Quality Gates — A must have thing for the code analysis process

Tarun Prakash Follow
Mar 26, 2019 · 5 min read

Overview

Before we introduce Quality Gates, let's understand the *code analysis* process code analysis is an important and necessary item in the agile product development cycle. It should be used to detect any possible failures and defects arising out of the continuous changes in the source codes.

There are a few good reasons why this should be included in the development lifecycle.

- It helps in detecting areas in the code that needs refactoring or simplification.
- It can help to find the bug early in the development cycle, which means less cost to fix them.
- Overall improvement in the quality of the code.
- We can define project specific rules which will then be implemented without manual intervention.

More importantly, you can include this within the build process once and use it always without doing anything manually.

Problem Statement

Now let's talk about the actual problem. sonarqube which is a code analysis tool, helps us gain visibility into our code base. Soon at MIQ, we realized that having visibility into code was not enough and in order to address the issues flagged by code analysis, we had to make use of different data insights that we were getting from sonarqube.

At this point, we realized that we have to uphold certain guidelines and regulate them across all teams within the organization.

Quality Gates was exactly what we needed and this turned out to be the best way to ensure that standards are met and regulated across all the projects in our organization. **Quality Gates** can be defined as a set of threshold measures set on your project. Few conditions that can be in included are listed below.

- Code Coverage > certain value
- Number of Blocker issues >certain value
- Security Rating / Unit Test Pass Rate etc..

How — Enforce Quality Gates?

In this article, we will show you how to set up Quality Gates in sonarqube and then configure Jenkins job to enforce Quality Gates.

To manage Quality Gates, go to Quality Gates (top menu bar) in sonarqube.

Each Quality Gate condition is a combination of:

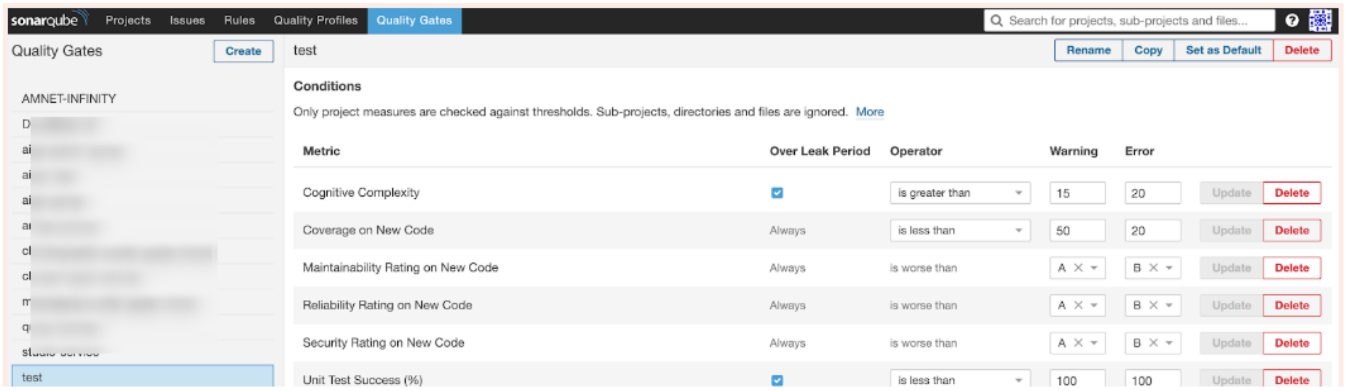
- measure
- period: Value (to date) or New Code (differential value over the New Code period)
- comparison operator
- warning value (optional)
- error value (optional)

For instance, a condition might be:

- measure: Blocker issue
- period: Value
- comparison operator: >
- error value: 0

Which can be stated as No blocker issues.

In the example given below, we have configured a quality gate only for the metric called "***code coverage***" for demo purpose. However, based on your project requirement, you can set up the metrics in your quality gate. For example, the following snapshot shows a few standard metrics that can be configured.



An example — Quality Gates metrics

Jenkins Job setup

At MIQ, we use Jenkins as our CI and sonarqube as a code inspection tool. Therefore, we wanted to force Jenkins's build job to fail if the code doesn't meet the specified Quality Gates.

