Jenkins and SonarQube

|  |  |
| --- | --- |
| Author: | Ian Addison |
| Document Name: | Jenkins and SonarQube |
| Document Location: | \\lukpc210\Projects\Product Projects\Operations\Vitic 2017.1\Project Documents\POC\Jenkins and SonarQube.docx |
|  |
| Date Created: | 30/01/2017 |
| Date Last Modified: | 16/03/2017 |
| Status: | Draft |
| Confidentiality: | <Select/enter confidentiality status> |
| Circulation: | <Detail circulation list> |
| Document Version: | 0.5 |

Contents

[1. Overview 2](#_Toc477768805)

[2. Jenkins 3](#_Toc477768806)

[2.1 Vitic\_POC\_Back\_End\_Branch-0.0.1.-SCM 3](#_Toc477768807)

[2.2 Vitic\_POC\_Front\_End-Branch-0.0.1-SCM 4](#_Toc477768808)

[2.3 Vitic\_POC\_Master-0.0.1-Pipeline 5](#_Toc477768809)

[3. SonarQube 8](#_Toc477768810)

# Overview

This document details the working configurations and lessons learned regarding Jenkins and SonarQube (with reference to Git) throughout the Vitic PoC project.

# Jenkins

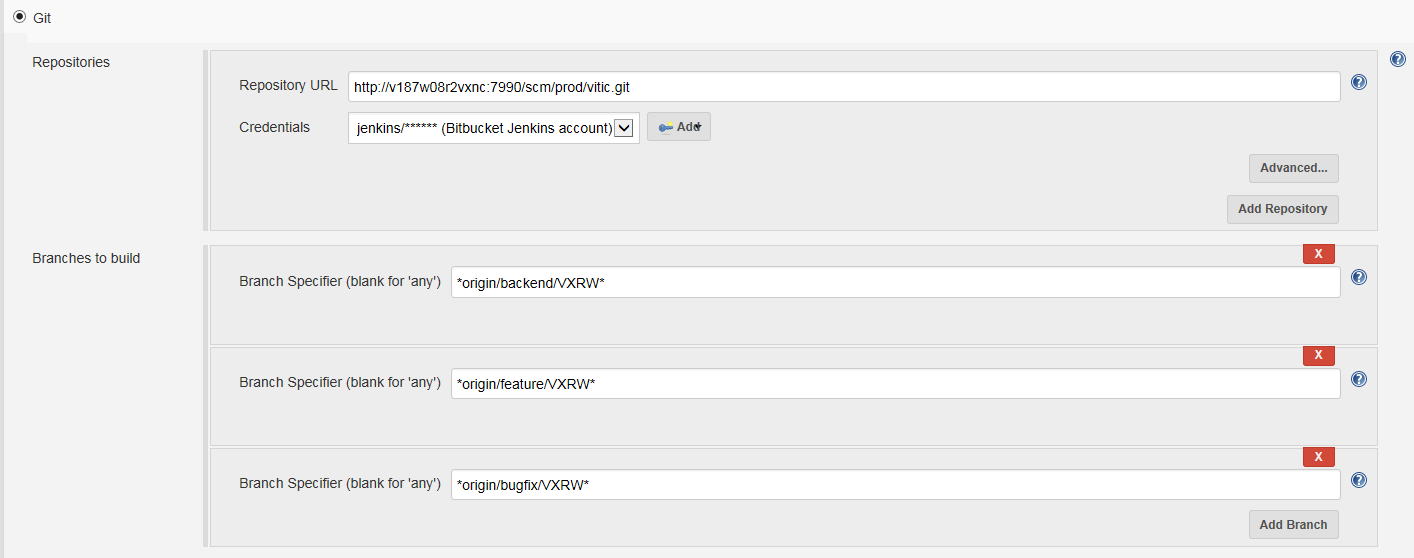
Although there is already an instance of Jenkins in Lhasa, a separate instance was installed for the Vitic PoC project. This was installed as a vanilla installation with the default set of plugins. Additional plugins that were installed afterwards were:

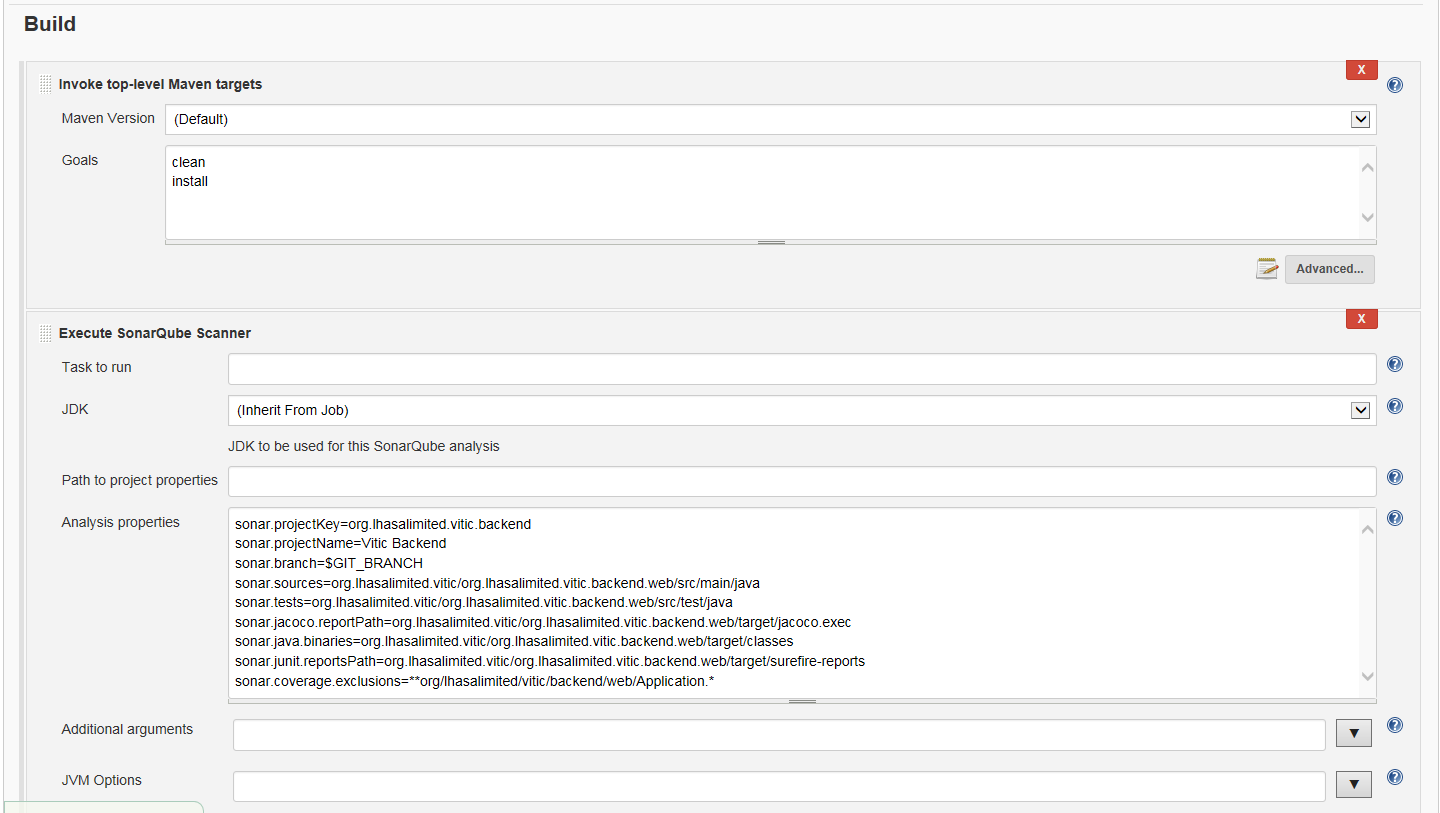
* NodeJS Plugin: Required to run the Angular 2 unit tests
* SonarQube Plugin: Required for SonarQube integration

Three jobs have been configured in Jenkins which are able to perform a build from a Git repository. The first two are standard jobs and the third is a pipeline job.

## 2.1 Vitic\_POC\_Back\_End\_Branch-0.0.1.-SCM

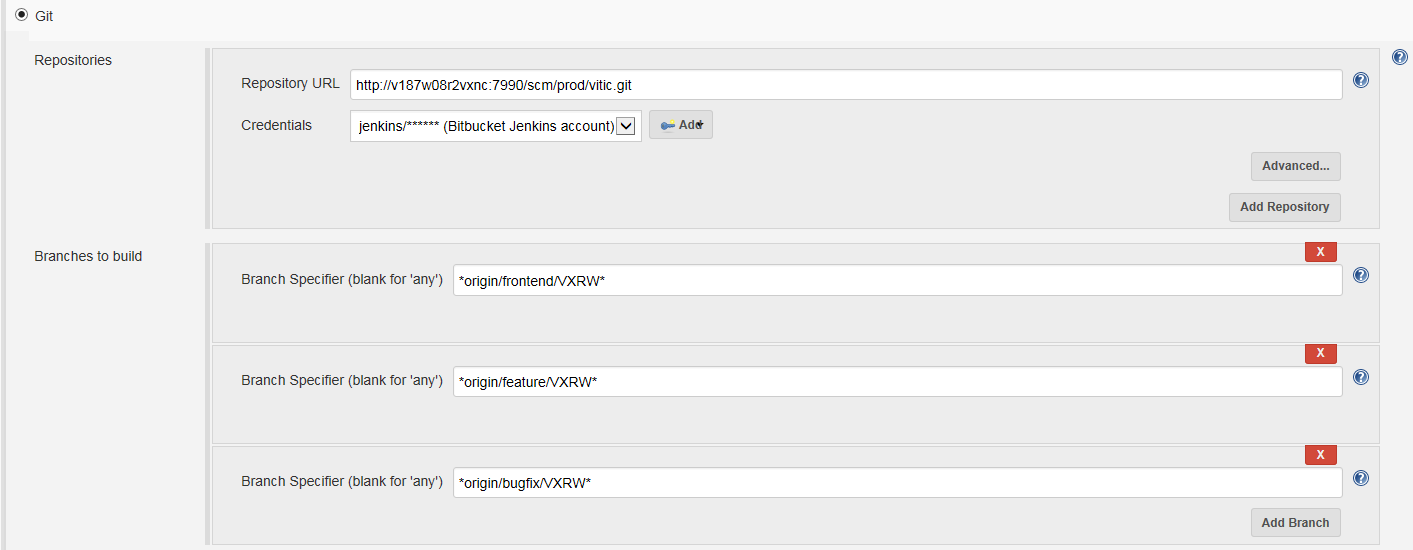
Used to perform a build of the backend project on any branch (including master). The “Branch Specifier” is used to determine which branches this job will build. All branches include the number of the Jira ticket. The naming convention adopted for branches in the Vitic PoC project is as follows:

* backend/VXRW-XX: Branch for work done on ticket VXRW-XX. This ticket only involves backend changes
* frontend/VXRW-XX: Branch for work done on ticket VXRW-XX. This ticket only involves frontend changes
* feature/VXRW-XX: Branch for work done on ticket VXRW-XX. This ticket involves both backend and frontend changes
* bugfix/VXRW-XX: Branch for work done on VXRW-XX. This ticket is a bug

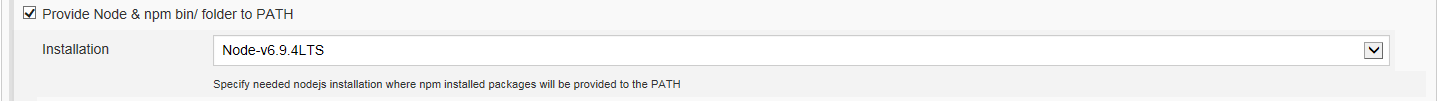
The Git configuration above will build any branches named backend/VXRW\* or feature/VXRW\* or bugfix/VXRW\*.

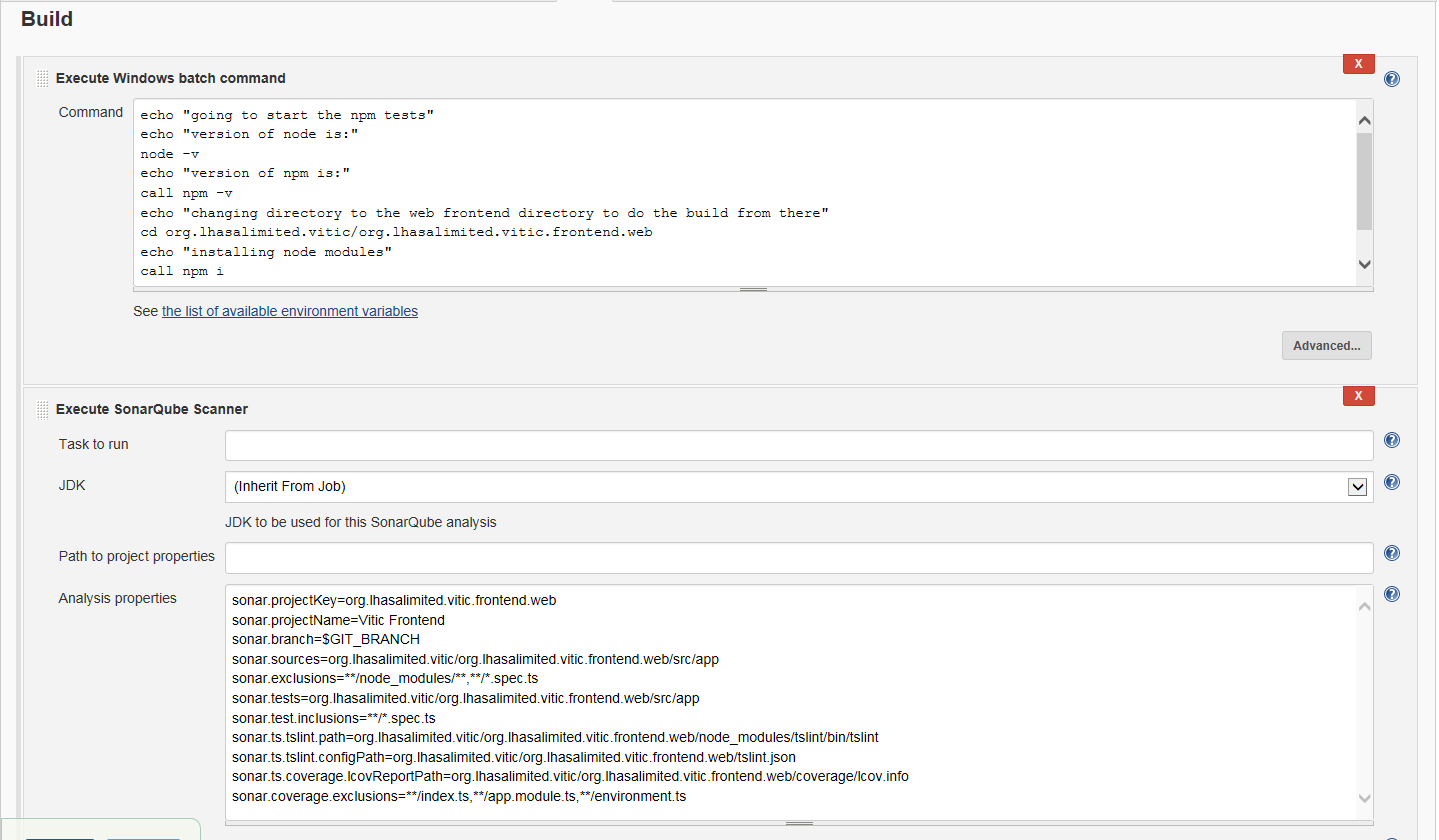
The build configuration above invokes mvn clean install to build the project and run the tests. Appendix C details how to configure the SonarQube Scanner to publish analysis and test coverage results to SonarQube.

