



# EXCELENCIA

**PROYECTOS DE INNOVACIÓN E INVESTIGACIÓN APLICADAS CON CENTROS:**

***“INCORPORACIÓN DE CONTENIDOS TRANSVERSALES SOBRE CIBERSEGURIDAD EN LOS CICLOS SUPERIORES DE LA FAMILIA DE INFORMÁTICA Y COMUNICACIONES”***

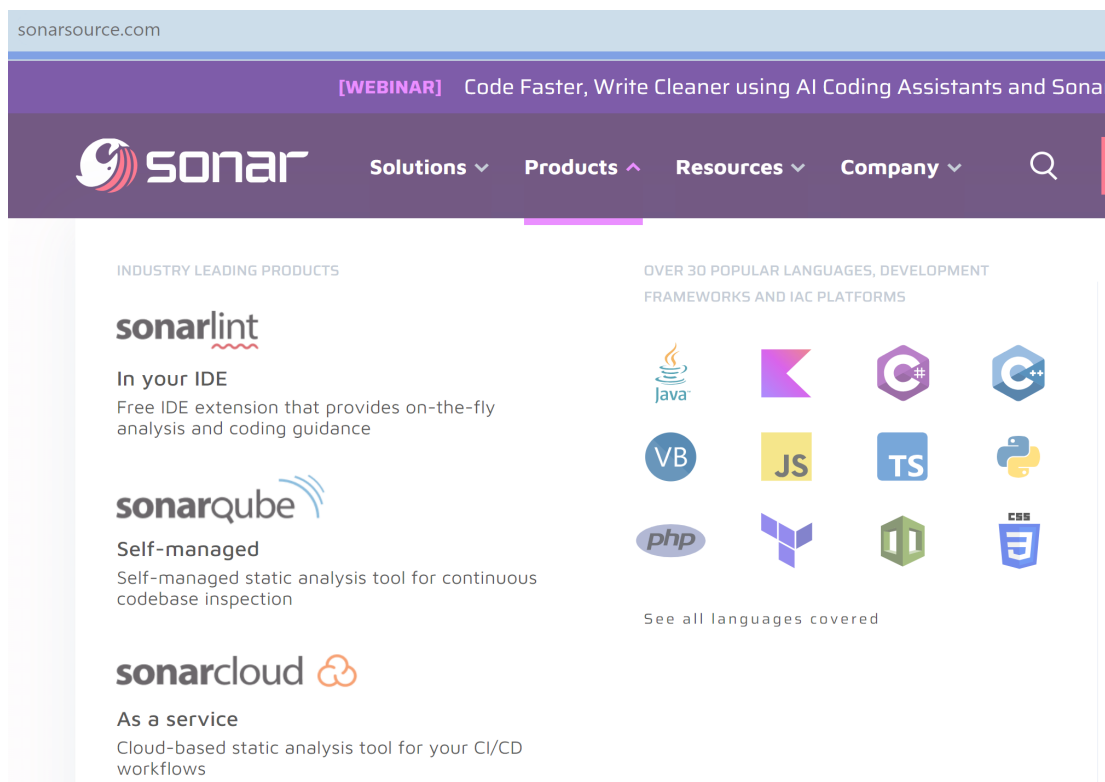
**Laboratorio de Análisis Estático de Código**

1º CFGS Desarrollo de Aplicaciones Web

1º CFGS Desarrollo de Aplicaciones Multiplataforma

# Índice

1. Introducción.....	3
2. Descarga de imagen.....	3
3. Creación de máquina docker con sonarqube.....	3
4. Acceso y configuración.....	3
5. Instalación y configuración del cliente.....	4
6. Realizando Escáner estático de código.....	5
7. Eliminando todos los rastros.....	6
8. Webgrafía.....	6



# 1. Análisis estático de código con plugins Sonar

[Sonar](#) es una compañía que ofrece a las empresas un conjunto de soluciones que permiten a los desarrolladores escribir código limpio (Clean Code) para lograr que todo el código sea apto para el desarrollo y la producción. Con ello se logra que las organizaciones minimicen los riesgos y obtengan software robusto y seguro, lo que daría un prestigio y un valor añadido a sus productos.

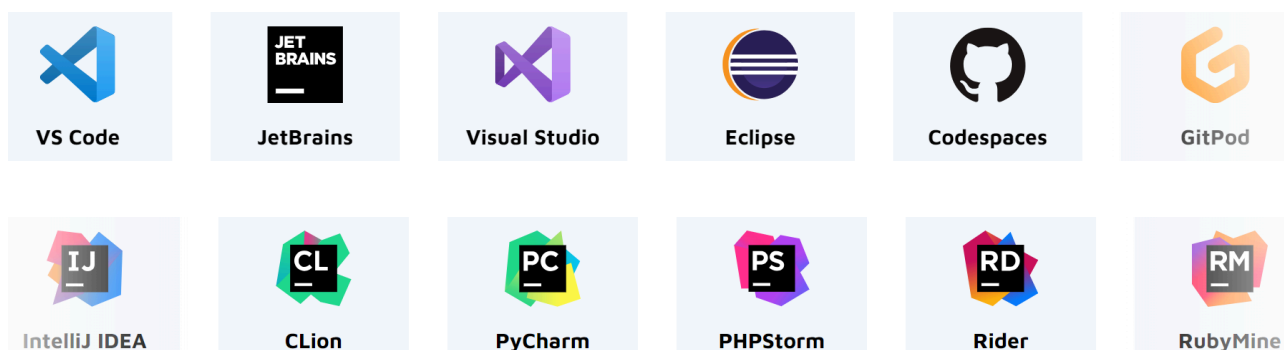
En esta primera parte del laboratorio vamos a descubrir la información que nos ofrece uno de los plugins de Sonar llamado [sonarLint](#) acerca de las vulnerabilidades o avisos acerca del código de nuestros programas.

