32 }
33 }
34

```

3.2.7. Steps (步骤)

Declarative Pipeline可以使用 Pipeline Steps reference中的所有可用步骤，并附加以下仅在Declarative Pipeline中支持的步骤。

3.2.7.1. script

script步骤需要一个script Pipeline，并在Declarative Pipeline中执行。对于大多数用例，script在Declarative Pipeline中的步骤不是必须的，但它可以提供一个有用的加强。

```

1  pipeline {
2      agent any
3      stages {
4          stage('Example') {
5              steps {
6                  echo 'Hello World'
7                  script {
8                      def browsers = ['chrome', 'firefox']
9                      for (int i = 0; i < browsers.size(); ++i) {
10                         echo "Testing the ${browsers[i]} browser"
11                     }
12                 }
13             }
14         }
15     }
16 }
17

```

3.3. Scripted Pipeline (脚本式pipeline)

3.3.1. 流程控制

pipeline脚本同其它脚本语言一样，从上至下顺序执行，它的流程控制取决于Groovy表达式，如if/else条件语句，举例如下：

```
1 Jenkinsfile (Scripted Pipeline)
2 node {
3     stage('Example') {
4         if (env.BRANCH_NAME == 'master') {
5             echo 'I only execute on the master branch'
6         } else {
7             echo 'I execute elsewhere'
8         }
9     }
10 }
11
```

pipeline脚本流程控制的另一种方式是Groovy的异常处理机制。当任何一个步骤因各种原因而出现异常时，都必须在Groovy中使用try/catch/finally语句块进行处理，举例如下：

```
1 Jenkinsfile (Scripted Pipeline)
2 node {
3     stage('Example') {
4         try {
5             sh 'exit 1'
6         }
7         catch (exc) {
8             echo 'Something failed, I should sound the klaxons!'
9             throw
10        }
11    }
12 }
13
```

3.3.2. Steps

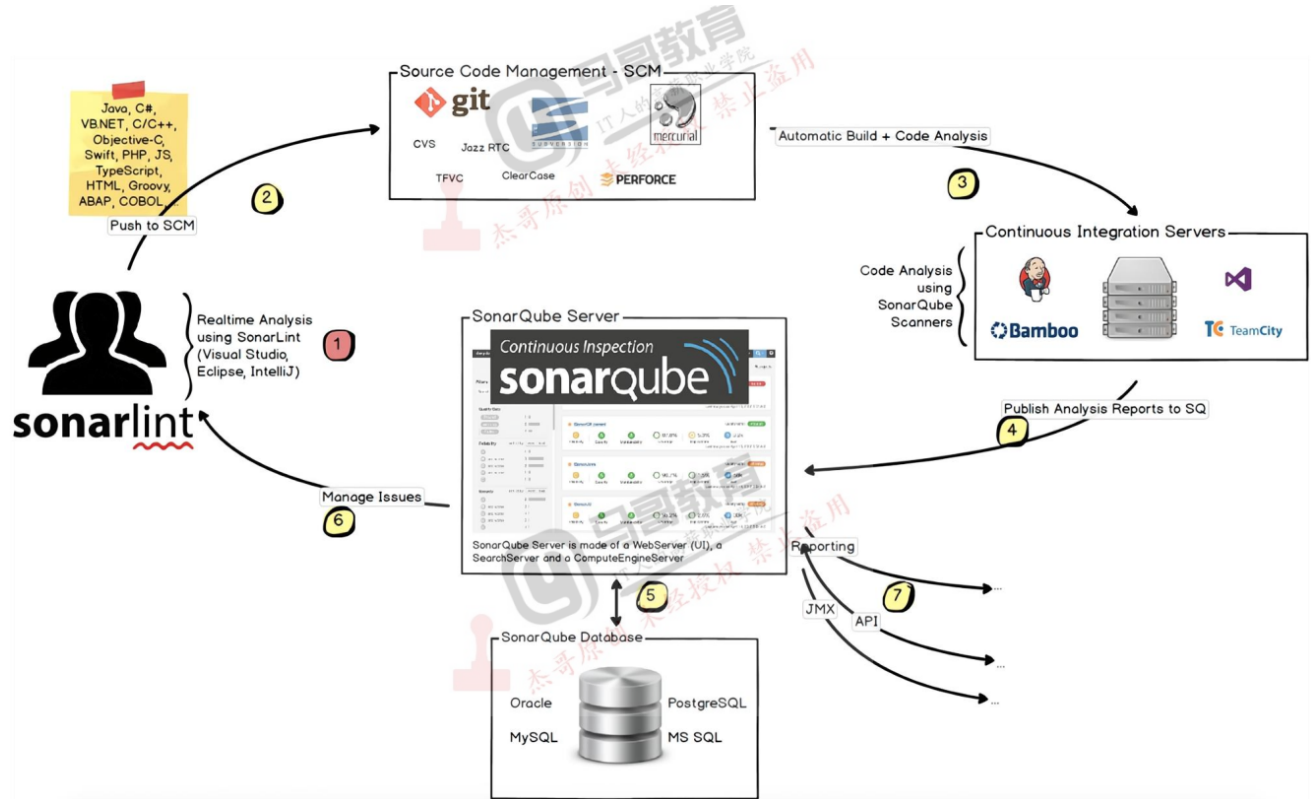
pipeline最核心和基本的部分就是“step”，从根本上来说，steps作为Declarative pipeline和Scripted pipeline语法的最基本的语句构建块来告诉jenkins应该执行什么操作。

3.4. Declarative pipeline和Scripted pipeline的比较

- 共同点：两者都是pipeline代码的持久实现，都能够使用pipeline内置的插件或者插件提供的steps，两者都可以利用共享库扩展。
- 区别：两者不同之处在于语法和灵活性。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，使其成为用户更理想的选择。但是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。

4. 部署代码质量检测服务 sonarqube

4.1. 扫描流程



4.2. 部署准备

4.2.1. 安装jdk

