

(/)

Code Analysis with SonarQube

Last modified: December 31, 2019

by Michal Aibin (<https://www.baeldung.com/author/michal-author/>)

DevOps (<https://www.baeldung.com/category/devops/>)

Static Analysis (<https://www.baeldung.com/tag/static-analysis/>)

I just announced the new *Learn Spring* course, focused on the fundamentals of Spring 5 and Spring Boot 2:

>> CHECK OUT THE COURSE (</ls-course-start>)

1. Overview

In this article, we're going to be looking at static source code analysis with SonarQube (<https://www.sonarqube.org/>) – which is an open-source platform for ensuring code quality.

Let's start with a core question – why analyze source code in the first place? Very simply put, to ensure quality, reliability, and maintainability over the life-span of the project; a poorly written codebase is always more expensive to maintain.

Alright, now let's get started by downloading the latest LTS version of SonarQube from the download page (<https://www.sonarqube.org/downloads/>) and setting up our local server as outlined in this quick start guide (<https://docs.sonarqube.org/display/SONAR/Get+Started+in+Two+Minutes>).

2. Analyzing Source Code

Now that we're logged in, we're required to create a token by specifying a name – which can be our username or any other name of choice and click on the generate button.

We'll use the token later at the point of analyzing our project(s). We also need to select the primary language (Java) and the build technology of the project (Maven).

Let's define the plugin in the *pom.xml*:

```
1 <build>
2   <pluginManagement>
3     <plugins>
4       <plugin>
5         <groupId>org.sonarsource.scanner.maven</groupId>
6         <artifactId>sonar-maven-plugin</artifactId>
7         <version>3.4.0.905</version>
8       </plugin>
9     </plugins>
10  </pluginManagement>
11 </build>
```

The latest version of the plugin is available here

(<https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.sonarsource.scanner.maven%22%20AND%20a%3A%22sonar-maven-plugin%22>). Now, we need to execute this command from the root of our project directory to scan it:

```
1 mvn sonar:sonar -Dsonar.host.url=http://localhost:9000
2   -Dsonar.login=the-generated-token
```

We need to replace *the-generated-token* with the token from above.

The project that we used in this article is available here

(<https://github.com/eugenp/tutorials/tree/master/cas/cas-secured-app>).

We specified the host URL of the SonarQube server and the login (generated token) as parameters for the Maven plugin.

After executing the command, the results will be available on the Projects dashboard – at *<http://localhost:9000>*.

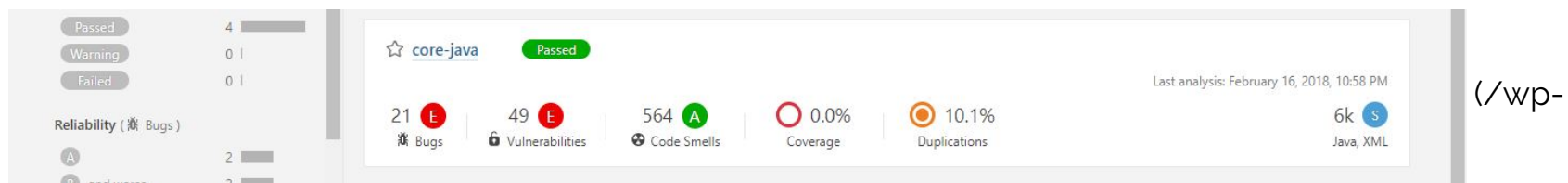
There are other parameters that we can pass to the Maven plugin or even set from the web interface; *sonar.host.url*, *sonar.projectKey*, and *sonar.sources* are mandatory while others are optional.

Other analysis-parameters and their default values are here (<https://docs.sonarqube.org/display/SONAR/Analysis+Parameters>). Also, note that each language-plugin has rules for analyzing compatible source code.

3. Analysis Result

Now that we've analyzed our first project, we can go to the web interface at <http://localhost:9000> and refresh the page.

There we'll see the report summary:



content/uploads/2018/02/9029390_overview.jpg)

Discovered issues can either be a Bug, Vulnerability, Code Smell, Coverage or Duplication. Each category has a corresponding number of issues or a percentage value.

Moreover, issues can have one of five different severity levels: *blocker*, *critical*, *major*, *minor* and *info*. Just in front of the project name is an icon that displays the Quality Gate status – passed (green) or failed (red).

