SonarQube is an automatic code review tool to detect bugs, vulnerabilities and code smells in your code. It can integrate with your existing workflow to enable continuous code inspection across your project branches and pull requests.

After installing sonarQube we need a scanner for analysis for exam Scanner for Maven

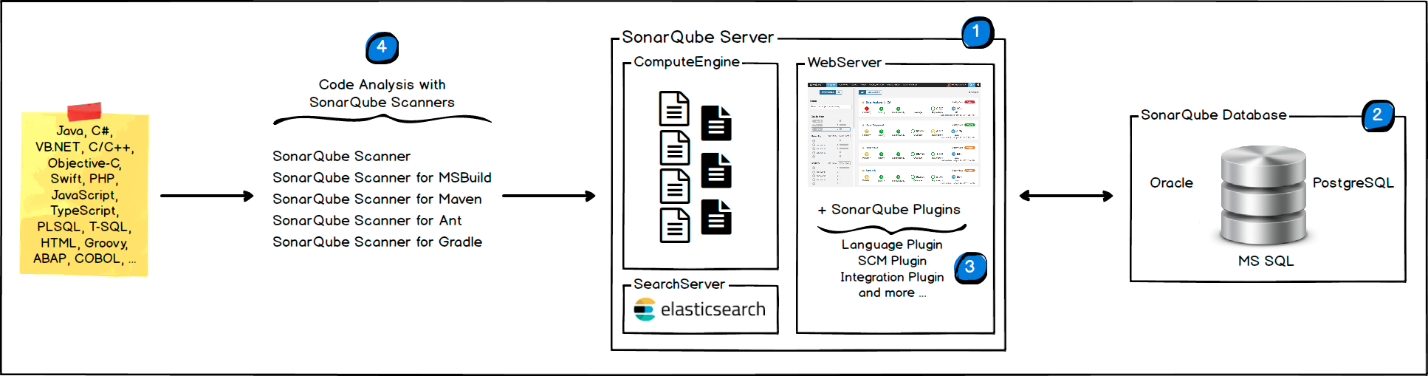
outcome of this analysis will be quality measures and issues (instances where coding rules were broken).

By default, only files that are recognized by a language analyzer are loaded into the project during analysis. For example if you're using SonarQube Community Edition, which includes analysis of Java and JavaScript, but not C++, all .java and .js files would be loaded, but .cpp files would be ignored.

**Analysis**

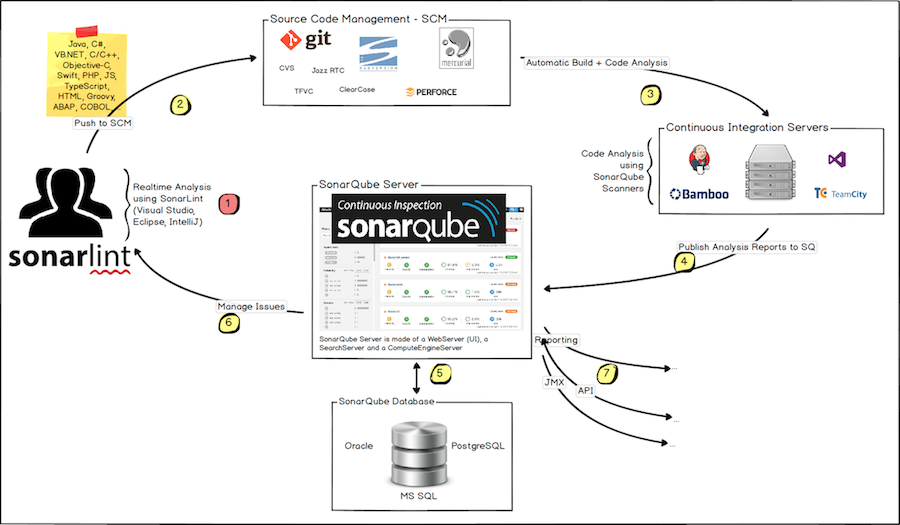
During analysis, data is requested from the server, the files provided to the analysis are analyzed, and the resulting data is sent back to the server at the end in the form of a report, which is then analyzed on server-side.