```
1 [root@k8s-jenkins-sonarqube-node1 ~]# java -version
2 openjdk version "11.0.17" 2022-10-18 LTS
3 OpenJDK Runtime Environment (Red_Hat-11.0.17.0.8-2.el7_9) (build 11.0.17+8-LTS)
4 OpenJDK 64-Bit Server VM (Red_Hat-11.0.17.0.8-2.el7_9) (build 11.0.17+8-LTS, mixed
  mode, sharing)
5 [root@k8s-jenkins-sonarqube-node1 ~]#
```

4.2.2. 修改内核参数&资源限制

```
1 [root@k8s-jenkins-sonarqube-node1 ~]# sysctl -p
2 vm.max_map_count = 524288
3 fs.file-max = 131072
4 [root@k8s-jenkins-sonarqube-node1 ~]# ulimit -n
5 1024
6 [root@k8s-jenkins-sonarqube-node1 ~]# ulimit -n 131072
7 [root@k8s-jenkins-sonarqube-node1 ~]# ulimit -n
8 131072
9 [root@k8s-jenkins-sonarqube-node1 ~]# ulimit -u 8192
10 You have new mail in /var/spool/mail/root
11 [root@k8s-jenkins-sonarqube-node1 ~]# ulimit -u
12 8192
13 [root@k8s-jenkins-sonarqube-node1 ~]#
```


4.3. 安装pg

```
1 [root@k8s-jenkins-sonarqube-node1 ~]# yum install postgresql
2 .....
3 Running transaction
4   Installing : postgresql-libs-9.2.24-8.el7_9.x86_64
                                                    1/2
5   Installing : postgresql-9.2.24-8.el7_9.x86_64
                                                    2/2
6   Verifying   : postgresql-libs-9.2.24-8.el7_9.x86_64
                                                    1/2
7   Verifying   : postgresql-9.2.24-8.el7_9.x86_64
                                                    2/2
8
9   Installed:
10    postgresql.x86_64 0:9.2.24-8.el7_9
11
12   Dependency Installed:
13    postgresql-libs.x86_64 0:9.2.24-8.el7_9
14
15   Complete!
16   ##版本不对, 重新下载下载源再安装
17 [root@k8s-jenkins-sonarqube-node1 ~]# yum remove postgresql
18 [root@k8s-jenkins-sonarqube-node1 ~]# yum install -y
   https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-
   latest.noarch.rpm
19 # (https://www.postgresql.org/download/linux/redhat/) 访问官网查询下载repo
20 [root@k8s-jenkins-sonarqube-node1 ~]# yum install -y postgresql12-server
21 .....
22   Installed:
23    postgresql12-server.x86_64 0:12.13-1PGDG.rhel7
24
25   Dependency Installed:
26    libicu.x86_64 0:50.2-4.el7_7
27    postgresql12.x86_64 0:12.13-1PGDG.rhel7
28    postgresql12-libs.x86_64 0:12.13-1PGDG.rhel7
29
30   Complete!
31 [root@k8s-jenkins-sonarqube-node1 ~]# /usr/pgsql-12/bin/postgresql-12-setup initdb
32 Initializing database ... OK
33 #修改postgresql.conf和pg_hba.conf
34
35 [root@k8s-jenkins-sonarqube-node1 data]# cat postgresql.conf |grep -A 5
```

```

listen_addresses
33 listen_addresses = '*'          # what IP address(es) to listen on;
34                                # comma-separated list of addresses;
35                                # defaults to 'localhost'; use '*' for all
36                                # (change requires restart)
37 port = 5432                     # (change requires restart)
38 max_connections = 4096          # (change requires restart)
39 [root@k8s-jenkins-sonarqube-node1 data]# cat pg_hba.conf |grep -A 2 IPv4
40 # an integer (between 0 and 32 (IPv4) or 128 (IPv6) inclusive) that
41 # specifies the number of significant bits in the mask. A host name
42 # that starts with a dot (.) matches a suffix of the actual host name.
43 --
44 # IPv4 local connections:
45 host      all             all             0.0.0.0/0          ident
46 [root@k8s-jenkins-sonarqube-node1 data]# systemctl enable postgresql-12 --now
47 Created symlink from /etc/systemd/system/multi-user.target.wants/postgresql-
  12.service to /usr/lib/systemd/system/postgresql-12.service.
48 [root@k8s-jenkins-sonarqube-node1 data]# ps -ef |grep sql
49 postgres 22226      1  1 19:30 ?                00:00:00 /usr/pgsql-12/bin/postmaster -D
  /var/lib/pgsql/12/data/

```

4.3.1. 创建库&用户

```

1 [root@k8s-jenkins-sonarqube-node1 data]# su - postgres
2 -bash-4.2$ source
3 -bash: source: filename argument required
4 source: usage: source filename [arguments]
5 -bash-4.2$ psql -U postgres
6 psql (12.13)
7 Type "help" for help.
8
9 postgres=# CREATE DATABASE sonar;
10 CREATE DATABASE
11 postgres=# CREATE USER sonar WITH ENCRYPTED PASSWORD '111111';
12 CREATE ROLE
13 postgres=# GRANT ALL PRIVILEGES ON DATABASE sonar TO sonar;
14 GRANT
15 postgres=# ALTER DATABASE sonar OWNER TO sonar;
16 ALTER DATABASE
17 postgres=# exit

```

4.4. 安装sonarqube

```

1 [root@k8s-jenkins-sonarqube-node1 ~]# mkdir /apps
2 [root@k8s-jenkins-sonarqube-node1 apps]# unzip sonarqube-8.9.10.61524\ (1\).zip
3 [root@k8s-jenkins-sonarqube-node1 apps]# ln -sv /apps/sonarqube-8.9.10.61524
  /apps/sonarqube
4 '/apps/sonarqube' -> '/apps/sonarqube-8.9.10.61524'
5 You have new mail in /var/spool/mail/root
6 [root@k8s-jenkins-sonarqube-node1 apps]# ls -l
7 total 0
8 lrwxrwxrwx  1 root root  28 Dec  4 20:14 sonarqube -> /apps/sonarqube-8.9.10.61524

```