Clicking on the project name will take us to a dedicated dashboard where we can explore issues particular to the project in greater detail.

We can see the project code, activity and perform administration tasks from the project dashboard – each available on a separate tab.

Though there is a global *Issues* tab, the *Issues* tab on the project dashboard display issues specific to the project concerned alone:

(/wp-

content/uploads/2018/02/383732889_issues.jpg)

The issues tab always display the category, severity level, tag(s), and the calculated effort (regarding time) it will take to rectify an issue.

From the issues tab, it's possible to assign an issue to another user, comment on it, and change its severity level. Clicking on the issue itself will show more detail about the issue.

The issue tab comes with sophisticated filters to the left. These are good for pinpointing issues. So how can one know if the codebase is healthy enough for deployment into production? That's what Quality Gate is for.

4. SonarQube Quality Gate

In this section, we're going to look at a key feature of SonarQube – Quality Gate. Then we'll see an example of how to set up a custom one.

4.1. What Is a Quality Gate?

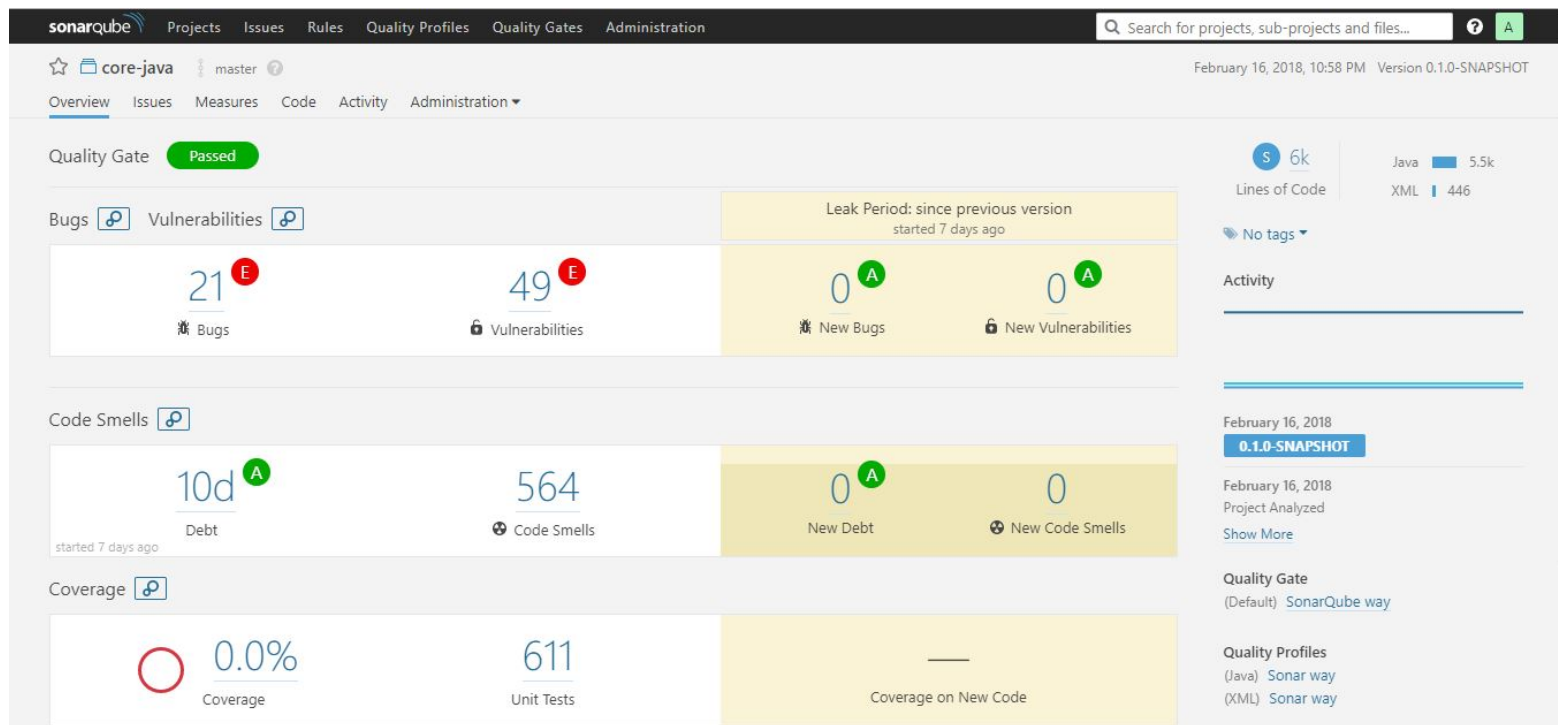
A Quality Gate is a set of conditions the project must meet before it can qualify for production release. **It answers one question: can I push my code to production in its current state or not?**

