# **CÔNG CỤ PHÂN TÍCH SONARQUBE**

## 1.1. Công cụ phân tích SonarQube

**SonarQube** (trước đây là Sonar) là một nền tảng nguồn mở được phát triển bởi SonarSource để kiểm tra chất lượng mã nguồn đó thực hiện các đánh giá một cách tự động. SonarQube sử dụng phương pháp phân tích tĩnh (static analysis of code) để phát hiện lỗi, mã nguồn thừa không có tác dụng và lỗ hổng bảo mật trên 20 ngôn ngữ lập 9 trình. Là một nền tảng mã nguồn mở giúp nhà phát triển có thể kiểm tra chất lượng mã nguồn của dự án.

**SonarQube** được viết bằng java nhưng hỗ trợ các ngôn ngữ khác nhau: PHP, Ruby, Java (bao gồm cả Android), C#, JavaScript, TypeScript, C/C++, Kotlin, Go, COBOL, PL/SQL, PL/I, ABAP, VB.NET, VB6, Python, RPG, Flex, Objective-C, Swift, CSS, HTML, và XML và hỗ trợ các cơ sở dữ liệu để lưu trữ kết quả: MySql, Postgresql.

**SonarQube** hỗ trợ trong dự án việc đánh giá mã nguồn theo các tiêu chuẩn của từng ngôn ngữ có trong dự án. Bên cạnh đó có thể thực hiện những việc sau:

• Phát hiện lỗi: phát hiện mã nguồn không dùng đến, các mã nguồn bị trùng lặp.

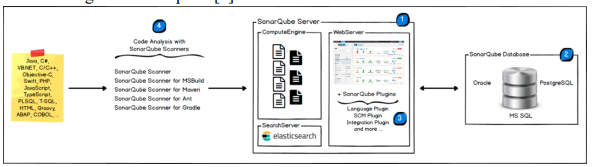
• Tính toán độ bao phủ của mã nguồn theo kiểm thử đơn vị (Unit test - Unittest coverage) – Đó là cách tính toán trong kiểm thử hộp trắng đảm bảo rằng tất cả các trường hợp trong mã nguồn đều được đảm bảo chạy qua khi chạy thực tế.

• Tính toán trong việc các mã nguồn được sử dụng mang tính tạm thời không được tối ưu để sử dụng lại và dễ bảo trì mã nguồn của hệ thống (Technical Debt) – Nghĩa là giải quyết vấn đề một cách nhanh chóng không theo quy chuẩn chỉ quan tâm đến kết quả. Giả sử đúng ra một giá trị phải khai báo là hằng số để sử dụng chung cho các trường hợp khác nhưng khi sử dụng lại dùng luôn giá trị chứ không khai báo biến hằng số. Ví dụ đoạn mã nguồn như sau: if (message == “Nhập thiếu thông tin”) {} thay vào đó có thể khai báo một biến là hằng số trong một lớp java như là Constant.java: public static final String MESSAGE = “Nhập thiếu thông tin”; Như vậy ở bất kỳ đâu cần sử dụng biến MESSAGE đều có thể sử dụng và khi thay đổi giá trị chỉ cần thay đổi ở một chỗ thay vì phải vào nhiều đoạn mã để thay đổi.

• So sánh chất lượng mã nguồn so với các lần kiểm tra trước.

• Phát hiện lỗ hổng bảo mật.

• Kiểm tra độ phức tạp của mã nguồn. Sonar gồm 4 thành phần:

Hình: 4 thành phần chính của SonarQube

➢ Một máy chủ SonarQube bắt đầu với 3 tiến trình chính:

* Máy chủ web cung cấp giao diện người dùng sử dụng và quản lý các phiên bản phân tích mã nguồn.
* Máy chủ tìm kiếm dựa trên Elasticsearch để sao lưu các tìm kiếm từ giao diện người dùng.
* Máy chủ phụ trách xử lý các báo cáo phân tích mã nguồn và lưu chúng vào cơ sở dữ liệu của SonarQube.

➢ Một cơ sở dữ liệu để lưu trữ:

* + Cấu hình của SonarQube (bảo mật, cài đặt plugin,...).
  + Ảnh chụp nhanh chất lượng của dự án, chế độ xem,...