Analysis reports are queued, and processed sequentially, so it is quite possible that for a brief period after your analysis log shows completion, the updated values are not visible in your SonarQube project.



1. One SonarQube Server starting 3 main processes:
   * Web Server for developers, managers to browse quality snapshots and configure the SonarQube instance
   * Search Server based on Elasticsearch to back searches from the UI
   * Compute Engine Server in charge of processing code analysis reports and saving them in the SonarQube Database
2. One SonarQube Database to store:
   * the configuration of the SonarQube instance (security, plugins settings, etc.)
   * the quality snapshots of projects, views, etc.
3. Multiple SonarQube Plugins installed on the server, possibly including language, SCM, integration, authentication, and governance plugins
4. One or more SonarScanners running on your Build / Continuous Integration Servers to analyze projects

How SonarQube works:



* The SonarQube Platform cannot have more than one SonarQube Server and one SonarQube Database.
* For optimal performance, each component (server, database, scanners) should be installed on a separate machine, and the server machine(s) should be dedicated.
* There is **no communication** between **SonarScanners** and the **SonarQube Database**.
* In the Maven build, it already has much of the information needed for SonarQube to successfully analyze a project. By preconfiguring the analysis based on that information, the need for manual configuration is reduced significantly.

**Analysis Parameters:**

* Parameters set through the UI are stored in the database. If we write sonar.exclusions then the parameter is not stored in database.
* Property keys shown in the interface at both global and project levels can also be set as analysis parameters. Property keys for eg: the server url, some project configuration characters.
* There are different type of keys:

1. Mandatory keys:
2. Server url as key
3. Project configuration as key(characters allowed for projects unique key)
4. Optional Parameters:
5. Project Identity as key(name of project to be displayed, project version)
6. Authentication as key :Login and password credentials
7. Web services – waiting for response from web services
8. Project configuration-project description,homepage,CI,issue
9. Duplications key
10. Analysis logging key
11. Deprecated key- This should not be used in any anlysis.

* SonarSource analyzers do not run your tests or generate reports. They only import pre-generated reports.

**Background Tasks**

A Background Task can be:

* the import of an Analysis Report
* the computation of a Portfolio
* the import or export of a project

Analysis is not complete until the relevant Background Task has been completed. Even though the SonarScanner's log shows EXECUTION SUCCESS, the analysis results will not be visible in the SonarQube project until the Background Task has been completed. After a SonarScanner has finished analyzing your code, the result of the analysis - the Analysis Report - is sent to SonarQube Server for final processing by the Compute Engine. Analysis Reports are queued and processed serially.

**Generic Issue data**

SonarQube supports a generic import format for raising "external" issues in code. It is intended to allow you to import the issues from your favorite linter even if no plugin exists for it.

**Pull Requests**

Pull Request analysis allows you to see your Pull Request's Quality Gate and analysis in the SonarQube .

A [Quality Gate](https://docs.sonarqube.org/latest/user-guide/quality-gates/) lets you ensure you are meeting your organization's quality policy and that you can merge your Pull Request. The Pull Request Quality Gate:

* **Focuses on new code** – The PR quality gate only uses your project's quality gate conditions that apply to "on New Code" metrics.
* **Assigns a status** – Each PR shows a quality gate status reflecting whether it Passed or Failed.

There are analysis parameters for pull requests as well.

**Branch analysis**

With Branch Analysis, you can ensure that you're maintaining consistent code quality all the way down.

The branch Quality Gate lets you know if your branch is ready to be merged. Each branch has a quality gate that:

* Applies on conditions on New Code and overall code.
* Assigns a status (Passed or Failed).