```

 9  drwxr-xr-x 11 root root 172 Oct 14 09:25 sonarqube-8.9.10.61524
10  [root@k8s-jenkins-sonarqube-node1 apps]# useradd -r -m -s /bin/bash sonarqube &&
    chown sonarqube.sonarqube /apps/ -R
11  [root@k8s-jenkins-sonarqube-node1 apps]# ls -l
12  total 0
13  lrwxrwxrwx 1 sonarqube sonarqube 28 Dec 4 20:14 sonarqube -> /apps/sonarqube-
    8.9.10.61524
14  drwxr-xr-x 11 sonarqube sonarqube 172 Oct 14 09:25 sonarqube-8.9.10.61524
15  #修改sonar连接数据库配置
16  [root@k8s-jenkins-sonarqube-node1 apps]# vim /apps/sonarqube/conf/sonar.properties
17
18  #启动sonarqube
19  [sonarqube@k8s-jenkins-sonarqube-node1 ~]$ /apps/sonarqube/bin/linux-x86-64/sonar.sh
    start
20  Starting SonarQube...
21  Started SonarQube.
22  [sonarqube@k8s-jenkins-sonarqube-node1 ~]$

```

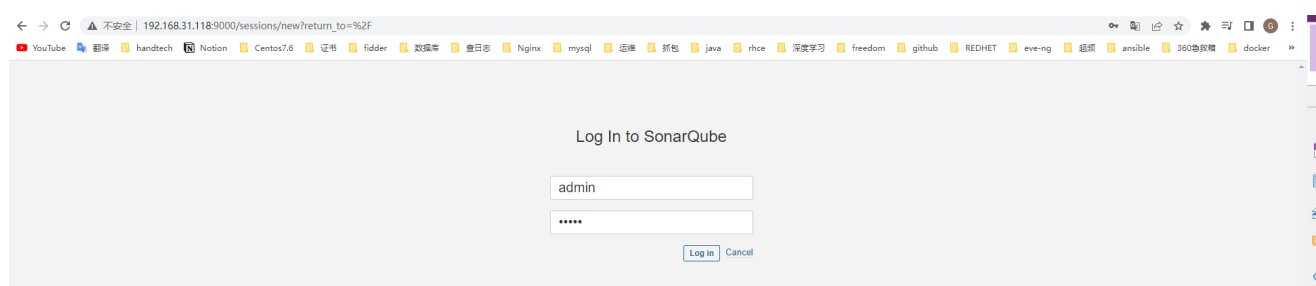
4.5. 验证sonarqube

注意：当未修改相关配置执行/apps/sonarqube/bin/linux-x86-64/sonar.sh start后服务启动异常，需要查看日志修改错误，并查看es进程并手动kill，否则启动还会报错

```

1  [sonarqube@k8s-jenkins-sonarqube-node1 ~]$ tail /apps/sonarqube/logs/*.log
2  .....
3  2022.12.04 21:01:44 INFO app[][o.s.a.SchedulerImpl] Process[ce] is up
4  2022.12.04 21:01:44 INFO app[][o.s.a.SchedulerImpl] SonarQube is up
5  .....
6  [sonarqube@k8s-jenkins-sonarqube-node1 ~]$ lsof -i:9000
7  COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
8  java 6212 sonarqube 12u IPv6 112093 0t0 TCP *:cslistener (LISTEN)
9  [sonarqube@k8s-jenkins-sonarqube-node1 ~]$

```



4.6. 配置systemd

```

1  [root@k8s-jenkins-sonarqube-node1 ~]# cat /etc/systemd/system/sonarqube.service
2  [Unit]
3  Description=SonarQube service
4  After=syslog.target network.target
5
6  [Service]
7  Type=simple
8  User=sonarqube

```

```

9  Group=sonarqube
10 PermissionsStartOnly=true
11 ExecStart=/bin/nohup /usr/bin/java -Xms32m -Xmx32m -Djava.net.preferIPv4Stack=true -
    jar /apps/sonarqube/lib/sonar-application-8.9.10.61524.jar
12 StandardOutput=syslog
13 LimitNOFILE=131072
14 LimitNPROC=8192
15 TimeoutStartSec=5
16 Restart=always
17 SuccessExitStatus=143
18
19 [Install]
20 WantedBy=multi-user.target
21 [root@k8s-jenkins-sonarqube-node1 ~]# systemctl enable sonarqube --now
22 Created symlink from /etc/systemd/system/multi-user.target.wants/sonarqube.service to
    /etc/systemd/system/sonarqube.service.
23 [root@k8s-jenkins-sonarqube-node1 ~]# lsof -i:9000
24 COMMAND      PID       USER    FD   TYPE DEVICE SIZE/OFF NODE NAME
25 java         10808 sonarqube  12u  IPv6 162474      0t0  TCP *:cslistener (LISTEN)

```

4.7. 部署sonar-scanner

```

1  [root@k8s-jenkins-node2 apps]# unzip sonar-scanner-cli-4.7.0.2747.zip
2  Archive:  sonar-scanner-cli-4.7.0.2747.zip
3      creating: sonar-scanner-4.7.0.2747/
4      creating: sonar-scanner-4.7.0.2747/bin/
5      creating: sonar-scanner-4.7.0.2747/conf/
6      creating: sonar-scanner-4.7.0.2747/lib/
7  inflating: sonar-scanner-4.7.0.2747/bin/sonar-scanner.bat
8  inflating: sonar-scanner-4.7.0.2747/bin/sonar-scanner-debug.bat
9  inflating: sonar-scanner-4.7.0.2747/bin/sonar-scanner-debug
10 inflating: sonar-scanner-4.7.0.2747/bin/sonar-scanner
11 inflating: sonar-scanner-4.7.0.2747/conf/sonar-scanner.properties
12 inflating: sonar-scanner-4.7.0.2747/lib/sonar-scanner-cli-4.7.0.2747.jar
13 [root@k8s-jenkins-node2 apps]# ln -sv /apps/sonar-scanner-4.7.0.2747 /apps/sonar-
    scanner
14 '/apps/sonar-scanner' -> '/apps/sonar-scanner-4.7.0.2747'
15 [root@k8s-jenkins-node2 apps]# ls -l
16 total 576
17 lrwxrwxrwx 1 root root      30 Dec  4 22:02 sonar-scanner -> /apps/sonar-scanner-
    4.7.0.2747
18 drwxr-xr-x 5 root root     40 Feb 22  2022 sonar-scanner-4.7.0.2747
19 -rw-r--r-- 1 root root 589185 Nov 14 23:57 sonar-scanner-cli-4.7.0.2747.zip
20
21 #修改配置
22 [root@k8s-jenkins-node2 apps]# cat sonar-scanner/conf/sonar-scanner.properties
23 #Configure here general information about the environment, such as SonarQube server
    connection details for example
24 #No information about specific project should appear here
25
26 #----- Default SonarQube server
27 sonar.host.url=http://192.168.31.118:9000