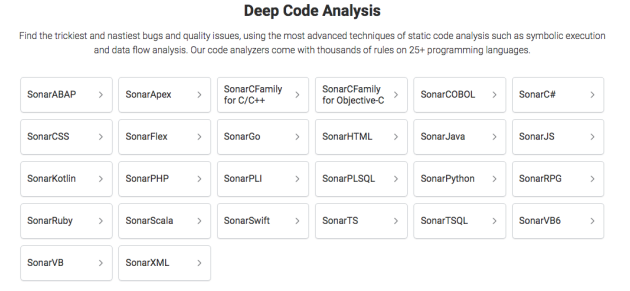
➢ Nhiều plugin SonarQube được cài đặt trên máy chủ, có thể bao gồm ngôn ngữ, SCM, tích hợp xác thực và quản trị.

➢ Một hoặc nhiều SonarScanners chạy trên máy chủ tích hợp liên tục và xây dựng để phân tích dự án.

**SonarQube Scanners** là các plugin sẽ được chạy ở phía người dùng. Giúp thu thập thông tin từ dự án, sinh các “thuộc tính" cho việc phân tích. Sau đó nó sẽ chạy bộ phân tích SonarQube dựa trên các “thuộc tính” này. Tương ứng với mỗi công cụ quản lý dựng (build management), sẽ có một Scanner tương ứng.

**SonarQube Plugins** bao gồm các plugin về ngôn ngữ lập trình, quản lý phiên bản mã nguồn hoặc thậm chí là các plugin cho các công cụ phân tích mã nguồn khác như PMD, FindBugs, jDepend, Android Lint, CheckStyle, WebDriver,…

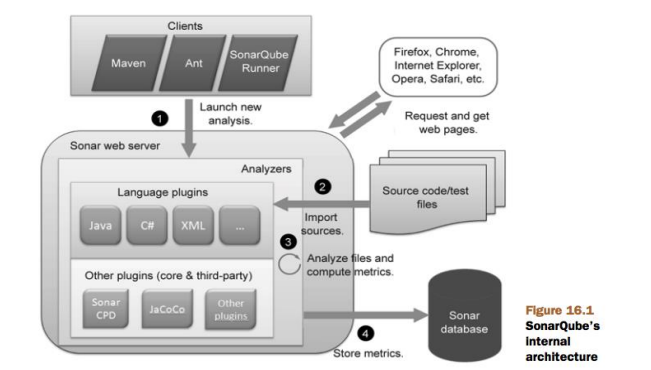
**SonarQube** hiện tại hỗ trợ cho 26 ngôn ngữ lập trình, do vậy sẽ có 26 plugin tương ứng để phân tích mã nguồn cho loại ngôn ngữ đấy.



Hình: Các ngôn ngữ SonarQube hỗ trợ

Một số kỹ thuật phân tích mã nguồn tĩnh bậc cao được sử dụng như: khớp mẫu (pattern matching), phân tích luồng dữ liệu, thực thi tượng trưng. SonarQube công bố các luật (rules) cho các ngôn ngữ này dựa theo các chuẩn nổi tiếng như CWE, SANS, OWASP, MISRA, CERT.

Cách thức hoạt động của SonarQube:



Hình: Cách thức hoạt động của SonarQube

• Sau khi các thông tin dự án được quét, thu thập, xử lý ở máy khách (client).

• **SonarQube Scanner** sẽ gửi những thông tin này ở dạng tệp nén lên máy chủ (server) thông qua các API mà máy chủ cung cấp. Ngay khi máy chủ nhận được yêu cầu xử lý này, việc đầu tiên là SonarQube sẽ nạp toàn bộ mã nguồn và các tệp đi kèm nhờ plugin ngôn ngữ. Sau mã nguồn được nạp xong, các bộ phân tích sẽ lần lượt quét qua mã nguồn và tạo ra chỉ số đo lường hoặc phát hiện các vấn đề. Các bộ phân tích được chia làm hai loại là sensor và decorator. Sensor là các bộ phân tích có thể tự tạo ra và cập nhật các chỉ số từ mã nguồn. Decorator là các bộ phân tích sử dụng chỉ số từ các sensor để phân tích sinh ra các chỉ số ở mức cao hơn.