## 2.2 Vitic\_POC\_Front\_End-Branch-0.0.1-SCM

Used to perform a build of the frontend project on any branch (including master). The naming convention adopted for branches in the Vitic PoC project is explained in section 2.1.

The Git configuration above will build any branches named frontend/VXRW\* or feature/VXRW\* or bugfix/VXRW\*.



The above build environment and configuration uses Node and NPM (Provide Node & npm bin/ folder to PATH) then runs the following batch script to exectute the Jasmine tests:

echo "going to start the npm tests"

echo "version of node is:"

node -v

echo "version of npm is:"

call npm -v

echo "changing directory to the web frontend directory to do the build from there"

cd org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web

echo "installing node modules"

call npm i

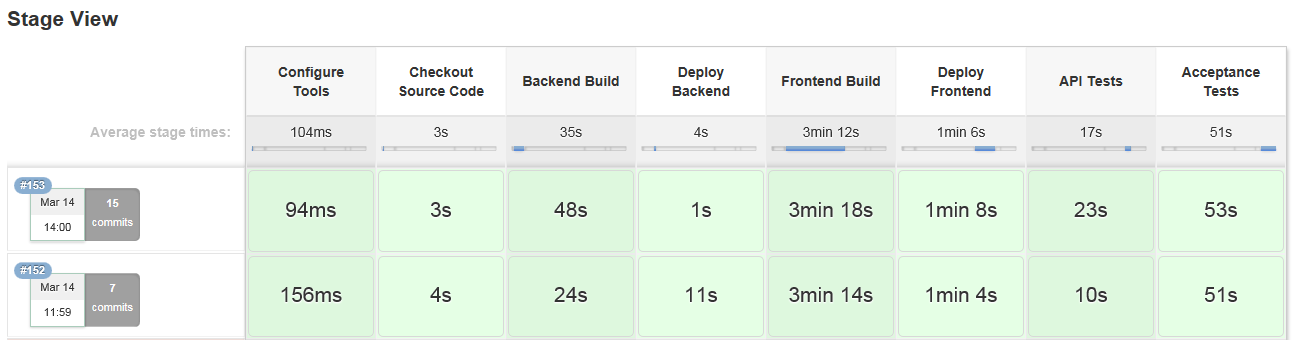
echo "running node tests"

call npm run test-jenkins

Appendix D details how to configure the SonarQube scanner to publish analysis and test coverage results to SonarQube.

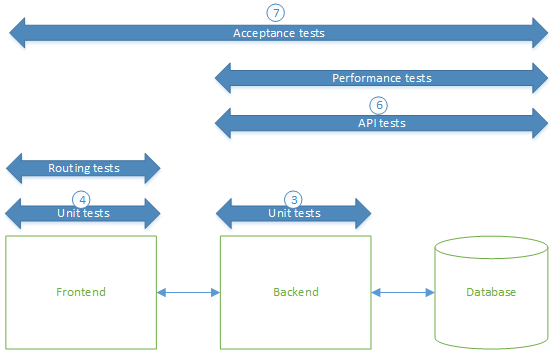
## 2.3 Vitic\_POC\_Master-0.0.1-Pipeline

This job uses a Jenkins Pipeline script in order to provide a continuous delivery pipeline. The Stage view shows each step of the pipeline and the time taken at each step. If the build was to fail at a particular step, the step would be marked in red (rather than green).