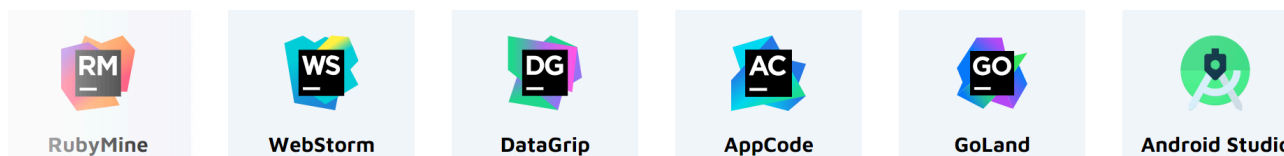
Por lo tanto, lo primero que vamos a hacer en el siguiente punto de este documento, va a ser la instalación del plugin **SonarLint** en nuestro Entorno de desarrollo. Fíjate, disponemos de este plugin tanto para un total de 25 lenguajes de programación así como para los IDEs más populares. Veamos algunos ejemplos:

## - Lenguajes:



## - IDEs:





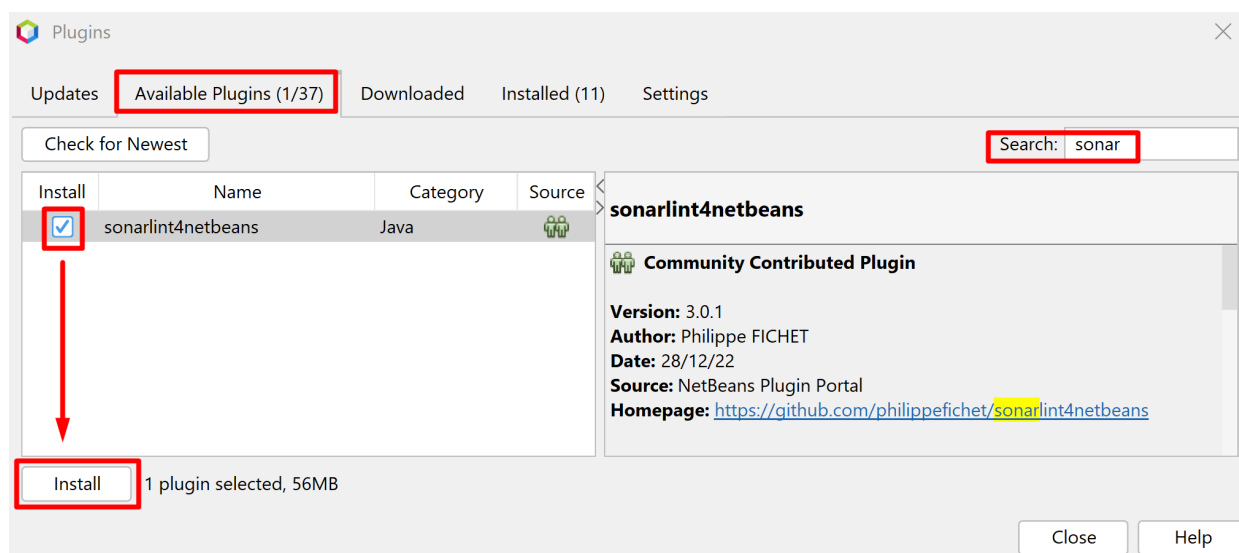
Posteriormente, veremos la información que nos da sobre un proyecto de ejemplo y analizaremos los problemas y sus posibles soluciones.

Si después de este laboratorio tienes más interés sobre los temas que vamos a tratar y quieres profundizar en ellos, puedes descargarte toda la documentación y material necesario desde el apartado [Documentation](#) de su página web.

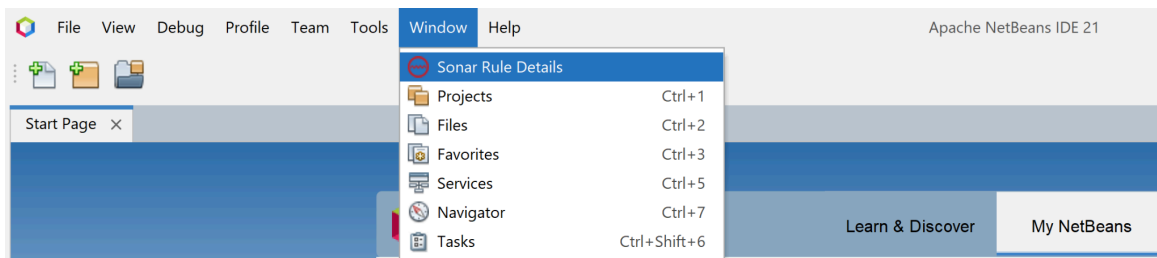
## 2. Instalación de plugins de sonar

SonarLint, como podemos ver en su página web <https://www.sonarsource.com/products/sonarlint/>, es un producto que podemos incorporar a nuestros entornos de desarrollo como extensión o plugin.

