ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

SCIENTIFIC Research and Community

Review Article

Open Access

Integration of SonarQube, The Quality Inspector for GO & Docker Compose

Pallavi Priya Patharlagadda

United States of America

ABSTRACT

Project success at any company is mostly dependent on code quality and security. Continuous evaluation of code quality is necessary to attain peak efficiency and minimize the likelihood of mistakes. Nevertheless, without the use of a static code analysis tool, achieving complete code visibility might be difficult. The SonarQube platform will be of interest to your company if your software development team is looking to enhance code quality. Nonetheless, developers must make sure that coding standards are followed at all times if their teams employ a CI/CD pipeline to update the code base. This article will define SonarQube, highlight some of its best features, and discuss why companies should use it for code analysis, and integrate it with a GO project and Docker Compose.

***Corresponding author**

Pallavi Priya Patharlagadda, United States of America.

Received: August 05, 2023; **Accepted:** August 12, 2023; **Published:** August 19, 2023

Problem Statement

Let's begin with a fundamental question why is source code analysis necessary at all? In other words, badly written codebases are always more expensive to maintain during the project. The cost of fixing bugs at production is tenfold expensive than fixing the bugs at the Development stage. So, Detecting the bugs or problems at the early stages of development is very important to assure quality, dependability, and maintainability. SonarQube is the answer to these questions. In the following sections, we will do a deep dive into GO, and Docker Compose and integrate them with SonarQube to ensure the code is written with the highest quality.

Introduction

Go is a procedural programming language. Rob Pike, Ken Thompson, and Robert Griesemer created it at Google in 2007, but it wasn't released as an open-source programming language until 2009. Packages are used in program assembly to effectively handle dependencies. Like dynamic languages, this language allows environment adoption patterns as well. For instance, type inference (y:= 0 is a legitimate declaration of a float variable y). Because of its straightforward, effective, and low learning curve, it is a well-liked option for developing online applications, command-line tools, and scalable network services.

The ability to perform several tasks at once is known as concurrency, and Go is well-known for supporting this feature. Go allows for concurrency through the use of channels and goroutines, which let you design code that can execute several tasks concurrently. Go is therefore the best option for developing scalable, high-performance network services and for resolving challenging computing issues.

Garbage collection, which takes care of memory management automatically for you, is another key component of Go. As a result, there is no longer a requirement for manual memory management, which lowers the possibility of memory leaks and other issues. Go is effective and supports contemporary hardware architectures. Building highly performant apps and massively distributed systems is a common use case for Go.

Go Vet

Go comes pre-installed with the utility vet, which analyzes Go code statically. Run go vet source/directory/*.go to utilize the vet program. The vet will identify all infractions and carry out several tests, such as nil function comparison, shadowed variable detection, and many more (see go doc cmd/vet for a complete list). The vet is fantastic since it quickly completes a lot of fundamental examinations.

Go Linters

However, Go Vet is insufficient if we wish to find typographical errors, redundant code, or even security issues. Linters such as GolangCI-Lint and Go Meta Linter can do all these tests. These are traditional linters that analyze Go code statically and conventionally provide their findings. Editors like as GoLand, Vim, and VisualStudio Code can incorporate these linters. Thus, these linters may identify a wide range of infractions by thoroughly analyzing Go code. In your build process, linting might take a long time depending on the linter setup.

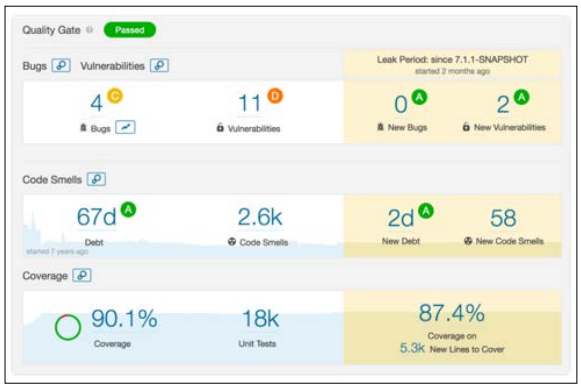
Calculating important metrics for your source code, such as lines of code or cyclomatic complexity, is another benefit of static code analysis. When combined with code coverage or the number of unit tests, these important metrics may give a clear picture of how well your source code is doing. They may be visible to everyone on an atlasboard-style dashboard in your project. However, where can we find these important metrics? Linters such as Go Meta Linter are capable of computing the most important metrics,

including cyclometric complexity. Tests yield other important data, such as code coverage and unit test count. In this manner, throughout the continuous integration process, we may compute the important metrics.