1. Sample quality gate metrics setup in sonarqube.
2. Configure Jenkins job to fail the build when not meeting Quality Gates.

> ***Prerequisites*** *— Install Jenkins plugin "**sonar-quality-gates-plugin**" if not already present. Email-ext plugin for Jenkins to be able to send emails. We also need Jenkins job which is already configured with sonar analysis and build for it passing successfully.*

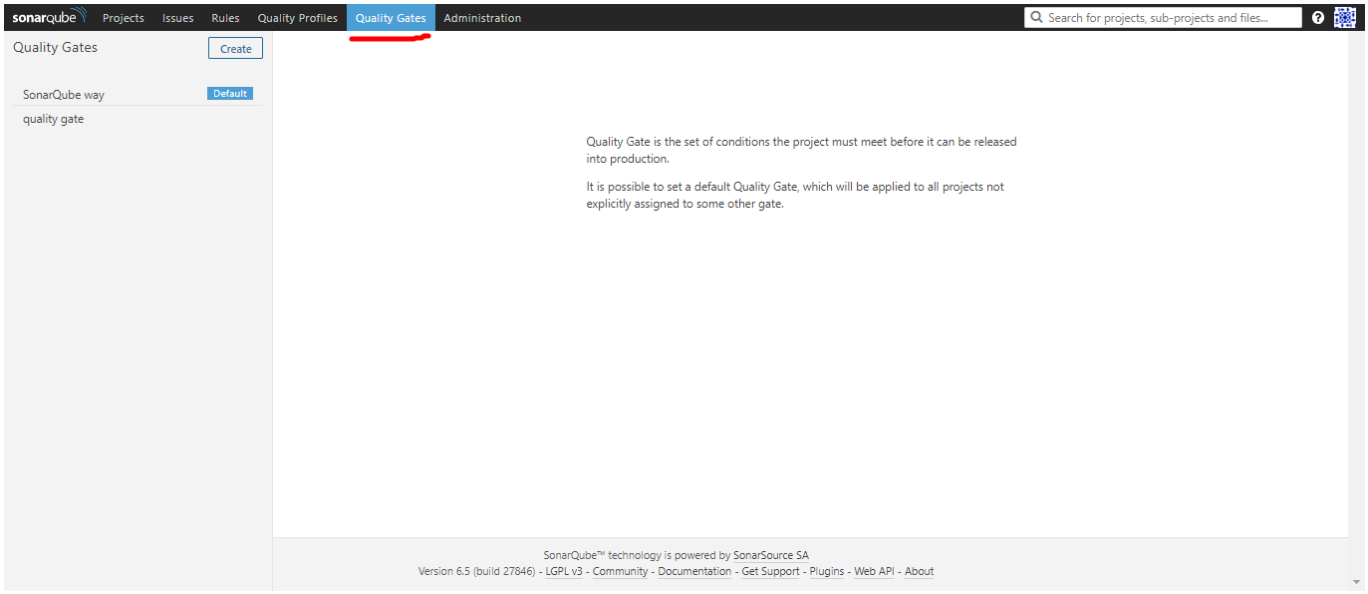
For example, here is the snapshot of the Jenkins build job that is currently passing Jenkins build job before Quality Gates setup.

```
[DEBUG] 07:05:25.194 Execution stop
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] miqp ..................................... SUCCESS [ 12.475 s]
[INFO] Miqp Ui ..................................... SUCCESS [ 32.142 s]
```

```
[INFO] miqp-server ..................................... SUCCESS [  3.910 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49.951 s
[INFO] Finished at: 2018-05-30T07:05:25+00:00
[INFO] Final Memory: 62M/612M
[INFO] ------------------------------------------------------------------------
```