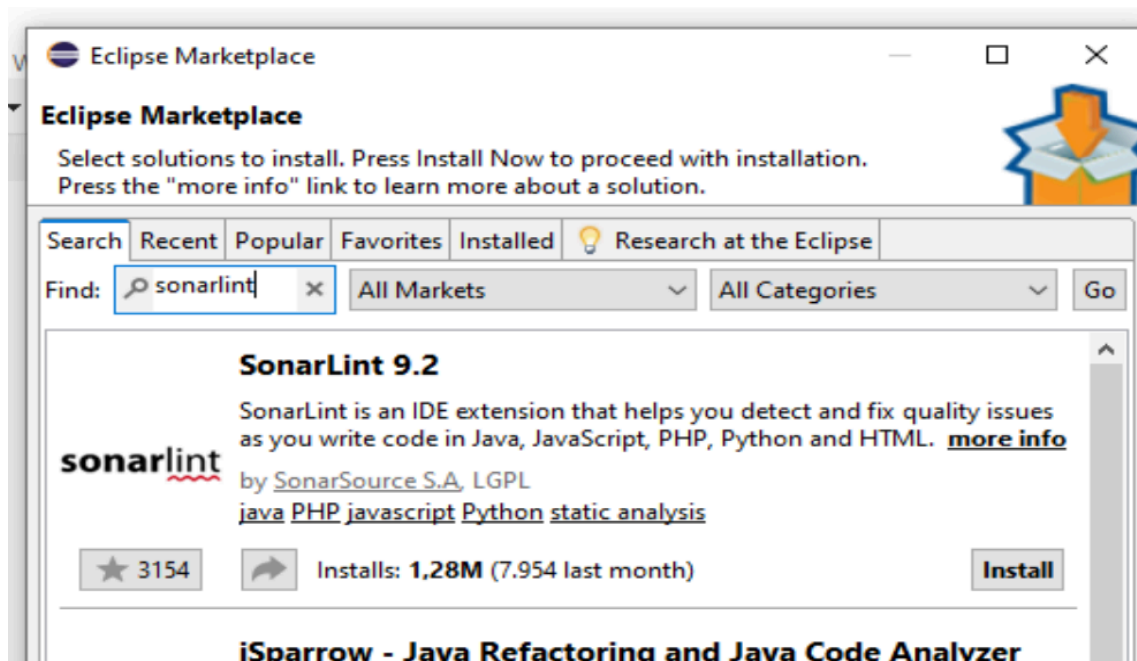
En el caso de [Netbeans](#) lo añadimos desde **Tools -> Plugins -> Plugins Disponibles** y allí, al hacer búsqueda con el término “sonar”, nos aparecerá que el plugin **sonarlint4netbeans** está disponible. Lo seleccionamos y pulsamos sobre el botón “Install”:



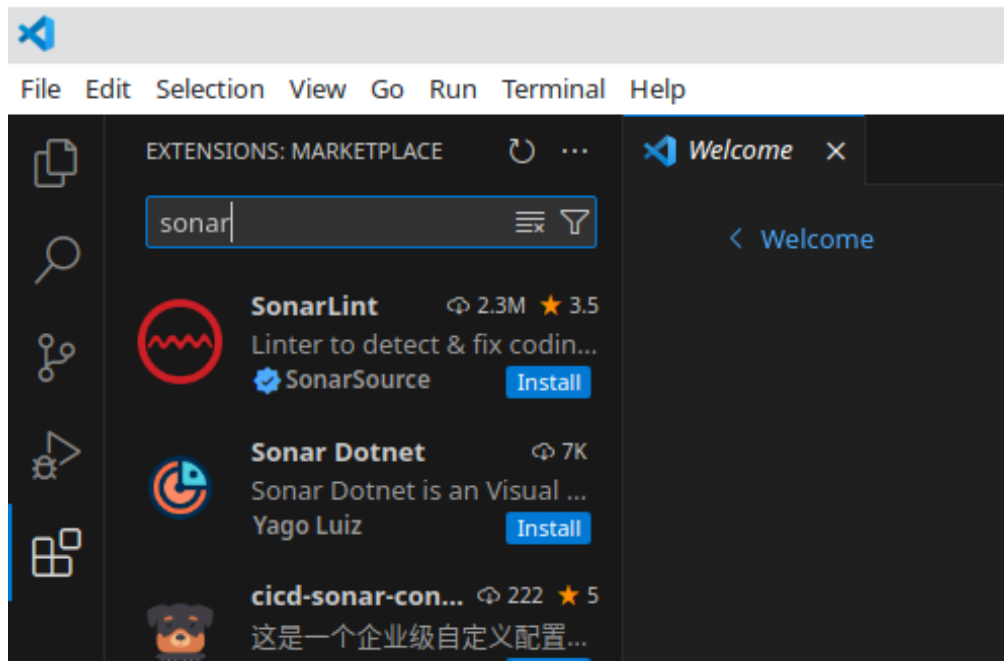
Una vez instalado el plugin se nos reiniciará Netbeans y comprobaremos que ya tendremos el plugin instalado:



En el caso del IDE Eclipse podemos acceder a la instalación desde **Ayuda -> Eclipse Marketplace**. Allí encontraremos la extensión **SonarLint**. La instalamos pulsando sobre el botón **Install**:

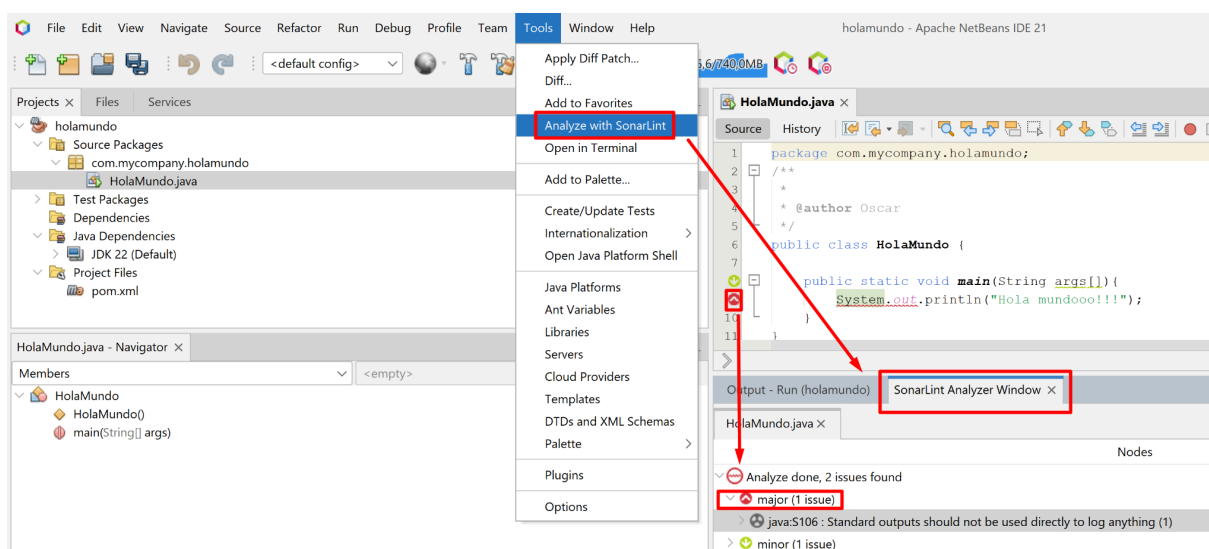


En el caso de **Visual Studio Code** accedemos desde la opción **Extensión** que disponemos en el panel izquierdo. Procedemos de la misma forma que en Eclipse:

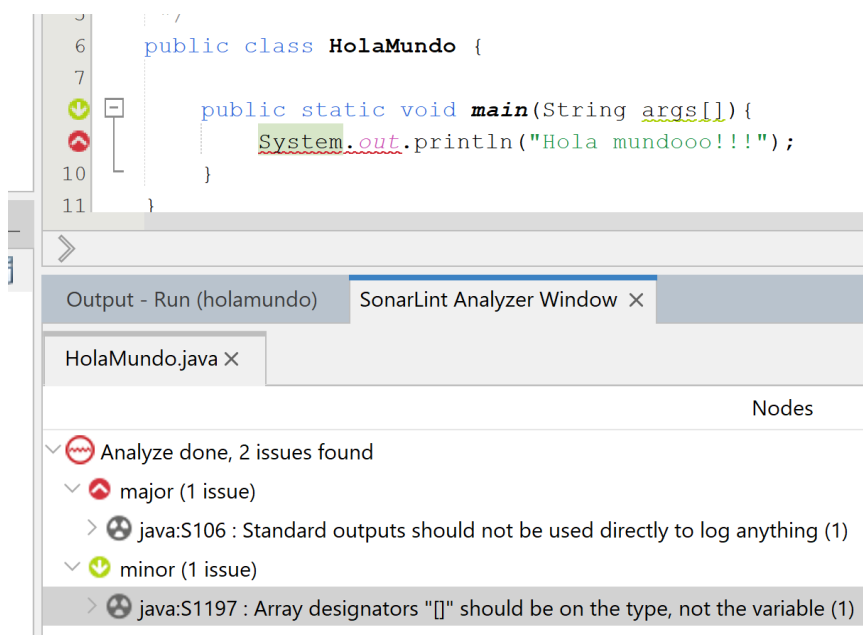


### 3. Utilización del plugin SonarLint

A medida que vayamos escribiendo el código en nuestro IDE, si tenemos instalado el plugin de SonarLint, nos van a aparecer avisos y recomendaciones para mejorar la calidad de nuestro código. Además, también podremos ejecutar un análisis del código que, en el caso del IDE **Netbeans**, lo lanzamos desde el menú **Tools -> Analyze with sonarLint**:



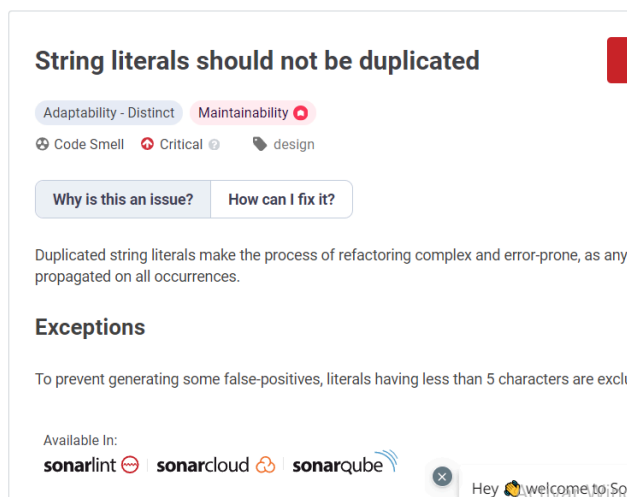
Como podemos ver, en la ventana de sonarLint nos aparecerán todos los avisos agrupados por su gravedad, desde **minor**, como avisos sin importancia, a **blocker**, como elementos con mucha gravedad:



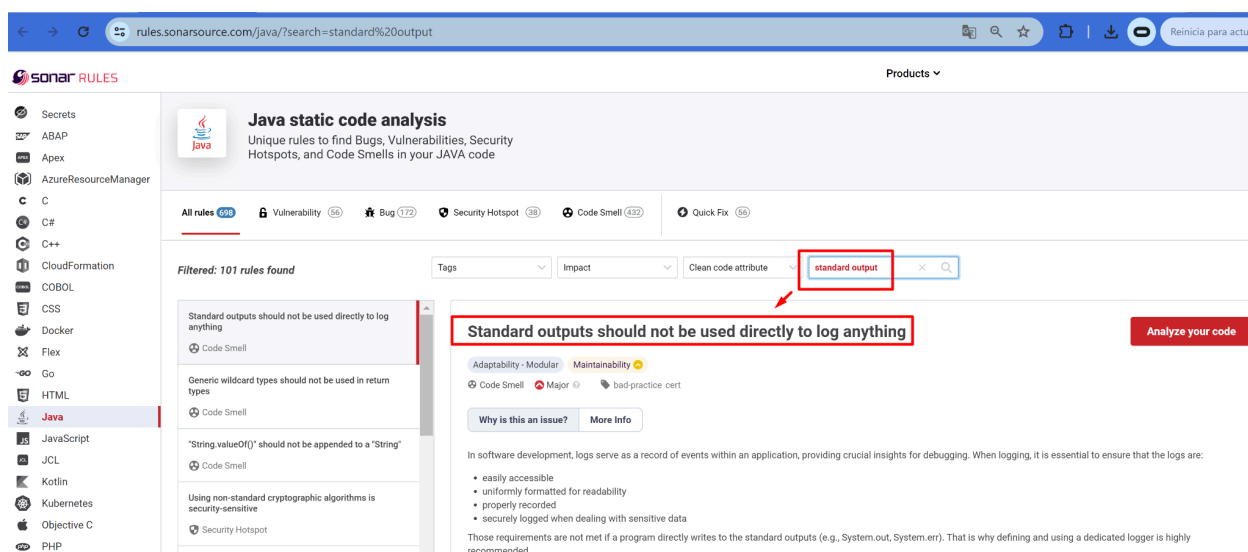


## 4. Interpretación de los avisos de sonar.

Los avisos que nos indica SonarLint son reglas que podemos encontrar para los diferentes lenguajes de programación en la página de SonarSource. En el caso de las reglas de **java** las podemos encontrar aquí: <https://rules.sonarsource.com/java/>.



Aquí podemos encontrar información más detallada sobre cada uno de los avisos que nos han sido generados en el Entorno de Desarrollo:



En la captura vemos por qué es esto un problema. Concretamente nos explica que en el desarrollo de software, los registros (logs) actúan como un registro de eventos dentro de una aplicación, proporcionando información crucial para la depuración. Al registrar (logging), es esencial asegurarse de que los registros (logs) sean:



- fácilmente accesibles
- uniformemente formateados para facilitar la lectura
- adecuadamente registrados
- registrados de manera segura al manejar datos sensibles

Estos requisitos no se cumplen si un programa escribe directamente en las salidas estándar (por ejemplo, `System.out`, `System.err`). Por eso se recomienda definir y usar un registrador (logger) dedicado.

A continuación se nos muestra un ejemplo y su posterior solución. Fíjate en ella e intenta aplicarla posteriormente a nuestro programa:

### Code examples

The following noncompliant code:

```
class MyClass {
    public void doSomething() {
        System.out.println("My Message"); // Noncompliant, output directly to System.out without a logger
    }
}
```

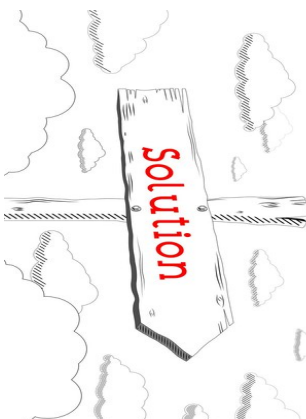
Could be replaced by:

```
import java.util.logging.Logger;

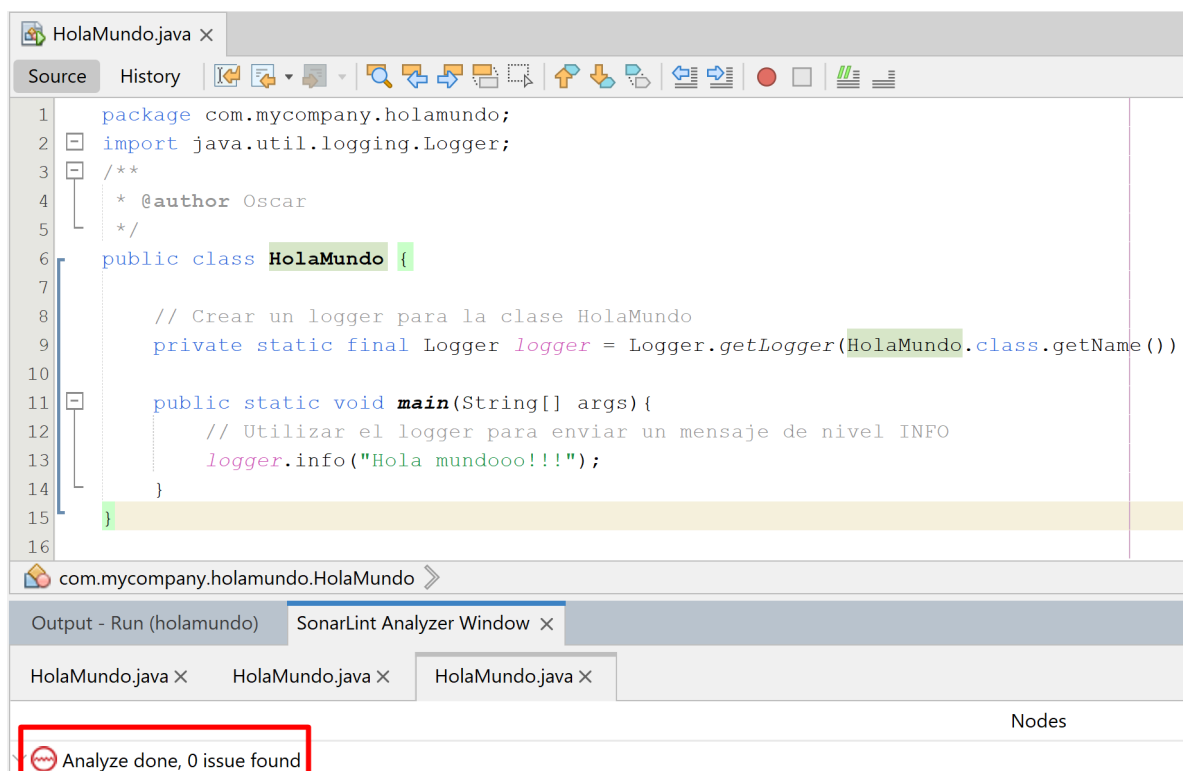
class MyClass {

    Logger logger = Logger.getLogger(getClass().getName());

    public void doSomething() {
        // ...
        logger.info("My Message"); // Compliant, output via logger
        // ...
    }
}
```