SonarQube

Are Linters insufficient? What makes SonarQube special? Yes. What if we were interested in learning how our source code's important metrics changed over time? Since the latest release, have we been able to increase code coverage? Every sprint, how many lines of code do we add? Which are the largest components in our projects?

This is where SonarQube is useful. SonarQube is an open-source tool that is used to continuously inspect code quality. Organizations and development teams use it to track, evaluate, and control the quality of their source code. In addition to supporting many programming languages, SonarQube offers insightful information on the state of software projects. Code coverage, unit tests, and static code analysis are the methods that SonarQube uses to examine source code over time. SonarQube can then respond to each of the aforementioned queries. You can see that version 2.1.0 has 3.842 lines of code, compared to version 1.0.0's 1.562 lines of code and 85% coverage. Upon closer inspection, the packet with the most growth is auth. You can examine in SonarQube how your source code and important metrics have changed over time and between different versions, as seen below.



SonarQube offers an additional advantage for projects or companies that build services in many programming languages when it comes to important source code metrics. For a wide range of programming languages, including Go, Java, C#, JavaScript, and many more, SonarQube can compute these crucial metrics. In this manner, you may compute aggregated critical metrics for projects or organizations that employ many programming languages. For instance, the total number of lines of code in your project is 12.346, and its overall code coverage is 76%. With 3.704 lines of code, the JavaScript frontend makes up almost one-third of the Java backend. Let's look at the key features of SonarQube.

Key Features of SonarQube

1. Code Quality Analysis: To find errors, security flaws, and code smells (badly written code), SonarQube does static code analysis. It verifies compliance with best practices and coding standards.
2. Metrics and Dashboards: SonarQube is a tool for gathering and displaying code-quality data, such as maintainability, test coverage, complexity, and code duplication. Using interactive dashboards, it displays the metrics.
3. Issue Tracking and Management: SonarQube identifies code problems and gives comprehensive details about each issue. With this data, developers can effectively prioritize and address issues.
4. Continuous Inspection: SonarQube facilitates the integration of pipelines for Continuous Integration/Continuous Deployment (CI/CD), enabling the automated execution of code quality tests upon every code contribution.
5. Language Support: Programming languages supported by SonarQube include Java, C/C++, C#, JavaScript, TypeScript, Python, Go, and more. It is therefore a flexible tool for code analysis across a range of applications.
6. Quality Gate: A "Quality Gate" is a collection of quality requirements that you may establish using SonarQube. Should the project fall short of these requirements, it may halt progress until the problems are fixed.
7. Custom Rules and Profiles: With SonarQube, you can customize quality profiles and coding rules to meet the unique needs and coding standards of your company.
8. Security Analysis: It is possible to find security flaws like SQL injection and cross-site scripting by using plugins such as SonarSource's Security plugins (such as SonarQube Security for Java and JavaScript).
9. Plugin Ecosystem: A robust community of plugins for SonarQube expands its capabilities. To add more languages, integrations, and custom rules, install more plugins.
10. Integration with Development Tools: SonarQube is compatible with build technologies like Maven, Gradle, and Jenkins as well as major development tools like Eclipse, IntelliJ IDEA, and Visual Studio.
11. Community and Commercial Editions: SonarQube is free and open-source software, with community versions accessible at no cost. Additionally, SonarSource, the business that created SonarQube, offers commercial editions with much more sophisticated capabilities and support services.

SonarQube for Go Projects

Establishing static code analysis and code quality checks in a Golang project requires many steps when using SonarQube. The main purpose of SonarQube is to analyze JVM-based languages, such as Java. However, with the use of a "SonarGo" plugin, you may use the SonarQube Scanner to analyze other languages, such as Golang. SonarGo is an external plugin that facilitates the analysis of Golang projects within SonarQube.

A step-by-step guide to using SonarQube with a Golang project

Step 1: Set up SonarQube Server

- Download and install the SonarQube server from the official website: https://www.sonarqube.org/downloads/
- Start the SonarQube server by running the appropriate script (e.g., sonar.sh on Linux/macOS or StartSonar.bat on Windows).
- Access the SonarQube web interface at http://localhost:9000 (by default). Log in with the default credentials (admin/admin), and change the password after the first login.

Step 2: Install and Configure SonarGo Plugin

- Download the SonarGo plugin (JAR file) from the SonarGo GitHub repository: https://github.com/360EntSecGroup-Skylar/goreporter
- Copy the downloaded JAR file into the extensions/plugins directory of your SonarQube installation.
- Restart the SonarQube server to load the SonarGo plugin.

Step 3: Install SonarScanner

- Download and install the SonarScanner for your platform from: https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/
- Add the SonarScanner executable to your system PATH.