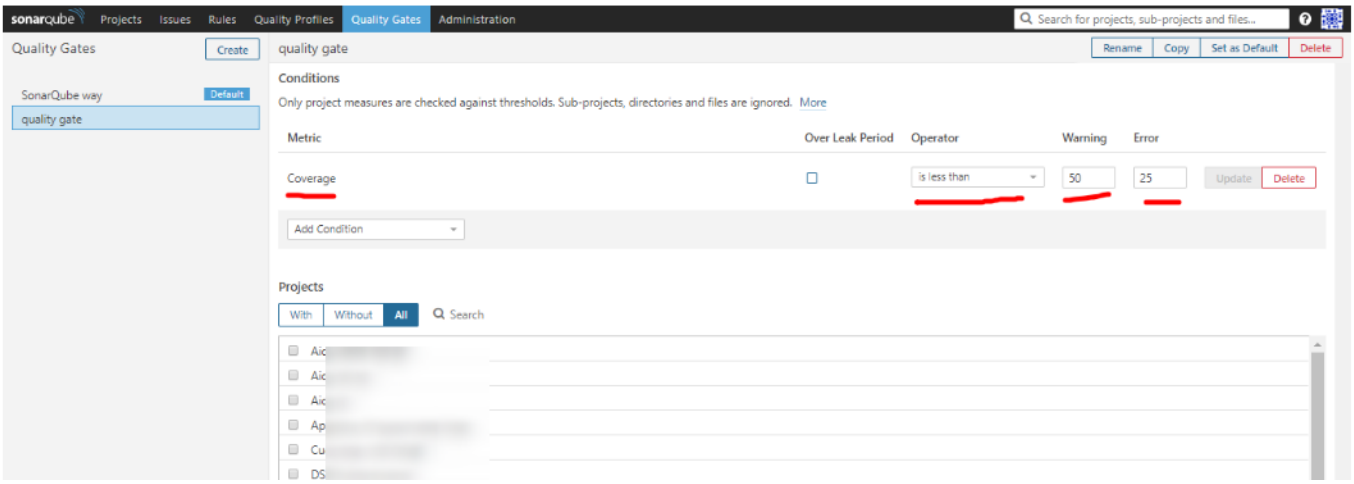
Let's setup Quality Gates metrics in Sonar. We are going to create a quality gate only for the metric called ***Code Coverage*** for demo purpose. But there are more metrics available that you should be selecting while creating quality gates.

Login to sonar > got Quality Gates as shown in the screen below.



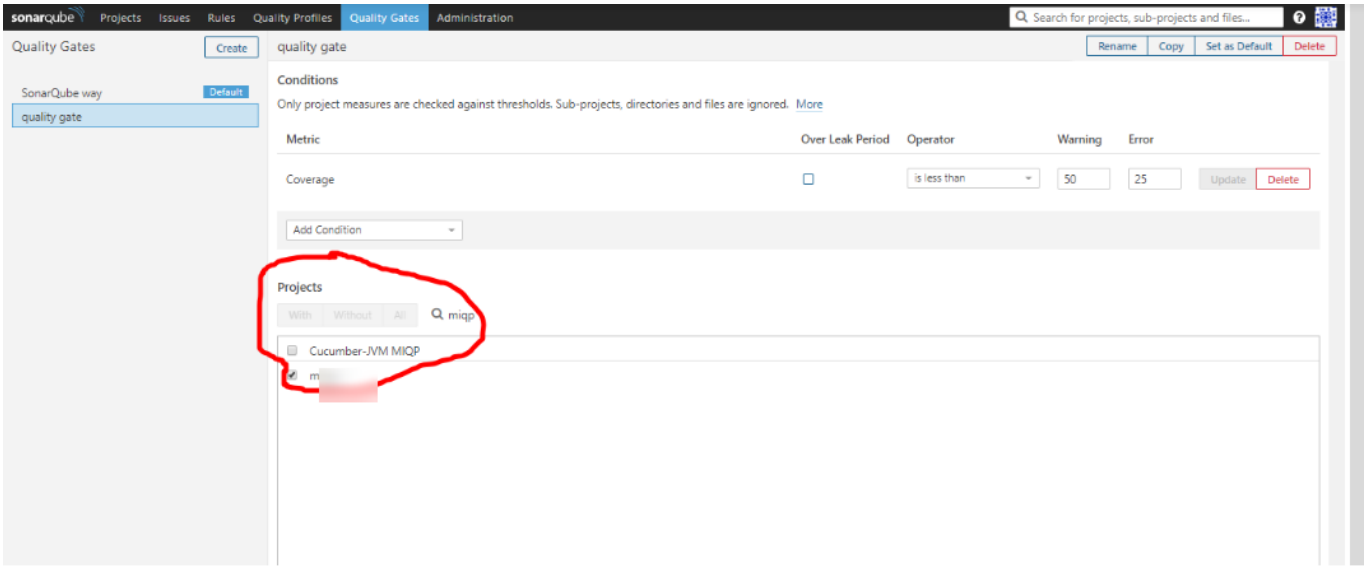
Quality Gate — screen shot in sonarqube

Click on create > Add Condition > Choose metrics (In this example, we selected **Code Coverage**) > select operator along with warning and error threshold.