• Trong toàn bộ quá trình, dữ liệu phân tích sẽ được lưu vào cơ sở dữ liệu ở các bước: sau khi nhập bởi plugin ngôn ngữ, sau khi sensor phân tích, sau khi decorator phân tích. Lúc này người dùng có thể kiểm tra kết quả phân tích trên giao diện web. Thay vì sử dụng trực tiếp các mã lỗi từ đã được định nghĩa sẵn từ các nguồn mở như CWE, OWASP,... SonarQube đã tạo ra những mã lỗi của riêng mình bằng cách tự tổng hợp lại từ nhiều nguồn khác nhau. Sau đó, các bộ lọc, bộ kiểm tra (luật) sẽ được viết ra dựa trên các mã lỗi riêng như vậy. Các luật của Java được triển khai tại org.sonar.java.checks. Để kiểm tra các lỗi có trong một tệp dữ liệu, SonarQube sẽ có hai phương pháp để thực hiện điều này:

• Sử dụng cú pháp cây và API cơ bản (Using syntax trees and API basics): Trước khi chạy bất kỳ quy tắc nào, SonarQube Java Analyzer phân tích tệp 12 Java đã cho và tạo ra cấu trúc dữ liệu tương đương: cú pháp cây (Syntax Tree). Mỗi cấu trúc của ngôn ngữ Java có thể được biểu diễn bằng một thành phần cụ thể cú pháp cây (Syntax Tree), chi tiết từng đặc điểm của nó. Mỗi cấu trúc này được liên kết với một giao diện riêng mô tả rõ ràng tất cả các đặc tính của nó. Khi tạo ra một luật, lớp IssuableSubscriptionVisitor sẽ được thực thi, nó cung cấp các phương thức hữu ích để làm rõ các vấn đề cũng như xác định chiến lược sẽ sử dụng khi phân tích một tệp. Nó dựa trên cơ chế đăng ký (subscription mechanism), cho phép chỉ định loại cây mà luật này sẽ phản ứng với. Các nút mà cần đảm bảo sẽ được duyệt được chỉ định bởi một phương thức là nodesToVisit(). Từ đó SonarQube sẽ có được các thông tin để bắt đầu xác định các lỗi có trong mã nguồn hay thực thi các luật.

• Sử dụng API (Using semantic API): Ngoài việc thực thi các luật dựa vào dữ liệu được cung cấp bởi cú pháp cây (syntax tree), thì SonarQube cung cấp càng nhiều thông tin hơn liên quan đến mã nguồn thông qua một mô hình ngữ nghĩa của mã. Tuy nhiên mô hình này hiện chỉ hoạt động với mã nguồn Java. Mô hình ngữ nghĩa này cung cấp thông tin liên quan đến từng ký hiệu được thao tác. Ví dụ, đối với một phương thức, API ngữ nghĩa sẽ cung cấp dữ liệu hữu ích như chủ sở hữu của phương thức, công dụng của nó, các loại tham số và loại trả về của nó, ngoại lệ mà nó có thể ném,...

## 1.2. Cài đặt SonarQube

*Cấu hình yêu cầu:*

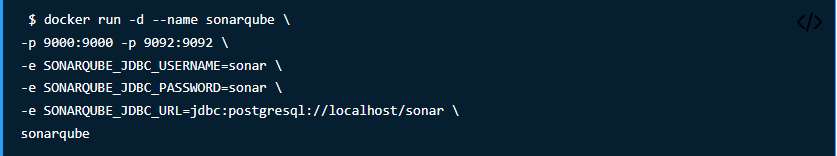
1. Tối thiểu 2GB RAM
2. Dung lượng ổ cứng phụ thuộc vào khối lượng code mà bạn sử dụng SonarQube phân tích
3. Ổ cứng có I/O tốt

*Cài đặt SonarQube khi sử dụng Docker:*

Để cài đặt các bạn sử dụng command sau:



Theo mặc định, Container của Docker sẽ sử dụng H2 Database, vì vậy trong môi trường production các bạn nên cấu hình database. Lúc đấy cài đặt các bạn sử dụng command sau:



Khởi **động** bằng docker:

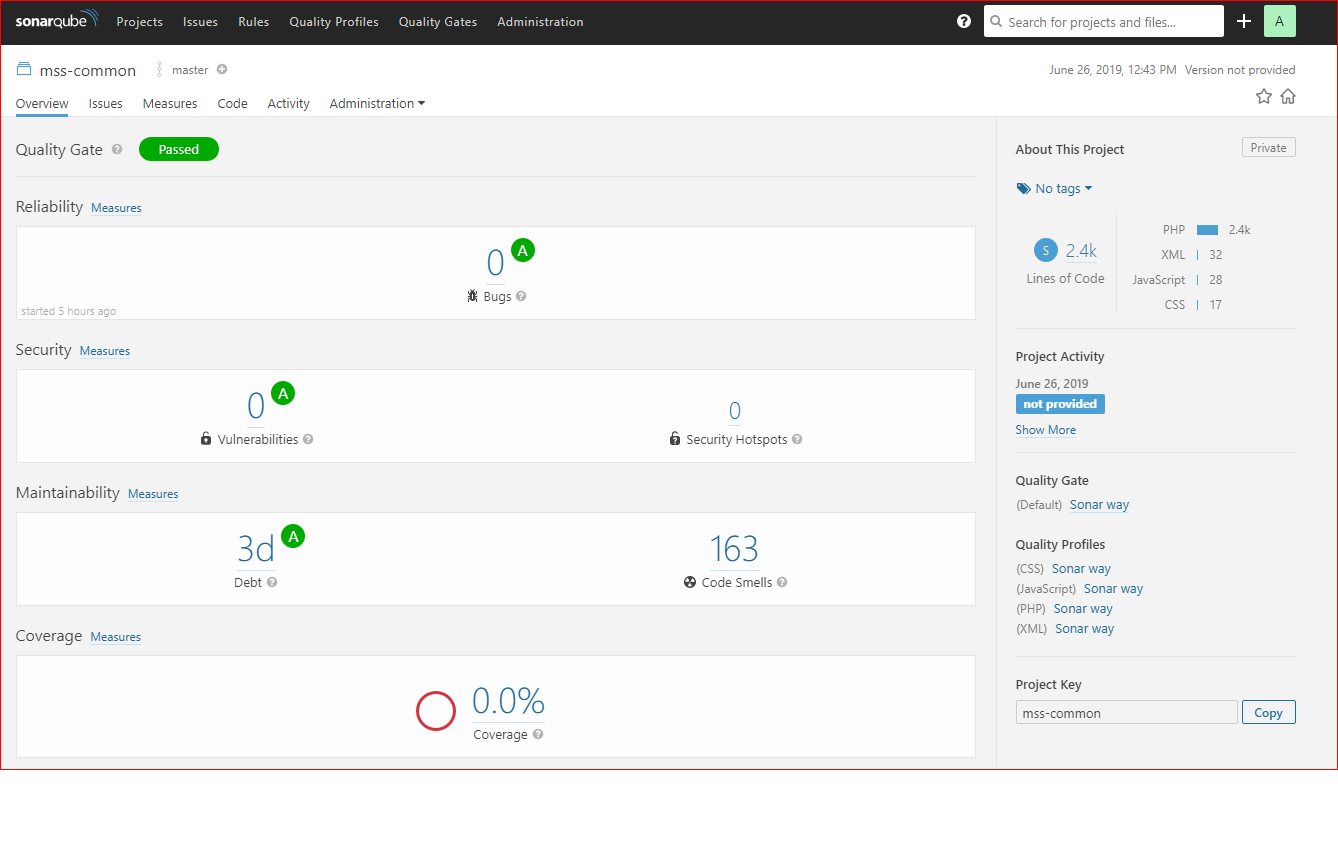


Sau khi cài đặt xong các bạn truy cập bằng đường link: <http://localhost:9000>

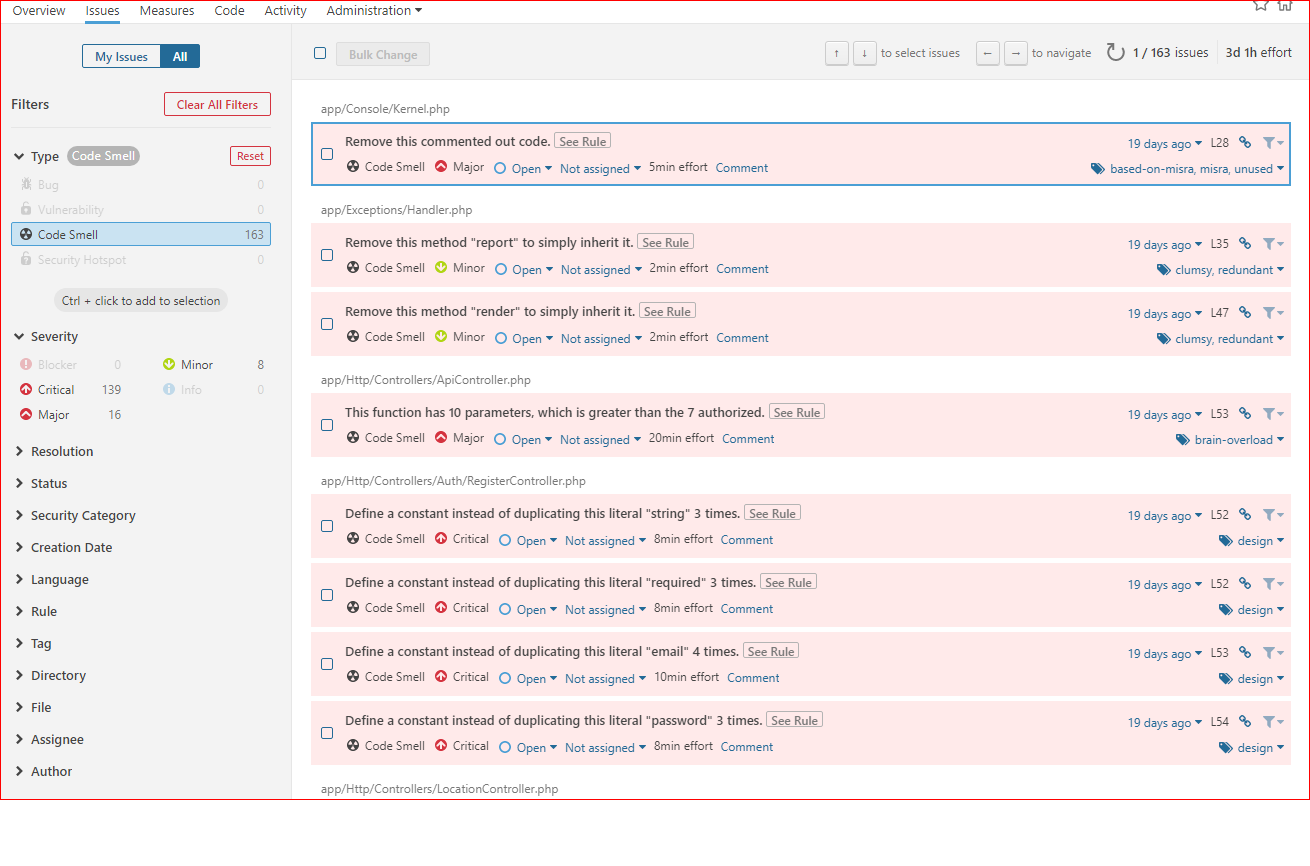
## 1.3 Hướng dẫn sử dụng SonarQube

**Bước 1**: Click vào “Login” và sử dụng tài khoản mặc định “admin” với password “admin” để đăng nhập

**Bước 2**: Điền tên của Project của bạn để tạo token



**Bước 3**: Chọn ngôn ngữ chính cho Project



**Bước 4**: Thực hiện theo thứ tự:

–Chọn hệ điều hành  
– Định nghĩa 1 project key  
– Download Scanner tool.