Step 4: Prepare the Golang Project

- Make sure your Golang project is structured according to the GOPATH convention.
- Ensure your project contains a sonar-project.properties file in the root directory. This file is used by SonarScanner to configure the analysis.

Step 5: Configure SonarQube Analysis

- Open the sonar-project.properties file and configure it according to your Golang project:

```
# Project identification
sonar.projectKey=my_project_key
sonar.projectName=My Golang Project
sonar.projectVersion=1.0
# Path to the project sources
sonar.sources=.
# Define the language
sonar.language=go
# Define the Go import path (optional)
sonar.go.goroot=/usr/local/go
sonar.go.gopath=/path/to/your/gopath
#Sonar.Exclusions are required otherwise, sonarwube would consider the test and vendor files as source files and ask for coverage for those.
sonar.exclusions=**/*_test.go,**/vendor/**,**/testdata/*,**/*.sql,**/*.json,**/*.yml

# Additional configuration options (optional)
# sonar.go.tests=./path/to/tests
sonar.test.inclusions=**/*_test.go
sonar.test.exclusions=**/vendor/**
sonar.go.coverage.reportPaths=/path/to/coverage_reports
```

- Customize the properties according to your project structure and requirements.

Step 6: Run SonarScanner

Open a terminal and navigate to the root directory of your Golang project.

Run the SonarScanner command:

```
sonar-scanner
```

SonarScanner will analyze your Golang project and send the results to the SonarQube server.

Step 7: View Analysis Results in SonarQube

Go back to the SonarQube web interface at http://localhost:9000 (or the address where your SonarQube server is running). You should see the analysis results for your Golang project under the project key you specified in the sonar-project.properties file. You can now investigate your Golang project's code quality metrics, possible problems, and other analysis results in SonarQube.

Please be aware that SonarGo is a third-party plugin and might not be as complete as the language analyzers that come with the program. In addition, Golang support could be less extensive than that of JVM-based languages like Java. SonarGo can still offer insightful information on the caliber of the code in your Golang projects, nevertheless.

SonarQube with Docker Compose

Developers may package, distribute, and run programs and their dependencies in separate containers with the help of Docker and Docker Compose, two potent tools for containerization. Here is a quick rundown of the available utilities along with installation instructions for Ubuntu computers:

Docker

You can develop, launch, and use apps inside containers with the help of the open-source Docker platform. Containers are self-contained, lightweight, and portable units that come pre-configured with all the applications, libraries, and software needed to operate them. Docker facilitates the creation, testing, and deployment of applications by offering a standardized environment at every level of the software development lifecycle.

Docker Compose: A tool for creating and overseeing multi-container Docker apps is called Docker Compose. It enables you to run a complicated application with several interconnected containers by using a YAML file to describe the services, networks, and volumes. Several containers and their settings may be more easily managed as a single, integrated application with Docker Compose.

Installing Docker and Docker Compose on Ubuntu

Step 1: Update System Packages

Execute the below commands in an open terminal window to install the necessary dependencies and update the package index:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Step 2: Install Docker

To install Docker, run the following commands:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Docker will be downloaded and installed on your Ubuntu machine as a result. Add your user to the "docker" group after installation to enable executing Docker commands without requiring sudo:

```
sudo usermod -aG docker $USER
```

Step 3: Install Docker Compose

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

This will trigger your Ubuntu computer to download and install the most recent version of Docker Compose.

Step 4: Verify Installations

Use these commands to confirm that Docker and Docker Compose are installed correctly:

```
docker --version
docker-compose –version
```

The corresponding Docker and Docker Compose versions should be shown in the terminal.

And that's it! Having successfully installed Docker and Docker Compose on your Ubuntu computer, you are now able to create, manage, and launch containerized apps.

Install SonarQube using docker-compose.yml

By using the official SonarQube Docker image from Docker Hub and configuring a Docker Compose file to set up the SonarQube service, you may deploy SonarQube using Docker Compose. Here's a detailed tutorial on using Docker Compose to deploy SonarQube:

Step 1: Create a Docker Compose File

Create a new file named docker-compose.yml in your desired directory and add the following content:

```
version: '3'
services:
sonarqube:
image: sonarqube:latest
container_name: sonarqube
ports:
- "9000:9000"
networks:
- sonarqube_network
volumes:
- sonarqube_data:/opt/sonarqube/data
- sonarqube_logs:/opt/sonarqube/logs
- sonarqube_extensions:/opt/sonarqube/extensions
networks:
sonarqube_network:
volumes:
sonarqube_data:
sonarqube_logs:
sonarqube_extensions:
```

Step 2: Deploy SonarQube with Docker Compose

To execute the command, open a terminal or command prompt, go to the directory containing the docker-compose.yml file, and type the following:

```
docker-compose up -d
```

After downloading the most recent SonarQube image from Docker Hub, Docker Compose will launch the SonarQube service in the background. "Detached mode," which operates the services in the background, is what the -d flag stands for.