Ensuring code quality of “new” code while fixing existing ones is one good way to maintain a good codebase over time. The Quality Gate facilitates setting up rules for validating every new code added to the codebase on subsequent analysis.

The conditions set in the Quality Gate still affect unmodified code segments. If we can prevent new issues arising, over time, we'll eliminate all issues.

This approach is comparable to fixing the water leakage (<https://docs.sonarqube.org/display/SONAR/Fixing+the+Water+Leak>) from the source. **This brings us to a particular term – Leakage Period. This is the period between two analyses/versions of the project.**

If we rerun the analysis, on the same project, the overview tab of the project dashboard will show results for the leak period:



(/wp-

content/uploads/2018/02/36326789289_leak_period.jpg)

From the web interface, the Quality Gates tab is where we can access all the defined quality gates. By default, *SonarQube way* came preinstalled with the server.

The default configuration for *SonarQube way* flags the code as failed if:

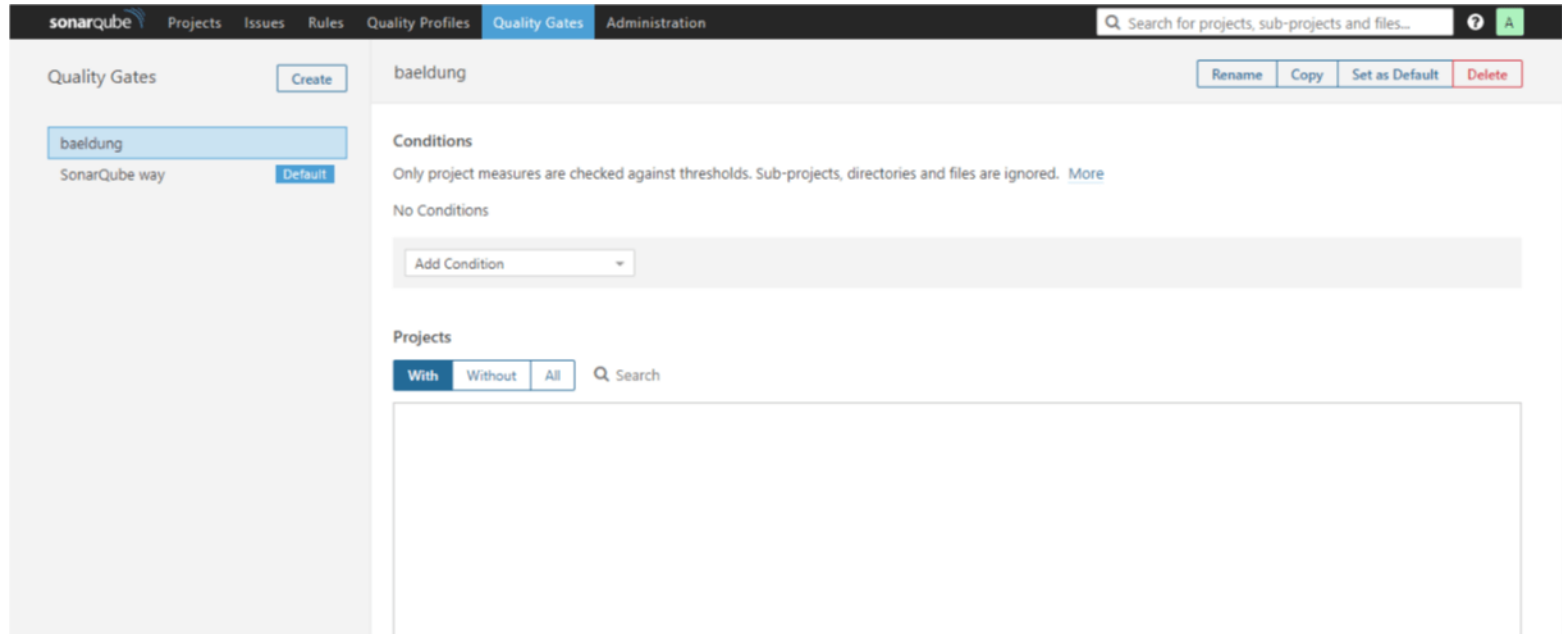
- the coverage on new code is less than 80%
- percentage of duplicated lines on new code is greater than 3
- maintainability, reliability or security rating is worse than A