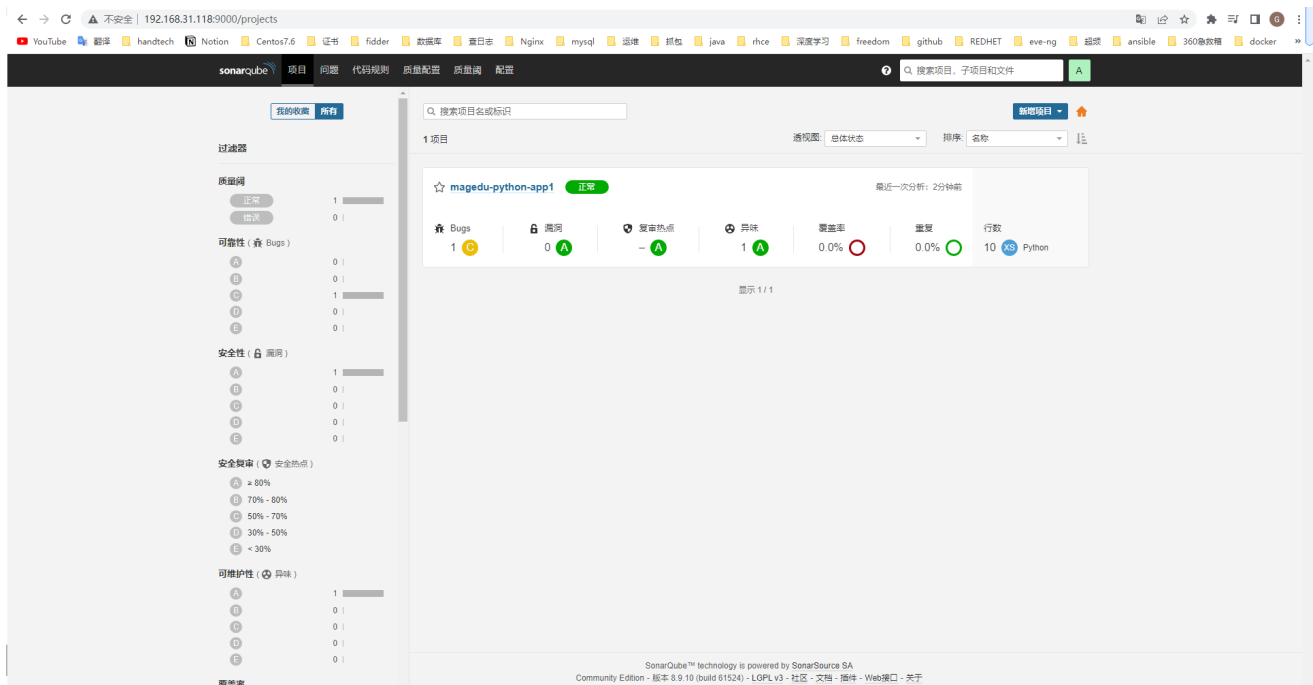
```

```
28
29 #----- Default source code encoding
30 sonar.sourceEncoding=UTF-8
```

5. 基于命令、shell 脚本和 pipeline 实现代码质量检测

5.1. 基于命令

```
1 [root@k8s-jenkins-node2 python-test]# /apps/sonar-scanner/bin/sonar-scanner
2 .....
3 INFO: Analysis report uploaded in 965ms
4 INFO: ANALYSIS SUCCESSFUL, you can browse http://192.168.31.118:9000/dashboard?
  id=magedu-python
5 INFO: Note that you will be able to access the updated dashboard once the server has
  processed the submitted analysis report
6 INFO: More about the report processing at http://192.168.31.118:9000/api/ce/task?
  id=AYTdgML6WttZ0mKW6PAZ
7 INFO: Analysis total time: 11.082 s
8 INFO: -----
9 INFO: EXECUTION SUCCESS
10 INFO: -----
11 INFO: Total time: 13.567s
12 INFO: Final Memory: 7M/34M
13 INFO: -----
```



5.2. 基于shell