Create: Quality Gate

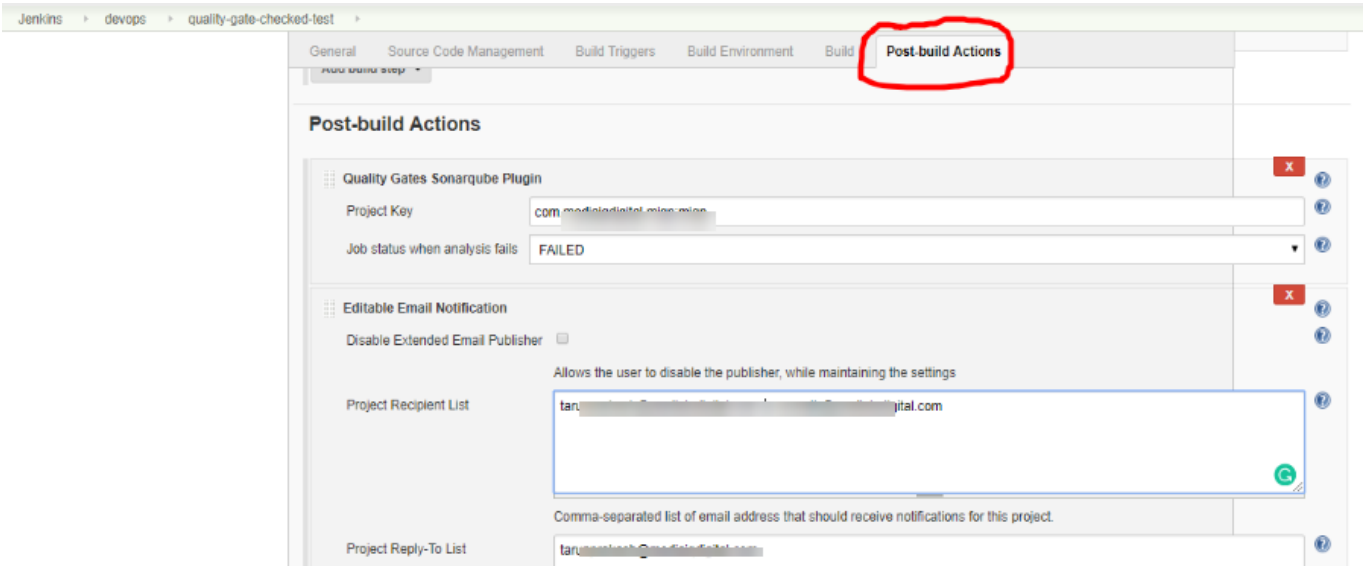
Select the project to add Quality Gates. We have selected a sample miqp project. In your case, project name would be different so please change it accordingly.

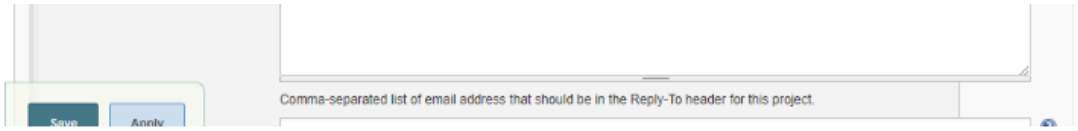


Add project to Quality Gate

Now go to the Jenkins job and configure the Quality Gate validation.

Click on the job and go to **Post-build Actions** and provide the project details you have associated with Quality Gate created in the earlier steps.





