**SonarQube Scans in Jenkins Declarative Pipeline using SonarScanner**

There are different ways to configure and run SonarQube scans, besides; there are various ways to configure and run Jenkins pipelines. That makes explaining how to do it a bit more complicated. Not because it is difficult but because of there a lot of different combinations.

There are a few ways to run the SonarQubeScans:

* Maven
* Gradle
* Manually, using the SonarScanner

You can use multiple ways to define a Jenkins pipeline:

* Manual configuration in the GUI
* DSL script
* Jenkinsfile with a scripted pipeline
* Jenkinsfile with a declarative pipeline

On top of that, there are two ways to configure the scanner:

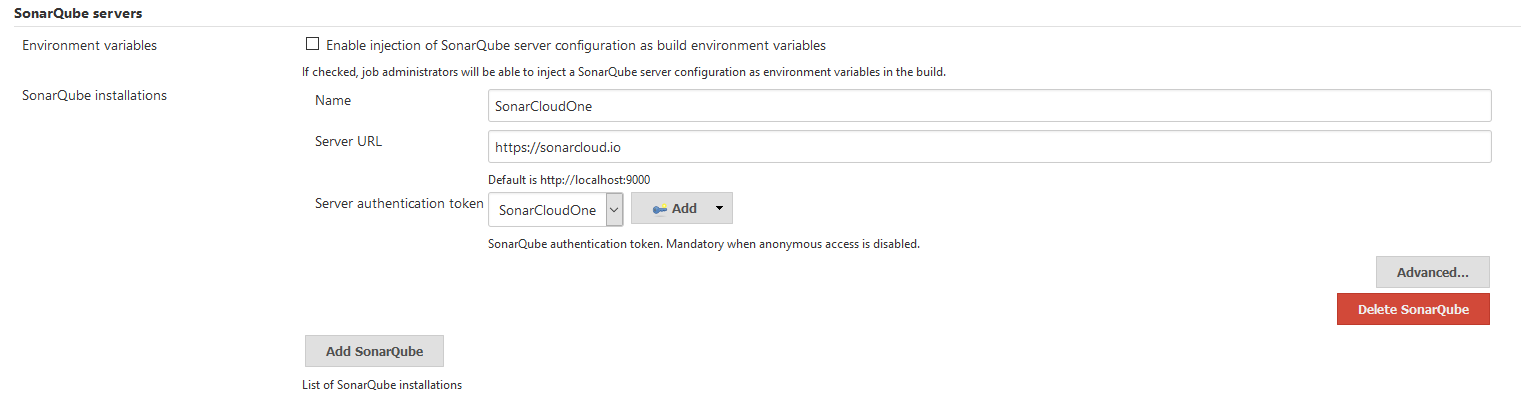
* Using sonar-project.properties
* By sending parameters to the SonarScanner

When I set out to do this configuration, my goal was to minimize the impact of the scanning process on the existing project I wanted to scan. I didn't want to add new files or plugins. I already had a Jenkinsfile, so I tried to concentrate all my changes there. That left me with the option of using the SonnarScanner with parameters passed directly to it. The Jenkinsfile already had a basic declarative pipeline defined.

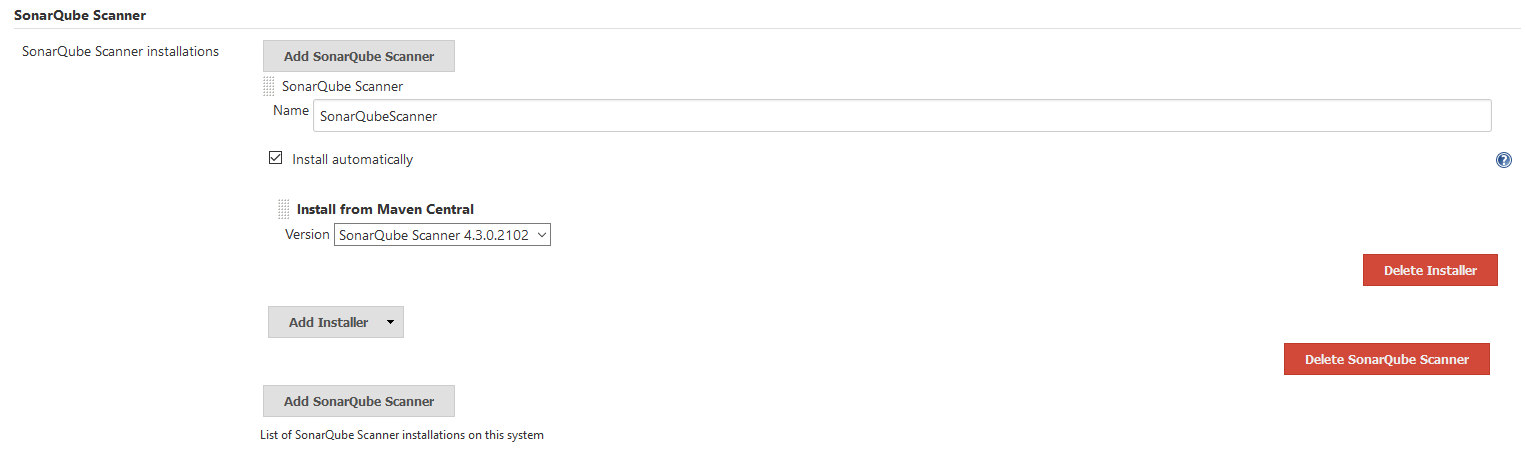
**Jenkins configuration**

There are configuration steps that need to be done in Jenkins before we can run the scans.