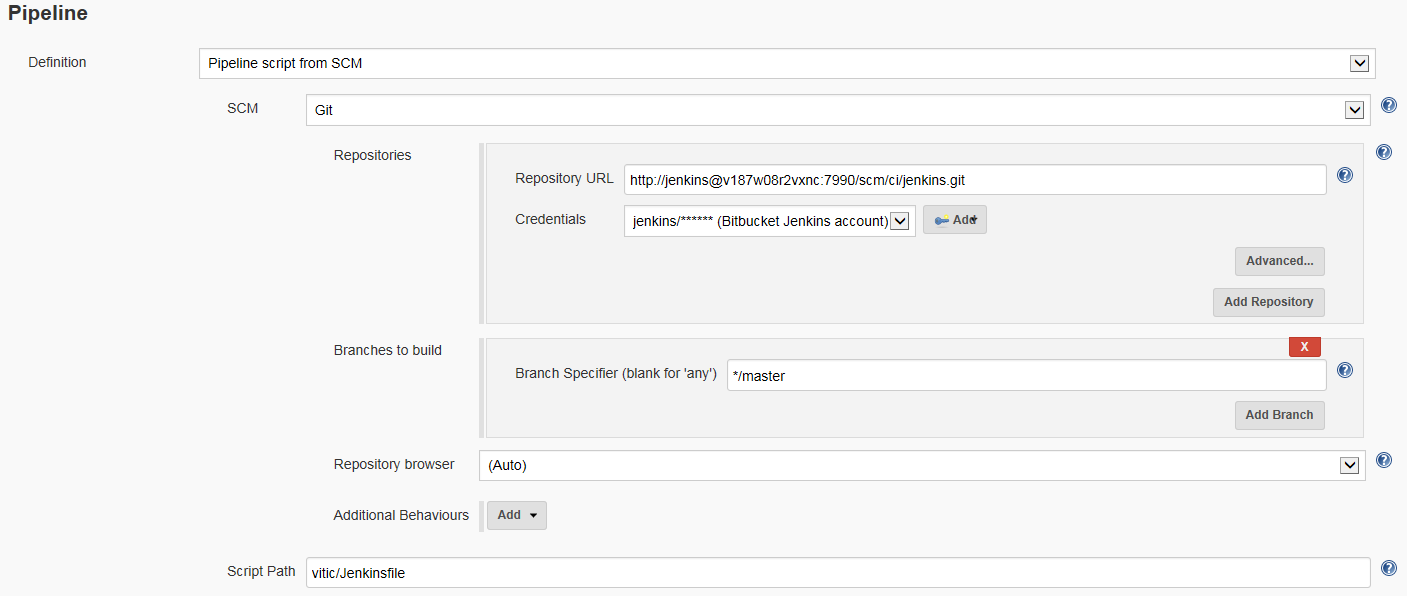


A description of the steps are:

* Configure Tools: Any configuration work that is required (e.g. ensuring Node and NPM are added to PATH)
* Checkout Source Code: Check the project source code out from Git
* Backend Build: Perform the build (and JUnit tests) of the backend project
* Deploy Backend: This deploys the backend (built in the previous step) to the development server and starts it up
* Frontend Build: Perform the build (and Jasmine tests) of the frontend project
* Deploy Frontend: This deploys the frontend (built in the previous step) to the development server and starts it up
* API Tests: Run the API level tests
* Acceptance Tests: Run the acceptance tests (Selenium)

The diagram below shows which tests get executed at each stage of the pipeline.

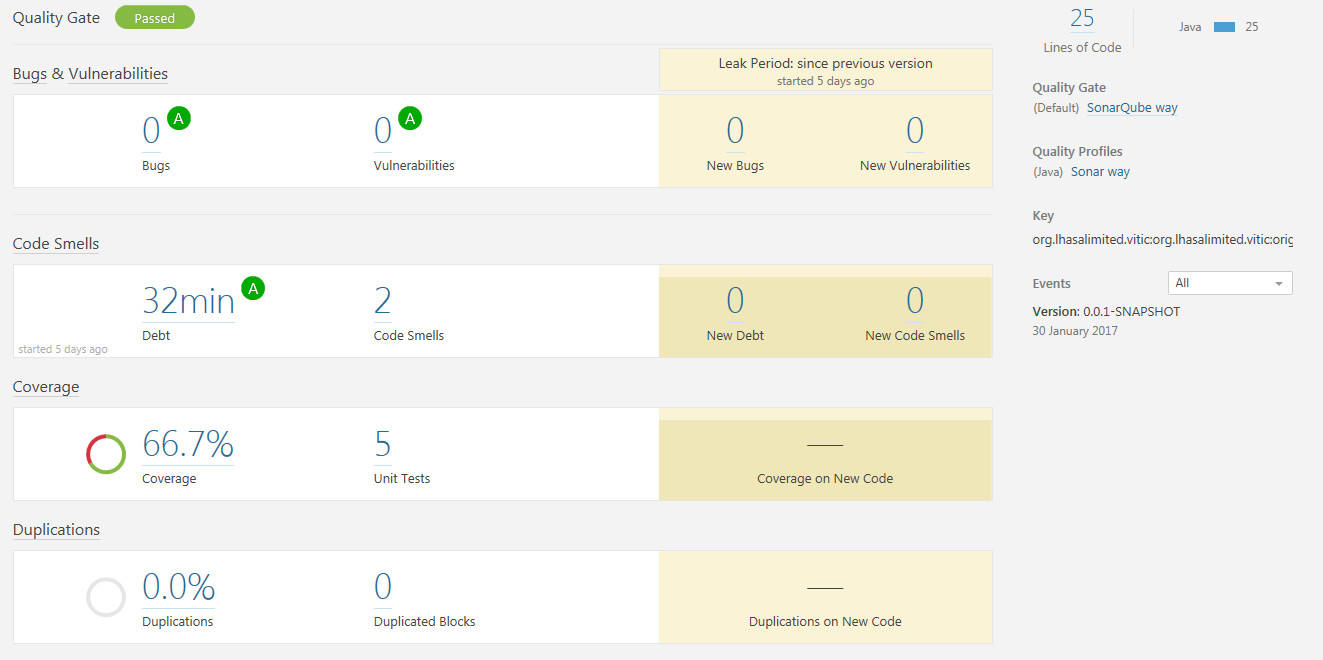
Pipeline scripts can either be defined directly in Jenkins, or stored in a source code repository and referenced from the Jenkins job. The Vitic PoC project has adopted the latter so that the scripts are versioned. The configuration to connect to Git and checkout the pipeline script can be seen below:



Rather than housing the pipeline scripts with the source code (as this would introduce a dependency on Jenkins), a new Git repository has been created (named Jenkins) to house any pipeline scripts. This separates the project source code from any build configuration and as a result should make it easier to migrate from Jenkins to another build tool if required.

The contents of the pipeline script (Jenkinsfile) can be found in Appendix A.

# SonarQube

SonarQube has also been installed (installation notes can be found in Appendix B) so that static analysis can be performed on the source code and the test coverage statistics (from JaCoCo) can be viewed:

A guide for configuring a Backend Jenkins job with respect to SonarQube can be found in Appendix C. This configuration will ensure the JaCoCo report is ran as part of the Maven build. There are two ways of achieving SonarQube integration:

1. For Maven projects you can add a build step of **Invoke top-level Maven target** and specify sonar:sonar –Dsonar.branch=$GIT\_BRANCH under the step goals.
2. For any project you can add a build step of **Execute SonarQube Scanner** and specify the required sonar properties (this is the approach detailed in Appendix C):

Both methods with have the same effect of publishing the JaCoCo report to SonarQube and performing static analysis on the code. As $GIT\_BRANCH is used, the results will appear in Sonar under a project with the same name as the branch in Git. This allows a Sonar run to be referenced in a pull request for peer review.

The preferred way to do this in Lhasa is the second option. The reason for this is that not all projects are Maven projects and although there will be more configuration for a Maven project using this approach, it means all Jenkins jobs will have a consistent SonarQube configuration. SonarQube can also display results of static analysis on Angular 2 (Typescript) code and test coverage reports (LCOV) for Javascript code. A guide for configuring a Front Jenkins job with respect to SonarQube can be found in Appendix D. By using the second option it means Backend and Frontend projects are configured in the same way with respect to SonarQube in Jenkins.

For SonarQube to recognise Typescript files, ensure that the Sonar TS Plugin (<https://github.com/Pablissimo/SonarTsPlugin>) is installed in SonarQube. This can be done as follows:

1. Download latest version of Sonar TS Plugin from <https://github.com/Pablissimo/SonarTsPlugin> (we used v1.0.0)
2. Install into SonarQube by placing the downloaded JAR file in %SONAR\_QUBE%\extensions\plugins directory and restarting SonarQube

APPENDIX A: Jenkins Pipeline Script

try {

node {

stage('Configure Tools') {

// Add Node (including NPM) to the path

def nodeHome = tool 'Node-v6.9.4LTS'

env.PATH = "${env.PATH};${nodeHome}/bin"

}

stage('Checkout Source Code') {

// Checkout source code from Git

checkout([

$class: 'GitSCM',

branches: [[name: '\*/master']], // master branch

doGenerateSubmoduleConfigurations: false,

extensions: [],

submoduleCfg: [],

userRemoteConfigs: [[credentialsId: 'Bitbucket', url: 'http://jenkins@v187w08r2vxnc:7990/scm/prod/vitic.git']] // vitic repository

])

}

stage('Backend Build') {

// Run the maven build

bat '''

cd org.lhasalimited.vitic/org.lhasalimited.vitic.backend.web

mvn clean install

'''

}

stage('Frontend Build') {

// Run the frontend tests

bat '''

echo "going to start the npm tests"

echo "version of node is:"

node -v

echo "version of npm is:"

call npm -v

echo "changing directory to the web frontend directory to do the build from there"

cd org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web

echo "installing node modules"

call npm i

echo "running node tests"

call npm run test-jenkins

'''