```
1 #配置shell脚本
```

Dashboard > jinliudev-app2 >

Configuration

General

源码管理

构建触发器

构建环境

Build Steps

构建后操作

Add timestamps to the Console Output

Inspect build log for published build scans

Setup Kubernetes CLI (kubectl) ?

Terminate a build if it's stuck

With Ant ?

Build Steps

执行 shell ?

命令

查看 可用的环境变量列表

cd /data/gitdata/jinliu/
rm -rf app2
git clone git@192.168.31.121:jinliudev/app2.git
cd app2

高级...

增加构建步骤

构建后操作

增加构建后操作步骤

保存 应用

Google Translate

Jenkins

查找 (CTRL+K)

1 3 jinliu 注销

Dashboard > jinliudev-app2 > #5

返回到工程

状态

变更记录

控制台输出

文本方式查看

构建输出信息

删除构建 '#5'

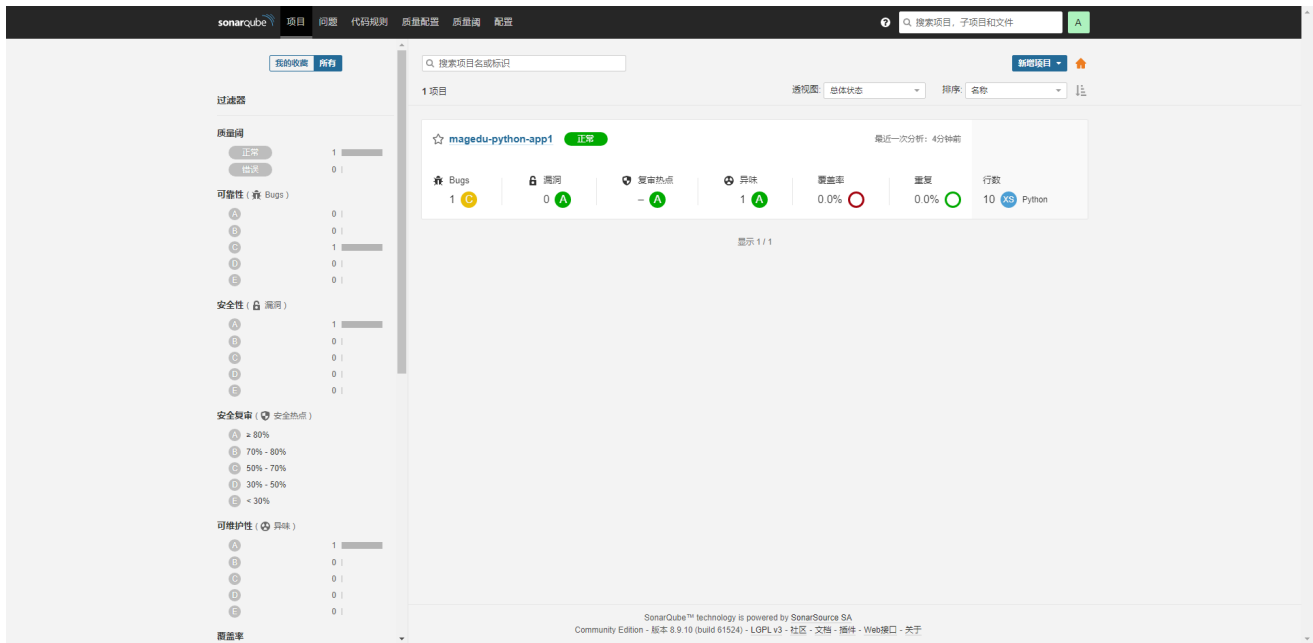
Timings

打开 Blue Ocean

上一次构建

控制台输出

Started by user jinliu
Running as SYSTEM
Building on the built-in node in workspace /var/lib/jenkins/workspace/jinliudev-app2
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[jinliudev-app2] \$ /bin/sh -xe /tmp/jenkins17141033016845700436.sh
+ cd /data/gitdata/jinliu/
+ rm -rf app2
+ git clone git@192.168.31.121:jinliudev/app2.git
Cloning into 'app2'...
+ cd app2
+ /apps/sonar-scanner/bin/sonar-scanner
INFO: Scanner configuration file: /apps/sonar-scanner-4.7.0.2747/conf/sonar-scanner.properties
INFO: Project root configuration file: /data/gitdata/jinliu/app2/sonar-project.properties
INFO: SonarScanner 4.7.0.2747
INFO: Java 11.0.17 Red Hat, Inc. (64-bit)
INFO: Linux 3.10.0-957.el7.x86_64 amd64
INFO: User cache: /root/.sonar/cache
INFO: Scanner configuration file: /apps/sonar-scanner-4.7.0.2747/conf/sonar-scanner.properties
INFO: Project root configuration file: /data/gitdata/jinliu/app2/sonar-project.properties
INFO: Analyzing on SonarQube server 8.9.10
INFO: Default locale: "en_US", source code encoding: "UTF-8"
INFO: Load global settings
INFO: Load global settings (done) | time=69ms
INFO: Server id: 6672D607-AYTdW6nTqrRixKeg_P
INFO: User cache: /root/.sonar/cache
INFO: Load/download plugins
INFO: Load plugins index
INFO: Load plugins index (done) | time=44ms
INFO: Plugin [l10nzh] defines 'l10nen' as base plugin. This metadata can be removed from manifest of l10n plugins since version 5.2.
INFO: Load/download plugins (done) | time=1308ms



- 1 #传参方式，可配置在流水线中
- 2 /apps/sonar-scanner/bin/sonar-scanner -Dsonar.projectKey=magedu -
Dsonar.projectName=magedu-python-app2 -Dsonar.projectVersion=2.0 -Dsonar.sources=./src
-Dsonar.language=py -Dsonar.sourceEncoding=UTF-8

5.3. 基于流水线

```
1 pipeline {
2   agent any
3   parameters {
4     string(name: 'BRANCH', defaultValue: 'develop', description: '分支选择') //字符串参
数，会配置在jenkins的参数化构建过程中
5     choice(name: 'DEPLOY_ENV', choices: ['develop', 'production'], description: '部署
环境选择') //选项参数，会配置在jenkins的参数化构建过程中
6   }
7   stages {
8     stage('变量测试1') {
9       steps {
10        sh "echo $env.WORKSPACE" //JOB的工作目录,可用于后期目录切换
11        sh "echo $env.JOB_URL" //JOB的URL
12        sh "echo $env.NODE_NAME" //节点名称, master 名称显示built-in
13        sh "echo $env.NODE_LABELS" //节点标签
14        sh "echo $env.JENKINS_URL" //jenkins的URL地址
15        sh "echo $env.JENKINS_HOME" //jenkins的家目录路径
16      }
17    }
18    stage("code clone"){
19      steps {
20        deleteDir()
21        script {
22          if ( env.BRANCH == 'main' ) {
23
24            git branch: 'main', credentialsId: '27aaea07-f640-4cdb-896f-
```