Procedemos a solucionar nuestro programa de ejemplo:



```
1 package com.mycompany.holamundo;
2 import java.util.logging.Logger;
3 /**
4  * @author Oscar
5  */
6 public class HolaMundo {
7
8     // Crear un logger para la clase HolaMundo
9     private static final Logger logger = Logger.getLogger(HolaMundo.class.getName());
10
11     public static void main(String[] args){
12         // Utilizar el logger para enviar un mensaje de nivel INFO
13         logger.info("Hola mundooo!!!");
14     }
15 }
16
```

com.mycompany.holamundo.HolaMundo

Output - Run (holamundo) SonarLint Analyzer Window

HolaMundo.java x HolaMundo.java x HolaMundo.java x

Analyze done, 0 issue found

Fíjate que he mejorado ligeramente la solución propuesta por Sonar utilizando los modificadores static y final para la clase Logger. Al usar static y final con Logger, estamos asegurando que todas las instancias de la clase compartan un único recurso de logging inmutable, lo cual es una práctica común y recomendada en muchas aplicaciones Java para mantener la consistencia y eficiencia del logging. Concretamente:

### Uso de static:

- **Propósito:** La palabra clave static se usa para indicar que un campo o método pertenece a la clase, en lugar de a una instancia específica de esa clase.
- **Beneficio:** Cuando se declara un Logger como static, significa que se crea una única instancia de Logger que es compartida por todas las instancias de la clase. Esto es especialmente útil en el contexto del logging porque generalmente no necesitas un objeto logger diferente para cada objeto de la clase; un solo logger puede ser compartido para todos los usos dentro de la clase.
- **Aplicación práctica:** En nuestro caso, usar static permite que cualquier método estático o de instancia dentro de la clase HolaMundo pueda acceder al mismo logger sin necesidad de crear un nuevo objeto Logger cada vez.

## Uso de final

- Propósito: final indica que el valor de un campo no puede cambiar después de su inicialización.
- Beneficio: Declarar un Logger como final asegura que el logger no pueda ser reasignado después de su inicialización inicial. Esto es importante para la integridad de la aplicación, ya que el logger actúa como un recurso central para registrar información y no debería ser susceptible a cambios una vez configurado.
- Aplicación práctica: En nuestra clase HolaMundo, una vez que el Logger es inicializado, marcarlo como final previene que otro código en la clase accidentalmente lo reemplace, lo cual podría llevar a comportamientos de logging inconsistentes o errores.

## 5. Análisis estático de código con SonarQube

Para ello vamos a necesitar crear un contenedor docker donde tendremos el servidor con el escaner estático de código SonarQube y luego usaremos, como cliente, nuestro propio equipo donde tendremos un proyecto en nuestro caso en el lenguaje de programación Java.

## 6. Instalación y configuración de Docker

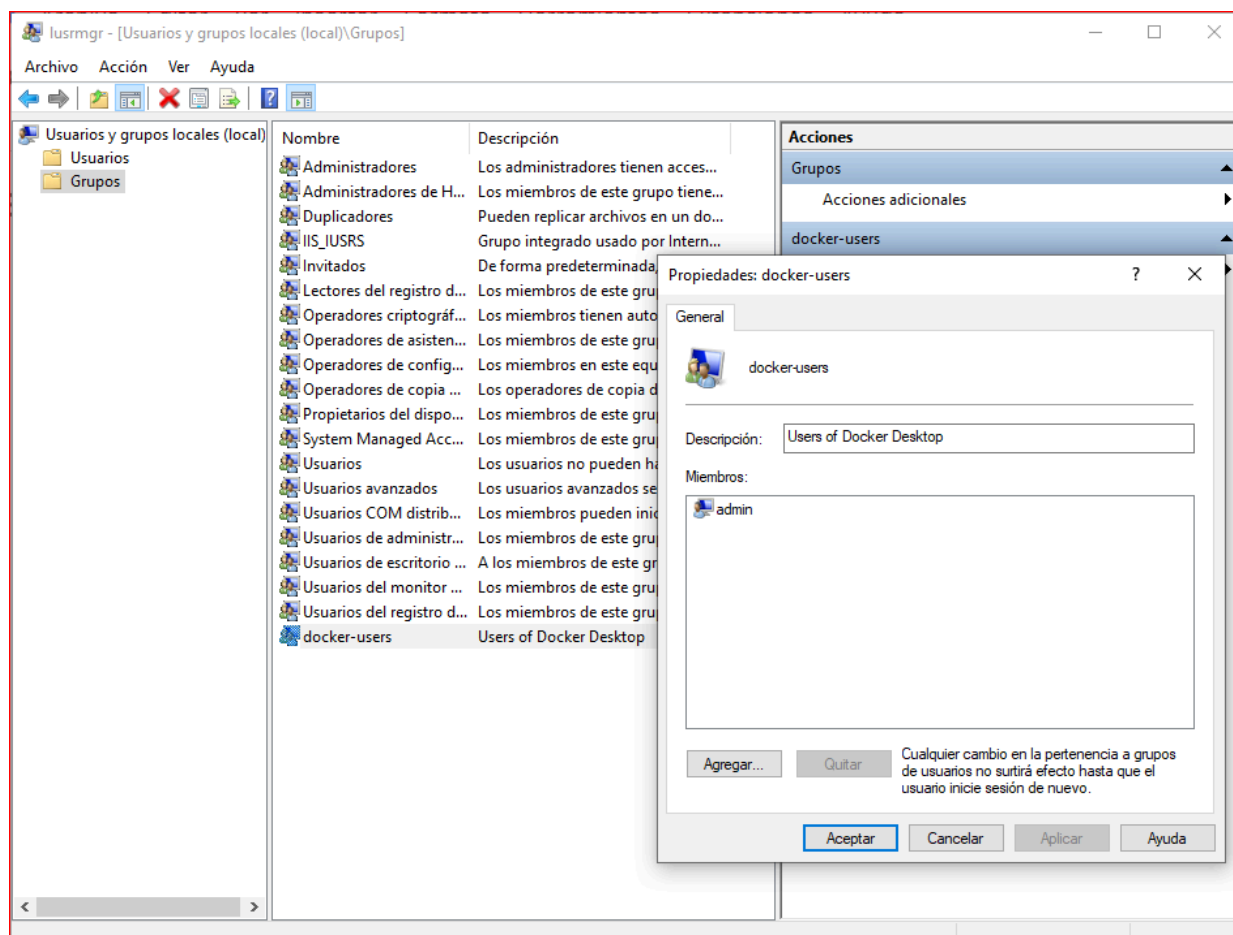
Lógicamente, el primer paso consistirá en instalar Docker en el caso de que no lo tengamos en nuestro equipo. Para ello, podemos seguir los siguientes tutoriales de Docker:

- <https://docs.docker.com/desktop/install/windows-install/>
- <https://docs.docker.com/engine/install/ubuntu/>

o buscar cualquier otro tutorial, hay infinidad de ellos.