}

}

} catch (error) {

def emailTo = 'paul.sehgal@lhasalimited.org ian.addison@lhasalimited.org richard.stevenson@lhasalimited.org fiona.mcDonald@lhasalimited.org mohammed.sultan@lhasalimited.org'

mail body: "Vitic pipeline job failed, please see $JOB\_URL" ,

from: 'jenkins@lhasalimited.org',

subject: 'Vitic Pipeline Job Failed',

to: emailTo

throw error

}

APPENDIX B: SonarQube Installation Notes

**SonarQube Configuration**

1. Download SonarQube 6.2 from <https://www.sonarqube.org/downloads/>
2. Unzip to c:\ on v187w08r2vxnc
3. Open sonar.properties in C:\sonarqube-6.2\conf and set:
   * sonar.jdbc.username=sonarqube
   * sonar.jdbc.password=sonarqube

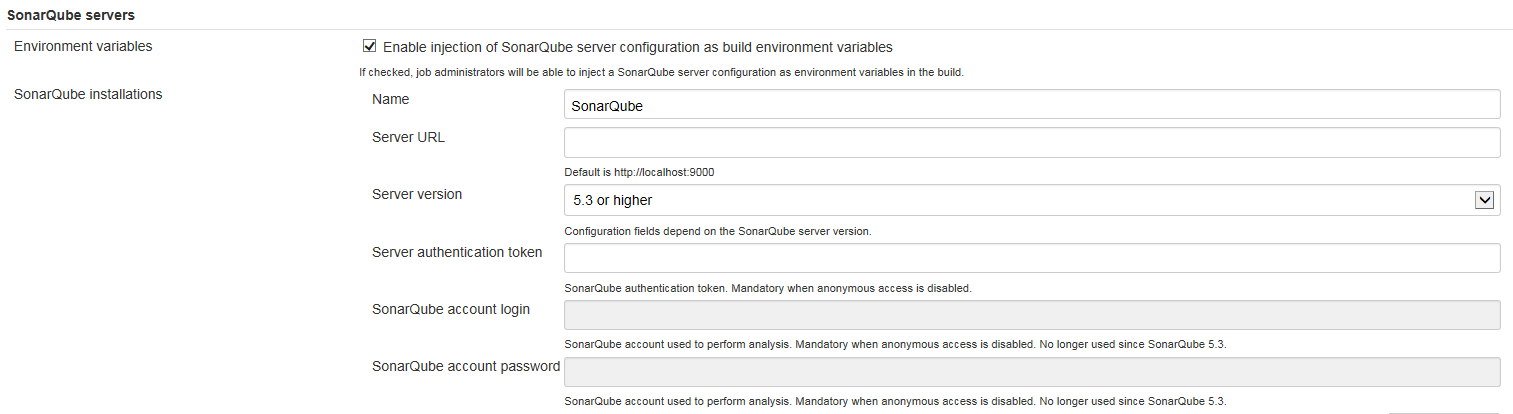
*Note: by not specifying sonar.jdbc.url, SonarQube will use the embedded database*

1. Open a command prompt and navigate to C:\sonarqube-6.2\bin\windows-x86-64
2. Run InstallNTService.bat
3. Open Windows services and start the SonarQube service

SonarQube will be available at http:// v187w08r2vxnc:9000

**Jenkins Configuration**

1. Go to **Manage Jenkins** > **Manage Plugins** and install the SonarQube plugin
2. Go to **Manage Jenkins** > **Configure System** and add a SonarQube server as follows:



*Note: the Server URL is left blank as it defaults to localhost:9000 (which is where SonarQube is installed)*

APPENDIX C: Backend Jenkins Job Configuration

1. In the projects POM file, add the following plugin under <build><plugins>:

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.7.8</version>

<executions>

<execution>

<id>default-prepare-agent</id>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>default-report</id>

<phase>prepare-package</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

<execution>

<id>default-check</id>

<goals>

<goal>check</goal>

</goals>

<configuration>

<rules></rules>

</configuration>

</execution>

</executions>

</plugin>

1. When creating the Jenkins build, under **Build Environment**, ensure **Prepare SonarQube Scanner environment** is checked
2. Add a **build step** of type **Execute SonarQube Scanner**
3. In the **Analysis Properties** field enter the following (using $GIT\_BRANCH ensures there will be a project in SonarQube relating to the branch being built by the Jenkins job):
   * sonar.projectKey=org.lhasalimited.vitic.backend
   * sonar.projectName=Vitic Backend
   * sonar.branch=$GIT\_BRANCH
   * sonar.sources= org.lhasalimited.vitic/org.lhasalimited.vitic.backend.web/src/main/java
   * sonar.tests= org.lhasalimited.vitic/org.lhasalimited.vitic.backend.web/src/test/java
   * sonar.jacoco.reportPath= org.lhasalimited.vitic/org.lhasalimited.vitic.backend.web/target/jacoco.exec
   * sonar.java.binaries= org.lhasalimited.vitic/org.lhasalimited.vitic.backend.web/target/classes
   * sonar.junit.reportsPath= org.lhasalimited.vitic/org.lhasalimited.vitic.backend.web/target/surefire-reports
4. Run the build. The project should be visible in Sonar along with test coverage statistics