First, you need to install the [SonarQube Scanner](https://plugins.jenkins.io/sonar/?ref=igorski.co) plugin in Jenkins. Configure the plugin in the "**Configure System**" Jenkins section. You need to set the URL of the SonarQube server you are using and setup credentials. When you are setting up the credential, it has to be of the "**Secret Text**" kind, or it won't show up in the dropdown. The secret text is the token you need to generate in the SonarQube server. I am using the public [https://sonarcloud.io](https://sonarcloud.io/?ref=igorski.co) SonarQube service and this is how the configuration sections looks for me:



You also need to configure the SonnarScanner tool in the "**Global Tool Configuration"** in Jenkins. I picked the auto install option from Maven Central:



The only other thing left to configure is the stage in the Jenkinsfile. Although it looks straightforward it took me quite some trial and error to get everything right. Especially because online I was finding many articles explaining this but for a scripted pipeline, and that would not work in my declarative pipeline.

stage('SonarCloud') {

environment {

SCANNER\_HOME = tool 'SonarQubeScanner'

ORGANIZATION = "igorstojanovski-github"

PROJECT\_NAME = "igorstojanovski\_jenkins-pipeline-as-code"

}

steps {

withSonarQubeEnv('SonarCloudOne') {

sh '''$SCANNER\_HOME/bin/sonar-scanner -Dsonar.organization=$ORGANIZATION \

-Dsonar.java.binaries=build/classes/java/ \

-Dsonar.projectKey=$PROJECT\_NAME \

-Dsonar.sources=.'''

}

}

}

Two things are worth noticing. You need to use the same SonarScanner tool name as the SonarScanner tool as you used in the previous step. In the same manner, you also need to use the correct name of the SonnarQube server. In my case, the names are SonarQubeScanner for the tool and SonarCloudeOne for the server.

In this example, I pass two mandatory parameters to the sonar-scanner tool, organization and projectKey. The complete list of values that you can pass to the scanner tool directly you can find in the SonarQube [documentation](https://docs.sonarqube.org/latest/analysis/languages/java/?ref=igorski.co).

The withSonarQubeEnv() method can take additional parameters. You can use a specific SonarQube server or specific credentials:

withSonarQubeEnv(installationName: 'SonarCloudOne', credentialsId: 'SonarCloudOne')

Once you have all this in place, the build should be successful and the scan should show up in the SonarQube server.

You can also add a quality gate stage that will fail the build if results were under a defined threshold. This would need to be placed after the last stage that utilizes the sonar-scanner.

stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'MINUTES') {

waitForQualityGate abortPipeline: true

}

}

}

In order for this to work you need to [configure a webhook](https://redirect.sonarsource.com/doc/webhooks.html?ref=igorski.co) in your SonarQube server pointing to <your Jenkins instance>/jenkins/sonarqube-webhook/.

You can find the full Jenkinsfile in [my GitHub project](https://github.com/igorstojanovski/jenkins-pipeline-as-code/blob/sonar-test/Jenkinsfile?ref=igorski.co).

**Debugging time**

Sometimes it is useful to know all the things that can go wrong. In my case, a lot of things went wrong before I managed to make it work. These are some of the mistakes I made and the matching error outputs produced.

I used the wrong name of the installation in the Jenkinsfile from what I had in the Jenkins configuration.

ERROR: SonarQube installation defined in this job (SonarCloudOne) does not match any configured installation. Number of installations that can be configured: 0.

If you want to reassign jobs to a different SonarQube installation, check the documentation under https://redirect.sonarsource.com/plugins/jenkins.htmls

I didn't set the Sonar project and organization names.

ERROR: Error during SonarScanner execution

ERROR: You must define the following mandatory properties for 'Unknown': sonar.projectKey, sonar.organization

I used withSonarQubeEnv() without parameters. I thought that Jenkins will be smart enough to pick up the only one I had configured. It wasn't.

WorkflowScript: 33: Missing required parameter: "installationName" @ line 33, column 9.

withSonarQubeEnv() {

^

1 error at org.codehaus.groovy.control.ErrorCollector.failIfErrors(ErrorCollector.java:310)atorg.codehaus.groovy.control.CompilationUnit.applyToPrimaryClassNodes(CompilationUnit.java:1085) at org.codehaus.groovy.control.CompilationUnit.doPhaseOperation(CompilationUnit.java:603) at org.codehaus.groovy.control.CompilationUnit.processPhaseOperations(CompilationUnit.java:581)

...

Finished: FAILURE

I also tried to define the SonarScanner tool path as a variable:

def scannerHome = tool 'SonarScanner 4.0';

This will work in a scripted pipeline but not in a declarative. This needs to be set in an env block instead as in the example code above.