```

8248d8b42155', url: 'git@192.168.31.121:jinliudev/app2.git'
24         } else if ( env.BRANCH == 'develop' ) {
25             git branch: 'develop', credentialsId: '27aaea07-f640-4cdb-896f-
8248d8b42155', url: 'git@192.168.31.121:jinliudev/app2.git'
26         } else {
27             echo '您穿的分支参数BRANCH ERROR, 请检查分支参数是否正确'
28         }
29         GIT_COMMIT_TAG = sh(returnStdout: true, script: 'git rev-parse --
short HEAD').trim() //获取clone完成的分支tagid, 用于做镜像tag
30     }
31 }
32 }
33 stage('python源代码质量扫描') {
34     steps {
35         sh "cd $env.WORKSPACE && /apps/sonar-scanner/bin/sonar-scanner -
Dsonar.projectKey=magedu -Dsonar.projectName=jinliu-pipeline-app1 -
Dsonar.projectVersion=1.0 -Dsonar.sources=./src -Dsonar.language=py -
Dsonar.sourceEncoding=UTF-8"
36     }
37 }
38 }
39 }

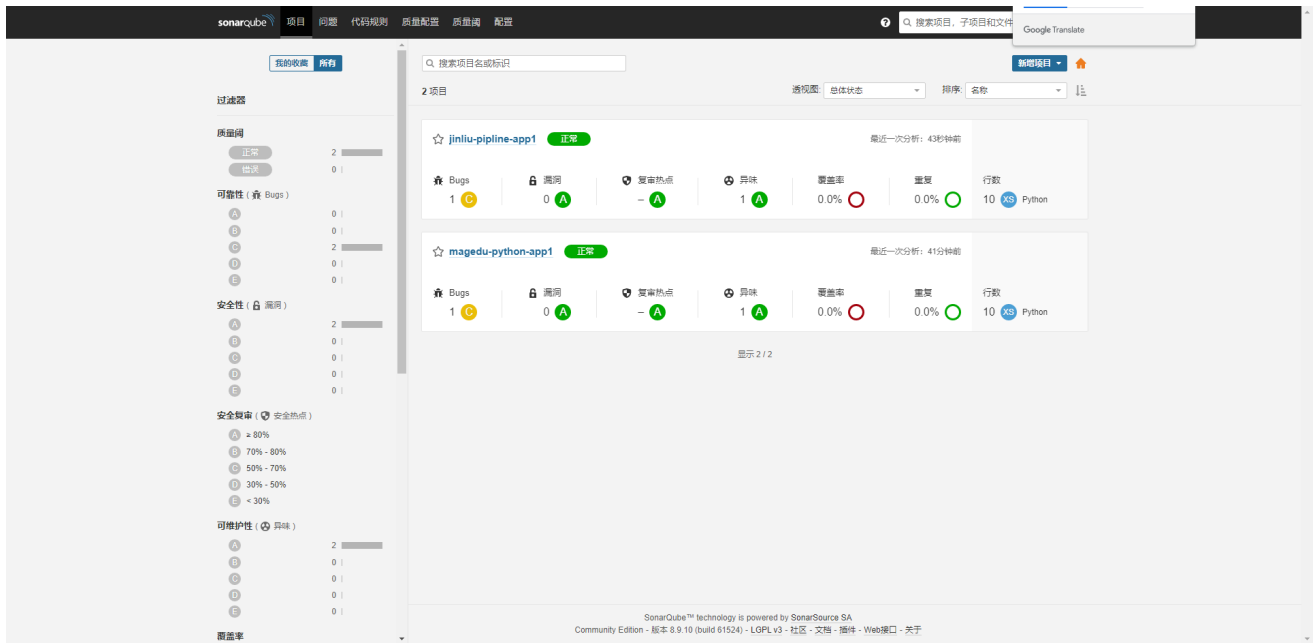
```

The screenshot shows the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search bar, and user information. The main content area displays the 'Console Output' for a pipeline named 'pipeline-sonarqube-test'. The output shows the pipeline starting, cloning the repository, and running a SonarQube scan. The left sidebar contains various navigation options like 'Dashboard', 'pipeline-sonarqube-test', 'Status', 'Change History', 'Console Output', 'View as plain text', 'Build Information', 'Parameters', 'Timings', 'Git Build Data', 'Open Blue Ocean', 'Thread Dump', 'Pause (Resume)', 'Replay', 'Flow Diagram', 'Workspaces', and 'Previous Builds'.

```

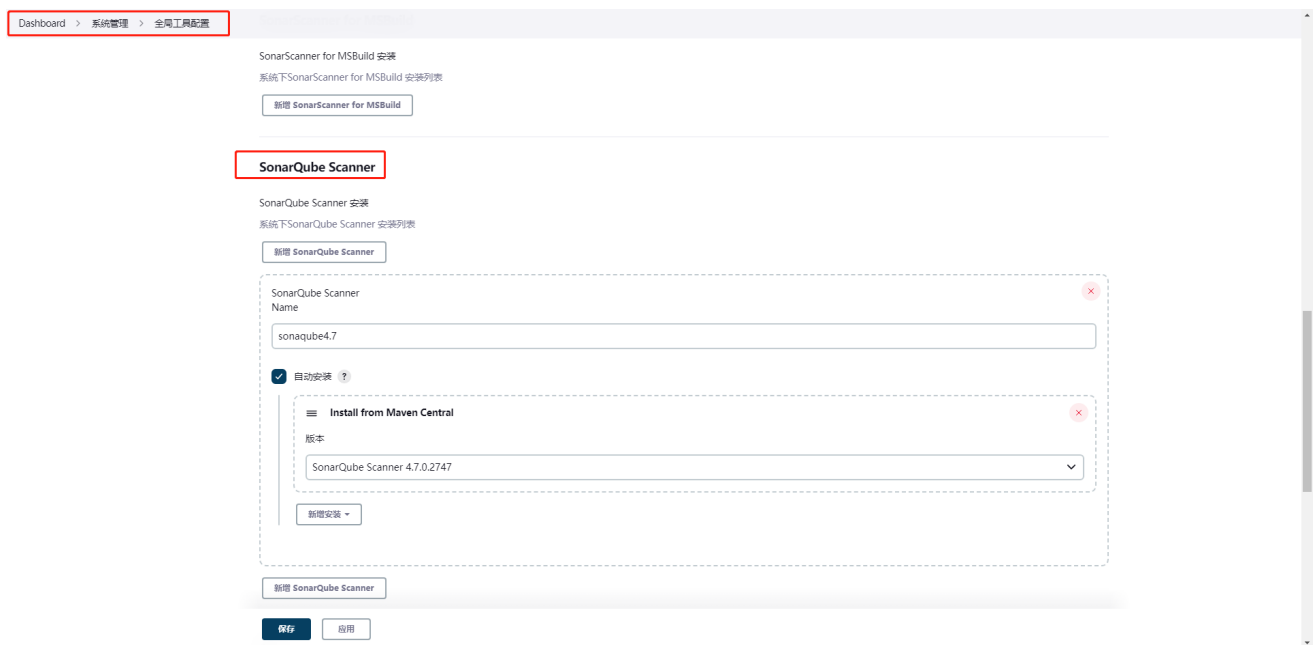
Started by user jinliu
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/pipeline-sonarqube-test
[Pipeline] {
[Pipeline] stage
[Pipeline] { (克隆测试1)
[Pipeline] sh
+ echo /var/lib/jenkins/workspace/pipeline-sonarqube-test
/var/lib/jenkins/workspace/pipeline-sonarqube-test
[Pipeline] sh
+ echo http://192.168.31.127:8080/job/pipeline-sonarqube-test/
http://192.168.31.127:8080/job/pipeline-sonarqube-test/
[Pipeline] sh
+ echo built-in
built-in
[Pipeline] sh
+ echo built-in
built-in
[Pipeline] sh
+ echo http://192.168.31.127:8080/
http://192.168.31.127:8080/
[Pipeline] sh
+ echo /var/lib/jenkins
/var/lib/jenkins
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (code clone)
[Pipeline] deleteDir
[Pipeline] script
[Pipeline] {