Descargamos el instalador de Docker para nuestro Sistema Operativo. Es posible que tengas que autenticarte con una cuenta de docker o Gmail para finalizar la instalación.

Dentro de la configuración de usuarios y grupos, añadimos nuestro usuario al grupo Docker-users (por ejemplo podemos hacerlo abriendo el administrador de usuarios y grupos desde terminar: lusrmgr.msc en windows)



En linux podemos usar el comando **adduser mi\_usuario docker** desde terminal

## 7. Descarga de la imagen Docker de SonarQube

Arrancamos Docker Desktop y descargamos la imagen antes de la creación del contenedor. Ejecutamos desde un terminal de Windows PowerShell:

```
docker pull sonarqube
```

## 8. Creación de máquina docker con sonarqube

Lanzamos la aplicación Docker Desktop y desde un terminal de Windows PowerShell arrancamos el contenedor con el siguiente comando:

```
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube
```

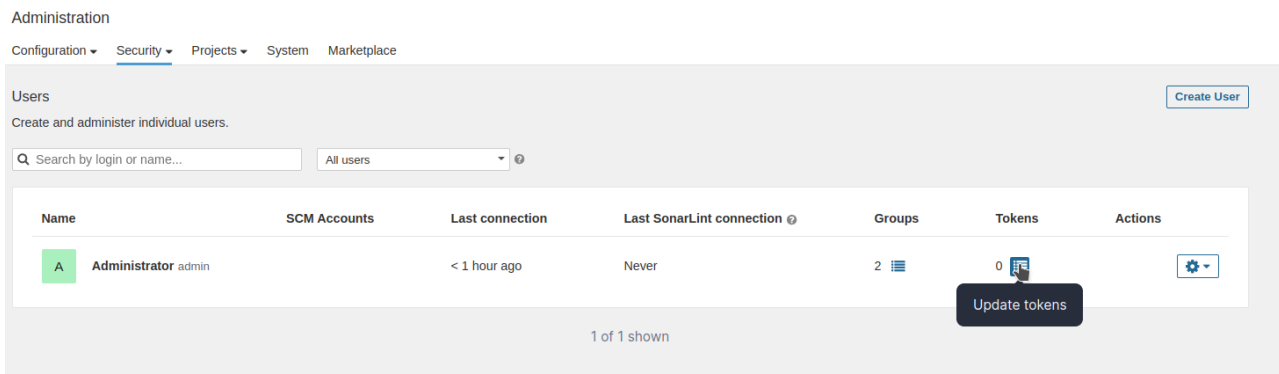
Este comando levanta una máquina virtual con el servidor SonarQube, por lo que es posible que nos salga aviso de firewall o de seguridad de Windows para preguntarnos. Le debemos de dar a permitir.

## 9. Acceso y configuración

Después de unos segundos que tarda en arrancar el contenedor o máquina virtual, accedemos a través del puerto 9000 con ip de la máquina y con usuario **admin** y contraseña **admin**, nos pedirá el cambio de contraseña. (yo en mi equipo le pongo: usuario).

Lo primero nos pedirá cambiar la contraseña de administrador

A continuación deberemos crear un token que permita a nuestra máquina enviar los resultados al servidor. Lo hacemos a través de tokens, por lo que creamos un token nuevo en el servidor: **Administration -> Security -> Users -> Administrator** pulsamos sobre **Token (Update tokens)**.



Administration

Configuration ▾ Security ▾ Projects ▾ System Marketplace

Users Create User

Create and administer individual users.

🔍 Search by login or name... All users

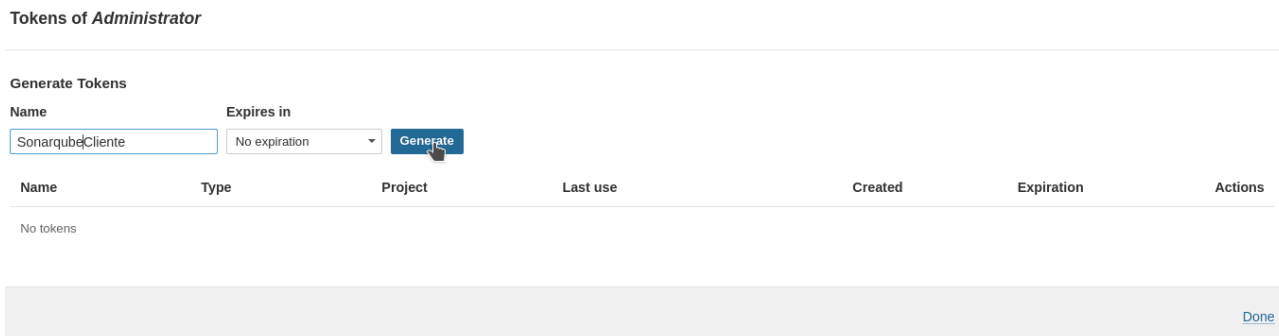
Name	SCM Accounts	Last connection	Last SonarLint connection ⓘ	Groups	Tokens	Actions
<span>A</span> Administrator admin		< 1 hour ago	Never	2	0	<span>⚙️</span>

1 of 1 shown

Update tokens

Se nos abre la ventana “Generate Token”.