With this understanding, we can create a custom Quality Gate.

4.2. Adding Custom Quality Gate

First, we need to **click on the *Quality Gates* tab and then click on the *Create* button** which is on the left of the page. We'll need to give it a name – *baeldung*.

Now we can set the conditions we want:

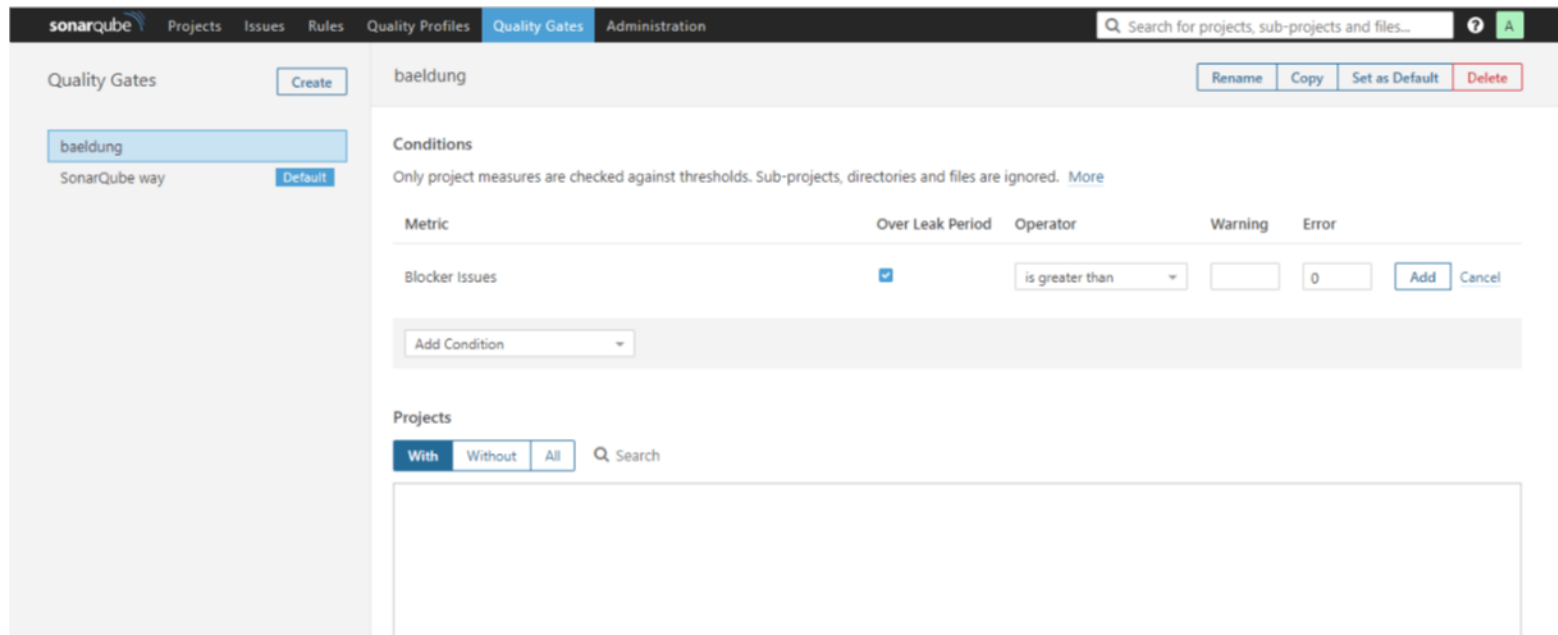


(/wp-

content/uploads/2018/02/create-custom-gate-1.png)

From the *Add Condition* drop-down, let's choose *Blocker Issues*; it'll immediately show up on the list of conditions.