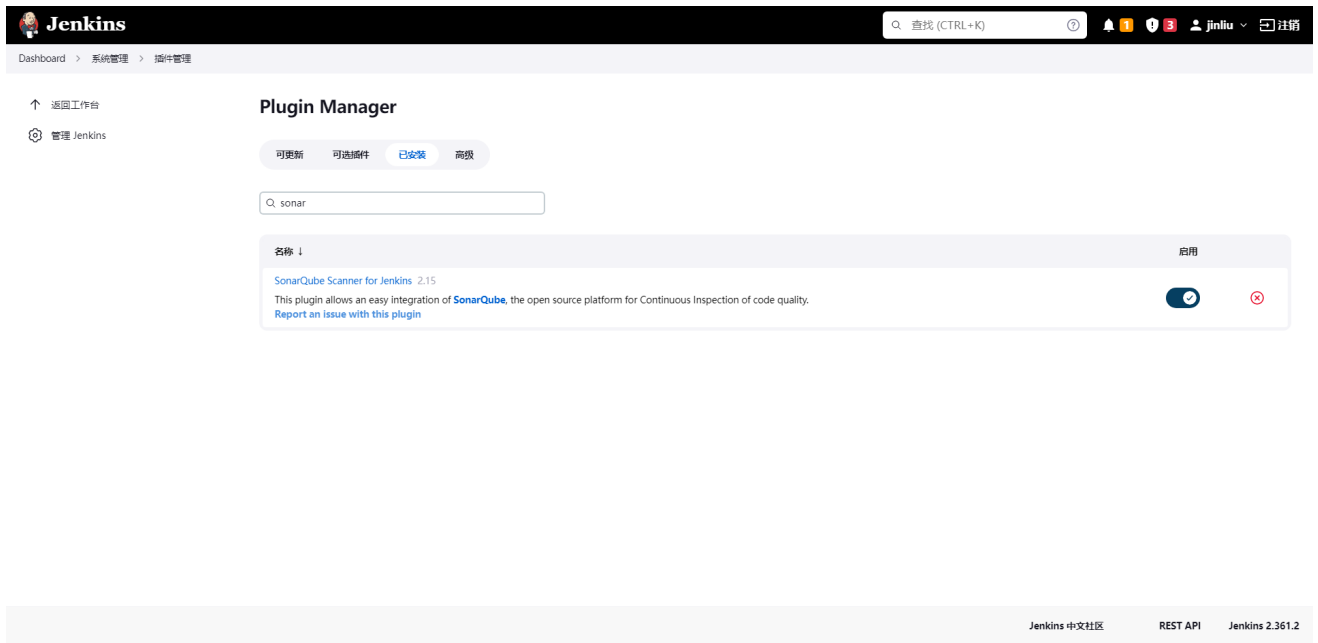
```

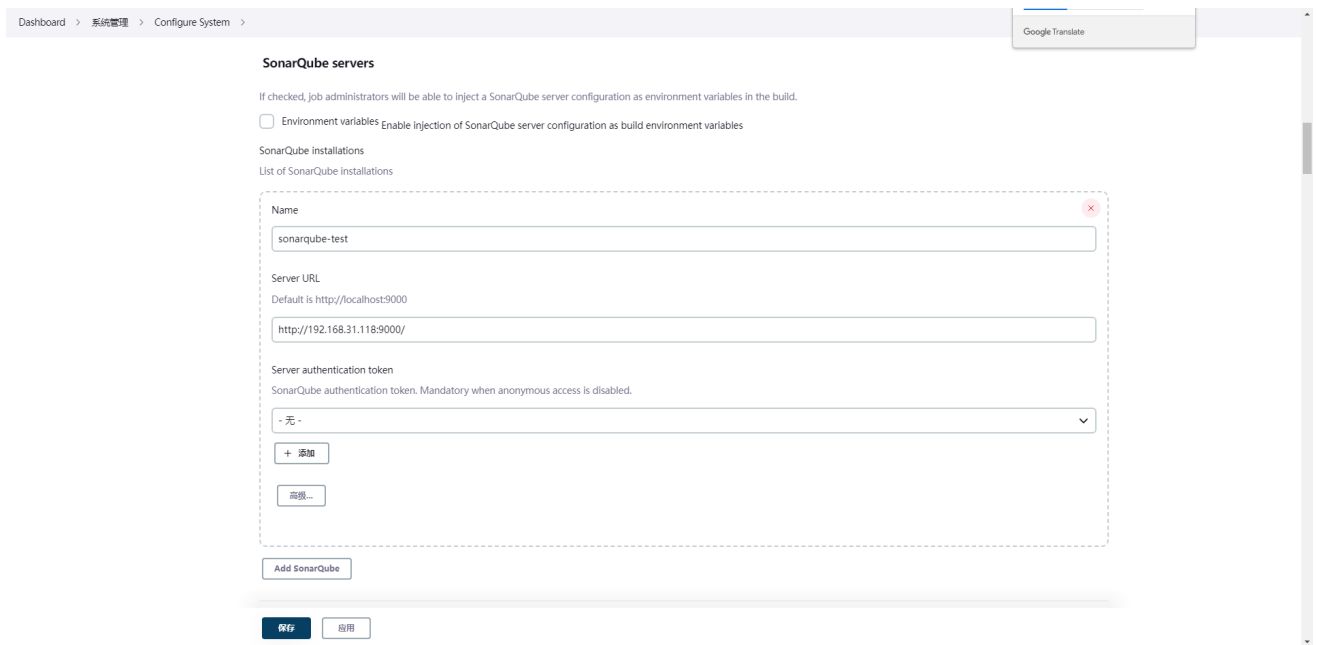
6. jenkins 安装 Sonarqube Scanner 插件、配置 sonarqube server 地址、基于 jenkins 配置代码扫描参数实现代码质量扫描

6.1. jenkins 安装 Sonarqube Scanner 插件





6.2. 配置 sonarqube server 地址



6.3. 基于jeknins-sonarqube代码扫描

Dashboard > sonarqube-in-jenkins >

Google Translate

Configuration

General

源码管理

构建触发器

构建环境

Build Steps

构建后操作

Task to run ?

JDK ?

JDK to be used for this SonarQube analysis

(Inherit From Job)

Path to project properties ?

Analysis properties ?

#Configure here general information about the environment, such as SonarQube server connection details for example
#No information about specific project should appear here

#----- Default SonarQube server
sonar.host.url=http://192.168.31.118:9000

#----- Default source code encoding
sonar.sourceEncoding=UTF-8

Additional arguments ?

JVM Options ?

执行 shell ?

保存 应用

Jenkins

查找 (CTRL+K)

Google Translate

jiniu 注销

Dashboard > sonarqube-in-jenkins > #4

↑ 返回到工程

📄 状态集

</> 变更记录

📄 控制台输出

📄 文本方式查看

📄 终端侧滑信息

🗑️ 删除构建 '#4'

🕒 Timings

📄 Git Build Data

🌐 打开 Blue Ocean

← 上一次构建

✅ 控制台输出

Started by user jiniu
Running as SYSTEM
Building on the built-in node in workspace /var/lib/jenkins/workspace/sonarqube-in-jenkins
The recommended git tool is: NONE
using credential 400177f2-s576-4411-b989-09a0220b9a81
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/sonarqube-in-jenkins/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url git@192.168.31.121:jiniudev/app2.git # timeout=10
Fetching upstream changes from git@192.168.31.121:jiniudev/app2.git
> git --version # timeout=10
> git --version # 'git version 1.8.3.1'
using GIT_ASKPASS to set credentials root-ssh-passwd
> git fetch --tags --progress git@192.168.31.121:jiniudev/app2.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 2a8d79baea2a7c7e2927f9f5d3cd499f56f72739 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 2a8d79baea2a7c7e2927f9f5d3cd499f56f72739 # timeout=10