Step 3: Access SonarQube Web Interface

Upon deployment completion, open a web browser and navigate to http://localhost:9000 to view the SonarQube web interface. Use the server's IP address or domain name and the relevant port number instead of localhost if Docker is executing on a remote server or a different port.

Step 4: Configure SonarQube

After you can access the SonarQube web interface, you must finish setting everything up initially.

- Log in to SonarQube using the default credentials: admin/admin.
- Change the admin password to a secure one.
- Create a new project and obtain an authentication token for the project. You will need this token later to analyze your code.

Step 5: Analyze Your Code

Now that SonarQube is operational, you may integrate it with several build technologies, like Maven, Gradle, or npm, to evaluate your code. Generally, you will need to utilize a SonarQube Scanner together with the relevant build tool to examine a project.

And that's it! Using Docker Compose, you can now use SonarQube to examine the quality of the code in your projects. Keep in mind that you might want to think about extra settings and security precautions, such as SSL certificates and appropriate data backups, for production installations.

Pull Request Analysis

A pull request analysis occurs when a pull request is opened and every time a change is pushed to the pull request branch. Analysis results only include issues that have been introduced by the pull request itself. The Quality Gate of your project is used for the pull request analyses. Quality gate on the pull request uses these results to ensure that the code changes introduced are always clean. Only the Quality Gate's conditions applying to new code metrics are used.

In a pull request analysis, new code is defined as the code that has changed in the pull request branch compared to the target branch. Only issues on new code are reported. The pull request analysis results and quality gate status can be reported directly to the configured DevOps platform's interface.

Prerequisites

Before Reviewing your Pull Requests, Confirm That

1. The local repository has the pull request source branch checked out.
2. The pull request target branch is fetched and is available in the local repository.
3. A local repository with legitimate repository metadata (such as the existence of.git folders) is being used for the analysis. Steer clear of any attempt to preview the merge or any activity related to your main branch.
4. The code in the local repository and the remote repository are identical (for example, no code is updated to the local branch on the CI side before analysis after a PR is published).

To set up Pull Request Analysis, Perform the below Steps.

1. Include the SonarQube analysis step in the CI pipeline for your pull request like installing a sonar scanner on the pipeline etc.
2. Verify that the pull request parameters needed for the project analysis are received by SonarScanner.

sonar.pullrequest.key - The pull request's unique identification number. It must match the pull request's key in your DevOps Platform. Usually, it would be the Pull Request number.

Example

```
sonar.pullrequest.key=3
```

sonar.pullrequest.branch - The name of the branch containing the changes that need to be merged.

Example

```
sonar.pullrequest.branch=feature/new-feature
```

sonar.pullrequest.base - The branch (target branch) into which the pull request will be merged.

Example

```
sonar.pullrequest.base=master
```

Automatic detection is overridden when pull request parameters are manually defined. By configuring the pull request analysis and Quality gates, we can mark the build as failed if the Quality metrics are not met. This way we can ensure that we will only commit the clean code to our main baseline [1-8].

Conclusion

SonarQube's mission is to give power to developers first and foster

an open community around code security and quality. At the outset, it can monitor changes in your most important metrics over time and offer insightful information about how your code quality is changing. Additionally, SonarQube can analyze code in a wide variety of programming languages, allowing you to compute vital metrics overall making it the go-to choose for organizations and developers to adopt and embrace it.

References

1. (2024) Introduction to the pull request analysis. https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/pull-request-analysis/introduction/.
2. (2024) Setting up the pull request analysis. sonarsource https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/pull-request-analysis/setting-up-the-pull-request-analysis/.
3. (2024) Overview. SonarSource https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/overview/.
4. (2023) How to Use Sonarqube in Go Project?. WCE https://golang.withcodeexample.com/blog/how-to-use-sonarqube-with-golang/.
5. (2023) How to Use Sonarqube With Docker Compose. WCE https://golang.withcodeexample.com/blog/how-to-deploy-sonarqube-with-docker-compose/.
6. SonarQube. https://en.wikipedia.org/wiki/SonarQube.
7. Code Analysis with SonarQube. https://www.baeldung.com/sonar-qube.
8. (2019) Set up SonarQube for Golang Project. Yubing Hou https://yubinghou.wordpress.com/2019/03/19/set-up-sonarqube-for-golang-project/.