We'll specify *is greater than* as the *Operator*, set zero (0) for the *Error* column and check *Over Leak Period* column:



(/wp-

content/uploads/2018/02/create-custom-gate-2.png)

Then we'll click on the *Add* button to effect the changes. Let's add another condition following the same procedure as above.

We'll select *issues* from the *Add Condition* drop-down and check *Over Leak Period* column.

The value of the *Operator* column will be set to "*is less than*" and we'll add one (1) as the value for the *Error* column. This means *if the number of issues in the new code added is less than 1, mark the Quality Gate as failed*.

I know this doesn't make technical sense but let's use it for learning sake. Don't forget to click the *Add* button to save the rule.

One final step, we need to attach a project to our custom Quality Gate. We can do so by scrolling down the page to the *Projects* section.

There we need to click on *All* and then mark our project of choice. We can as well set it as the default Quality Gate from the top-right corner of the page.

We'll scan the project source code, again, as we did before with Maven command. When that's done, we'll go to the projects tab and refresh.

This time, the project will not meet the Quality Gate criteria and will fail. Why? Because in one of our rules we have specified that, it should fail if there are no new issues.

Let's go back to the Quality Gates tab and change the condition for *issues* to *is greater than*. We need to click the update button to effect this change.

A new scan of the source code will pass this time around.

5. Integrating SonarQube into a CI

Making SonarQube part of a Continuous Integration process is possible. This will automatically fail the build if the code analysis did not satisfy the Quality Gate condition.

For us to achieve this, we're going to be using SonarCloud (<https://about.sonarcloud.io/>) which is the cloud-hosted version of SonaQube server. We can create an account here (<https://sonarcloud.io/sessions/new>).

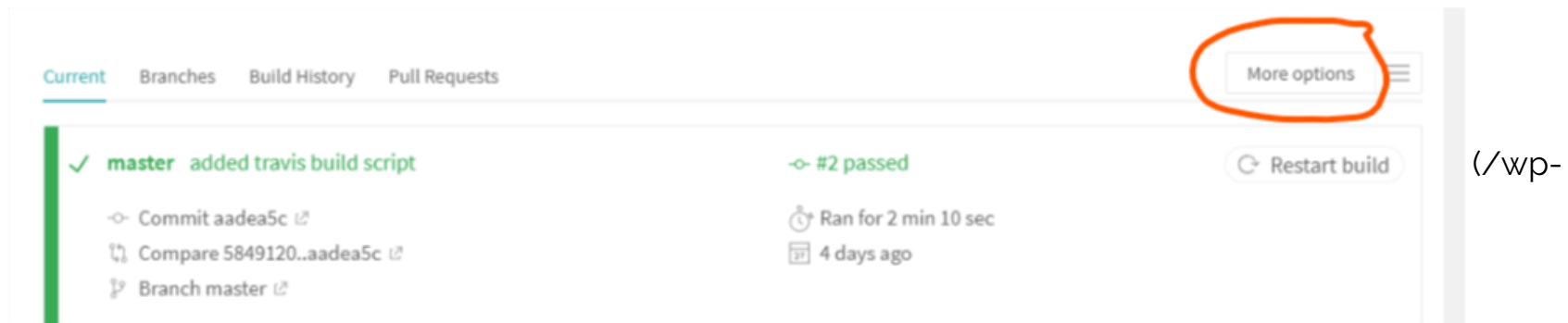
From My Account > Organizations, we can see the organization key, and it will usually be in the form *xxxx-github* or *xxxx-bitbucket*.

Also from *My Account* > *Security*, we can generate a token as we did in the local instance of the server. Take note of both the token and the organization key for later use.

In this article, we'll be using Travis CI, and we'll create an account here (<https://travis-ci.org/>) with an existing Github profile. It will load all our projects, and we can flip the switch on any to activate Travis CI on it.