Commit message: "add python src"
> git rev-list --no-walk 2a8d79baea2a7c7e2927f9f5d3cd499f56f72739 # timeout=10
Unpacking https://repo1.maven.org/maven2/org/sonarsource/scanner/cli/sonar-scanner-cli/4.7.0.2747/sonar-scanner-cli-4.7.0.2747.zip to /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube4.7 on Jenkins
[sonarqube-in-jenkins] \$ /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube4.7/bin/sonar-scanner -Dsonar.host.url=http://192.168.31.118:9000/ -Dsonar.sourceEncoding=UTF-8 -
WARN: Property 'sonar.host.url' with value 'http://192.168.31.118:9000/' is overridden with value 'http://192.168.31.118:9000'
INFO: Scanner configuration file: /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube4.7/conf/sonar-scanner.properties
INFO: Project root configuration file: /var/lib/jenkins/workspace/sonarqube-in-jenkins/sonar-project.properties
INFO: SonarScanner 4.7.0.2747
INFO: Java 11.0.17 Red Hat, Inc. (64-bit)
INFO: Linux 3.10.0-957.el7.x86_64 amd64
INFO: User cache: /root/.sonar/cache
INFO: Scanner configuration file: /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube4.7/conf/sonar-scanner.properties
INFO: Project root configuration file: /var/lib/jenkins/workspace/sonarqube-in-jenkins/sonar-project.properties

sonarqube

项目问题代码规则质量配置质量图配置

搜索项目、子项目和文件

Google Translate

我的收藏所有

过滤器

质量图

正常2

错误0

可靠性 (Bugs)

A0

B0

C2

D0

E0

安全性 (漏洞)

A2

B0

C0

D0

E0

安全策略 (安全热点)

A> 80%

B70% - 80%

C50% - 70%

D30% - 50%

E< 30%

可维护性 (异味)

A2

B0

C0

D0

E0

覆盖率

搜索项目名称或标识

2 项目

透视图总体状态

排序名称

刷新项目

☆ jinliu-pipeline-app1 正常

最近一次分析: 35分钟前

Bugs	漏洞	警告热点	异味	覆盖率	质量	行数
1	0	-	1	0.0%	0.0%	10 Python

☆ magedu-python-app1 正常

最近一次分析: 7分钟前

Bugs	漏洞	警告热点	异味	覆盖率	质量	行数
1	0	-	1	0.0%	0.0%	10 Python

显示 2 / 2

SonarQube™ technology is powered by SonarSource SA

Community Edition - 版本 8.9.10 (build 61524) - LGPL v3 - 社区 - 文档 - 插件 - Web接口 - 关于