APPENDIX D: Frontend Jenkins Job Configuration

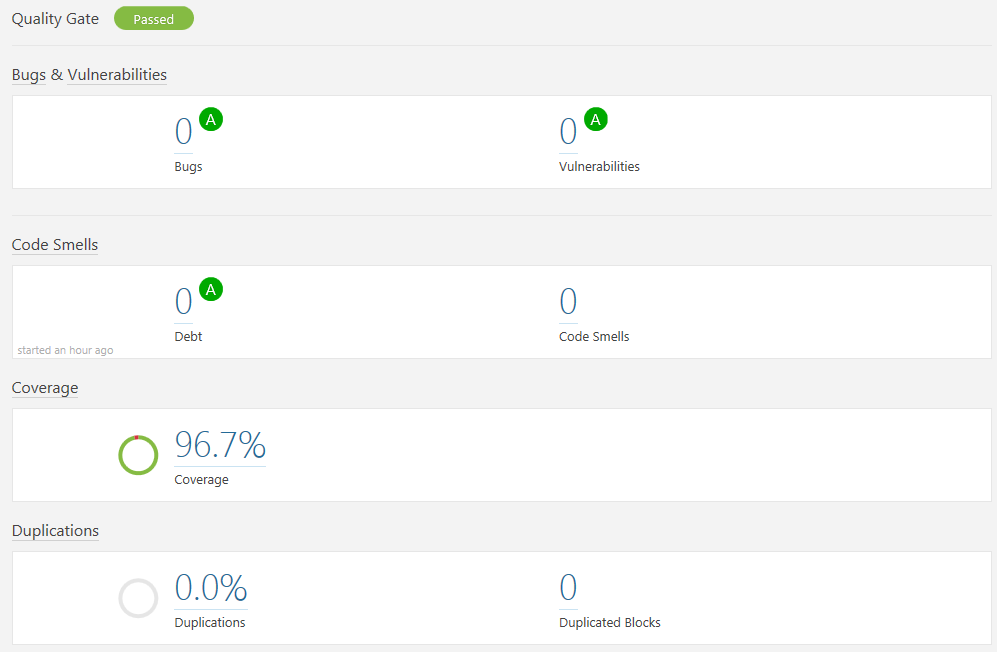
1. In conf/karma.config ensure the coverageReporter is configured as follows (the important part is to ensure the LCOV file is generated):

coverageReporter: {  
 **type**: **'in-memory'**,  
 **reporters**: [  
 { **type**: **'in-memory'** },  
 { **type**: **'lcovonly'**, **dir**: **'coverage'**, **subdir**: **'.'** },  
 ]  
},

1. When creating the Jenkins build, under **Build Environment**, ensure **Prepare SonarQube Scanner environment** is checked
2. Add a **build step** of type **Execute SonarQube Scanner**
3. In the **Analysis Properties** field enter the following (using $GIT\_BRANCH ensures there will be a project in SonarQube relating to the branch being built by the Jenkins job):

* sonar.projectKey=org.lhasalimited.vitic.frontend
* sonar.projectName=Vitic Frontend
* sonar.branch=$GIT\_BRANCH
* sonar.sources=org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web/src/app
* sonar.exclusions=\*\*/node\_modules/\*\*,\*\*/\*.spec.ts
* sonar.tests=org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web/src/app
* sonar.test.inclusions=\*\*/\*.spec.ts
* sonar.ts.tslint.path=org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web/node\_modules/tslint/bin/tslint
* sonar.ts.tslint.configPath=org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web/tslint.json
* sonar.ts.coverage.lcovReportPath=org.lhasalimited.vitic/org.lhasalimited.vitic.frontend.web/coverage/lcov.info
* sonar.coverage.exclusions=\*\*/index.ts,\*\*/app.module.ts,\*\*/environment.ts

5. Run the build. The project should be visible in Sonar along with test coverage statistics:

When the Jenkins Frontend job runs it will publish the tslint configuration to SonarQube so SonarQube can perform the analysis and display the results in the project. It will also publish the LCOV (test coverage) output file to SonarQube so test coverage reports can be displayed. Please note that as npm test invokes tslint (this is ran as part of the Jenkins Frontend build) we should never see the scenario where linting does not pass as this would fail the Jenkins build before the SonarQube Scanner is invoked.

Amendment Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Amendment Details | Changed/Approved by |
| 30-Jan-2017 | 0.1 | Initial version | IA |
| 03-Feb-2017 | 0.2 | Use SonarQube Scanner instead of a “mvn sonar:sonar” | IA |
| 03-Feb-2017 | 0.3 | Updated document with new repository structure | IA |
| 14-Mar-2017 | 0.4 | Updated document based on new job configuration | IA |
| 16-Mar-2017 | 0.5 | Added frontend SonarQube integration notes | IA |