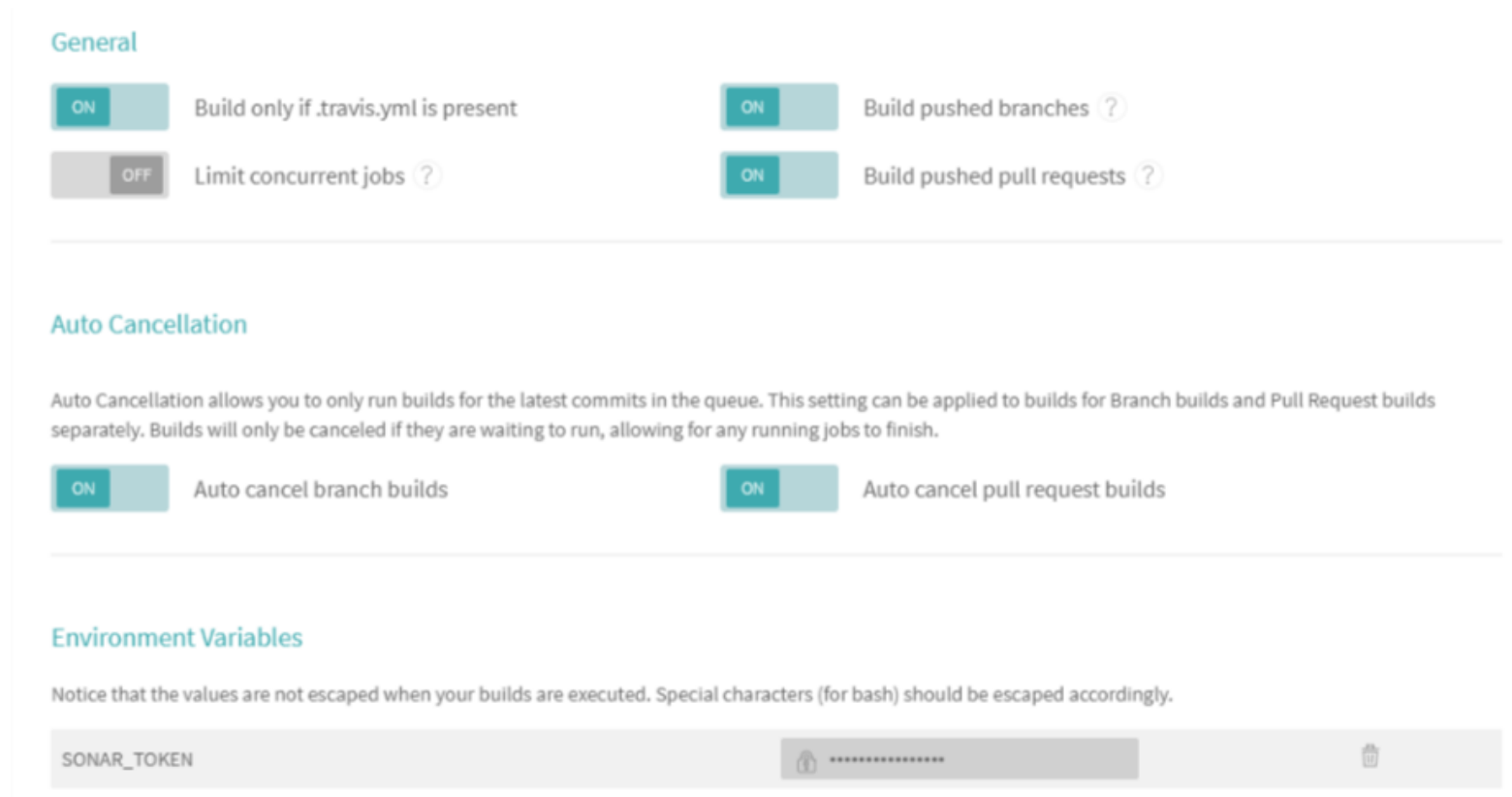
We need to add the token we generated on SonarCloud to Travis environment variables. We can do this by clicking on the project we've activated for CI.

Then, we'll click "More Options" > "Settings" and then scroll down to "Environment Variables":



content/uploads/2018/02/travis-ci-1.png)

We'll add a new entry with the name `SONAR_TOKEN` and use the token generated, on SonarCloud, as the value. Travis CI will encrypt and hide it from public view:



The image shows a screenshot of the Travis CI settings interface. It is divided into three main sections: General, Auto Cancellation, and Environment Variables.