Post-build action — Enforce Quality Gates

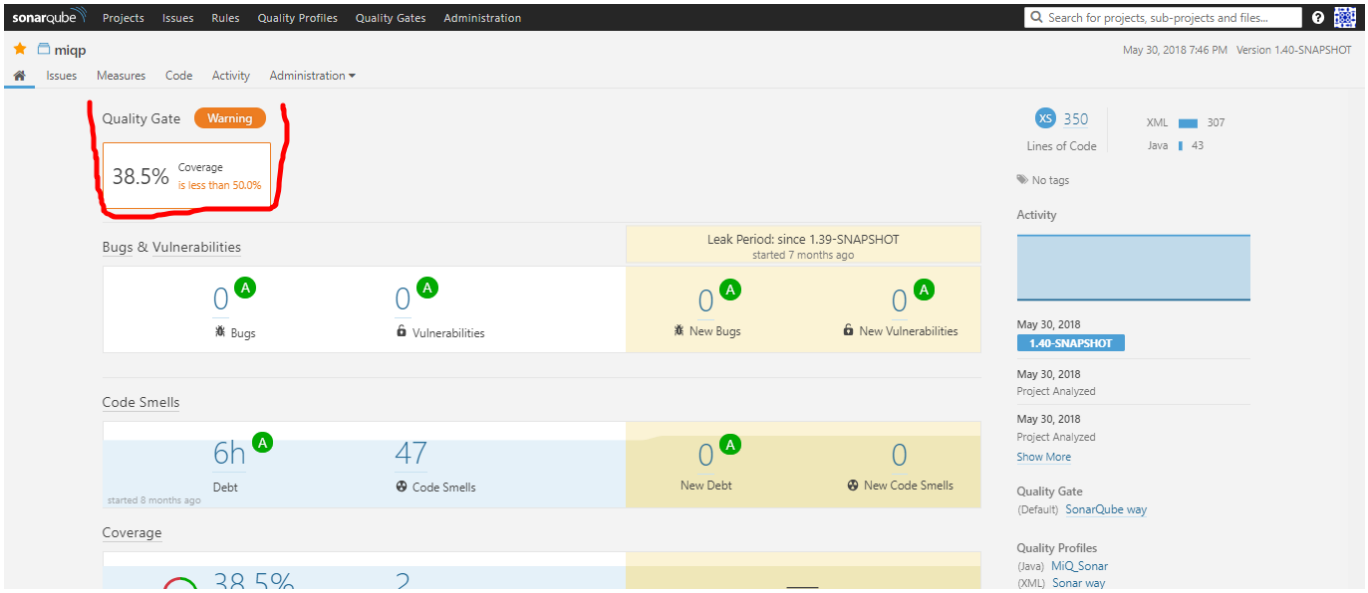
We have configured the project key of miqp sample project along with Job status when sonar analysis fails.

Verify if our build fails after the quality check was enabled.

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] miqp ..................................... SUCCESS [ 12.475 s]
[INFO] Miqp Ui .................................. SUCCESS [ 32.142 s]
[INFO] miqp-server .............................. SUCCESS [  3.910 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49.951 s
[INFO] Finished at: 2018-05-30T07:05:25+00:00
[INFO] Final Memory: 62M/612M
[INFO] ------------------------------------------------------------------------
Has build PENDING with id: AWOv3ZX95eWk5Hp8Fka1 - waiting 10000 to execute next check.
Status => SUCCESS
PostBuild-Step: Quality Gates plugin build passed: FALSE
Build step 'Quality Gates Sonarqube Plugin' marked build as failure
Email was triggered for: Always
Sending email for trigger: Always
Sending email to: ta
Finished: FAILURE
```

Post-build notification alert

You can verify the same in the sonarqube server.



Project status on Quality Gate

In the above screenshot, you can verify that the project miqp is reporting **WARNING** as the code failed to meet the code coverage metric (i.e we set in one of the earlier steps).

Conclusion

In this article, we have shown how you can take your code analysis benefits to the next level using Quality Gates. We also walked you through the process of setting up Quality Gates at build level, using sonarqube and Jenkins.

This definitely helped the teams at MiQ by being on top of our quality control check and that too in automated fashion. We are sure that this process can help you in your development process as well.

Jenkins Sonarqube Quality Assurance How To Code Analysis