General

- Build only if .travis.yml is present:** A toggle switch that is currently turned ON.
- Limit concurrent jobs:** A toggle switch that is currently turned OFF.
- Build pushed branches:** A toggle switch that is currently turned ON.
- Build pushed pull requests:** A toggle switch that is currently turned ON.

Auto Cancellation

Auto Cancellation allows you to only run builds for the latest commits in the queue. This setting can be applied to builds for Branch builds and Pull Request builds separately. Builds will only be canceled if they are waiting to run, allowing for any running jobs to finish.

- Auto cancel branch builds:** A toggle switch that is currently turned ON.
- Auto cancel pull request builds:** A toggle switch that is currently turned ON.

Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

Below this text is a table of environment variables. The first row shows a variable named `SONAR_TOKEN` with a value represented by a series of asterisks, indicating it is a secret. There are icons for adding, editing, and deleting variables.

(/wp-

content/uploads/2018/02/travis-ci-2.png)

Finally, we need to add a `.travis.yml` file to the root of our project with the following content:

```
1 language: java
2 sudo: false
3 install: true
4 addons:
5   sonarcloud:
6     organization: "your_organization_key"
7     token:
8       secure: "$SONAR_TOKEN"
9 jdk:
10   - oraclejdk8
11 script:
12   - mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent package sonar:sonar
13 cache:
14   directories:
15     - '$HOME/.m2/repository'
16     - '$HOME/.sonar/cache'
```

Remember to substitute your organization key with the organization key described above. Committing the new code and pushing to Github repo will trigger Travis CI build and in turn activate the sonar scanning as well.

6. Conclusion

In this tutorial, we've looked at how to set up a SonarQube server locally and how to use Quality Gate to define the criteria for the fitness of a project for production release.

The SonarQube documentation (<https://docs.sonarqube.org/display/SONAR>) has more information about other aspects of the platform.

I just announced the new *Learn Spring* course, focused on the fundamentals of Spring 5 and Spring Boot 2:

>> CHECK OUT THE COURSE ([/ls-course-end](#))



Learning to build your API
with Spring?

Enter your email address

>> Get the eBook

Comments are closed on this article!

CATEGORIES

SPRING ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/](https://www.baeldung.com/category/spring/))

REST ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/](https://www.baeldung.com/category/rest/))

JAVA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/](https://www.baeldung.com/category/java/))

SECURITY ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](https://www.baeldung.com/category/security-2/))

PERSISTENCE ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](https://www.baeldung.com/category/persistence/))

JACKSON ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/](https://www.baeldung.com/category/json/jackson/))

HTTP CLIENT-SIDE ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/](https://www.baeldung.com/category/http/))

KOTLIN ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](https://www.baeldung.com/category/kotlin/))

SERIES

JAVA "BACK TO BASICS" TUTORIAL ([/JAVA-TUTORIAL/](/java-tutorial/))

[JACKSON JSON TUTORIAL \(/JACKSON\)](#)
[HTTPCLIENT 4 TUTORIAL \(/HTTPCLIENT-GUIDE\)](#)
[REST WITH SPRING TUTORIAL \(/REST-WITH-SPRING-SERIES\)](#)
[SPRING PERSISTENCE TUTORIAL \(/PERSISTENCE-WITH-SPRING-SERIES\)](#)
[SECURITY WITH SPRING \(/SECURITY-SPRING\)](#)

ABOUT

[ABOUT BAELDUNG \(/ABOUT\)](#)
[THE COURSES \(HTTPS://COURSES.BAELDUNG.COM\)](#)
[JOBS \(/TAG/ACTIVE-JOB/\)](#)
[THE FULL ARCHIVE \(/FULL_ARCHIVE\)](#)
[WRITE FOR BAELDUNG \(/CONTRIBUTION-GUIDELINES\)](#)
[EDITORS \(/EDITORS\)](#)
[OUR PARTNERS \(/PARTNERS\)](#)
[ADVERTISE ON BAELDUNG \(/ADVERTISE\)](#)

[TERMS OF SERVICE \(/TERMS-OF-SERVICE\)](#)
[PRIVACY POLICY \(/PRIVACY-POLICY\)](#)
[COMPANY INFO \(/BAELDUNG-COMPANY-INFO\)](#)
[CONTACT \(/CONTACT\)](#)