Ponemos **nombre de token** (por ejemplo SonarqubeCliente) y el **tiempo de validez** que queramos y le damos a **Generar**.



Tokens of Administrator

Generate Tokens

Name Expires in

SonarqubeCliente No expiration Generate

Name	Type	Project	Last use	Created	Expiration	Actions
No tokens						

Done

Lo siguiente es copiarlo porque tendremos que utilizarlo al realizar el scanner desde el cliente.

En mi caso es este:

#### Tokens of Administrator

##### Generate Tokens

Name

Expires in

Enter Token Name

30 days

Generate



New token "SonarqubeCliente" has been created. Make sure you copy



Copy

squ\_30a4c65317b290fc0f2daf226d4c69a1b6d5385a

squ\_30a4c65317b290fc0f2daf226d4c69a1b6d5385a para esta prueba. Lo podéis copiar en algún lugar, ya que luego no podremos acceder a él y si no lo hemos guardado en algún sitio tendremos que eliminarlo y crear uno nuevo.

#### Tokens of Administrator

##### Generate Tokens

Name

Expires in

Enter Token Name

30 days

Generate

Name	Type	Project	Last use	Created	Expiration	Actions
SonarqubeCliente	User		Never	April 10, 2024	—	<a href="#">Revoke</a>

Lógicamente si estamos trabajando de forma seria, no es una buena idea guardarlo....

## 10. Instalación y configuración del cliente

El siguiente paso será instalar sonar-scanner-cli en nuestro equipo para poder enviar al servidor nuestros escaneos.

Las descargas las tenemos en la parte superior de la web con el siguiente enlace:  
<https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/scanners/sonarscanner/>

Descargamos el escaner correspondiente a nuestro sistema operativo.

Una vez descargado nos lo llevamos a la carpeta que queramos, yo por ejemplo en linux me la llevo a /opt y allí la descomprimo:

```
sudo mv sonar-scanner-cli-5.0.1.3006-linux.zip /opt
```

```
sudo unzip sonar-scanner-cli-5.0.1.3006-linux.zip
```

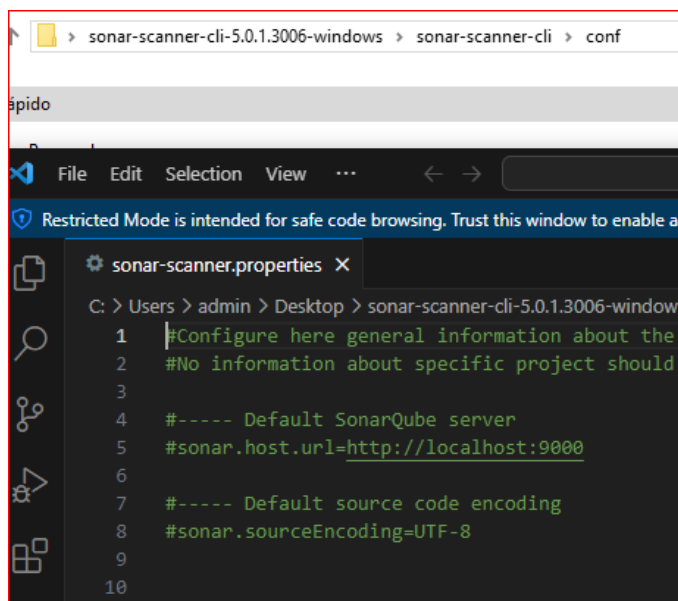
Añadimos la ruta del directorio donde hemos descomprimido al PATH, para que se pueda ejecutar.

En **Linux** algo así en el terminal:

**PATH=\$PATH:/opt/sonar-scanner-5.0.1.3006-linux**

En Windows editamos la variable de entorno del usuario (podemos verlo en [este enlace](#)).

**Solamente en el caso de que SonarQube no esté en nuestra máquina anfitriona**, es decir en <http://localhost:9000>, y lo tengamos en otra máquina diferente, debemos configurar la ip de la máquina donde lo tenemos dentro del archivo de configuración



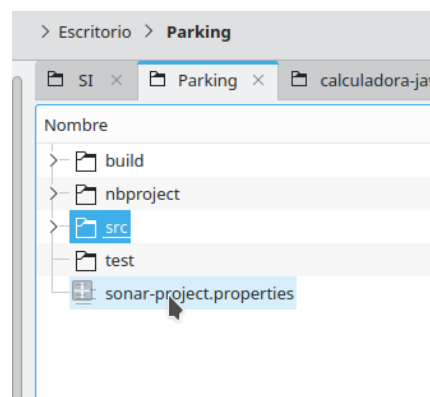
**sonar-sccaner-cli/conf/sonar-scanner.properties**. Allí descomentamos la línea de **sonar.host.url** y le ponemos la dirección ip que deseemos.

## 11. Realizando Escáner estático de código

Pues ya sólo nos queda elegir el proyecto que queremos escanear, configurar el archivo de propiedades de sonar y ejecutar el escaneo.

Dentro de la carpeta raíz de proyecto debe de haber un fichero con nombre sonar-project.properties con un contenido parecido a este:

```
sonar.projectKey=parking
sonar.projectName=parking
sonar.lenguaje=java
sonar.test=test
#sonar.sources=src/parking
sonar.java.binaries=src/parking
#sonar.exclusions=venv/**, .idea/**, .git/**
#sonar.python.version=3.9
```





Vemos que algunas líneas están comentadas... lo principal: sonar.projectName es el nombre del proyecto que nos va a aparecer en SonarQube después del escaneo, y en sonar.java.binaries ponemos la carpeta donde están los binarios de java.

Para realizar escaneo, desde un terminal y situados en la carpeta raíz del proyecto ejecutamos:

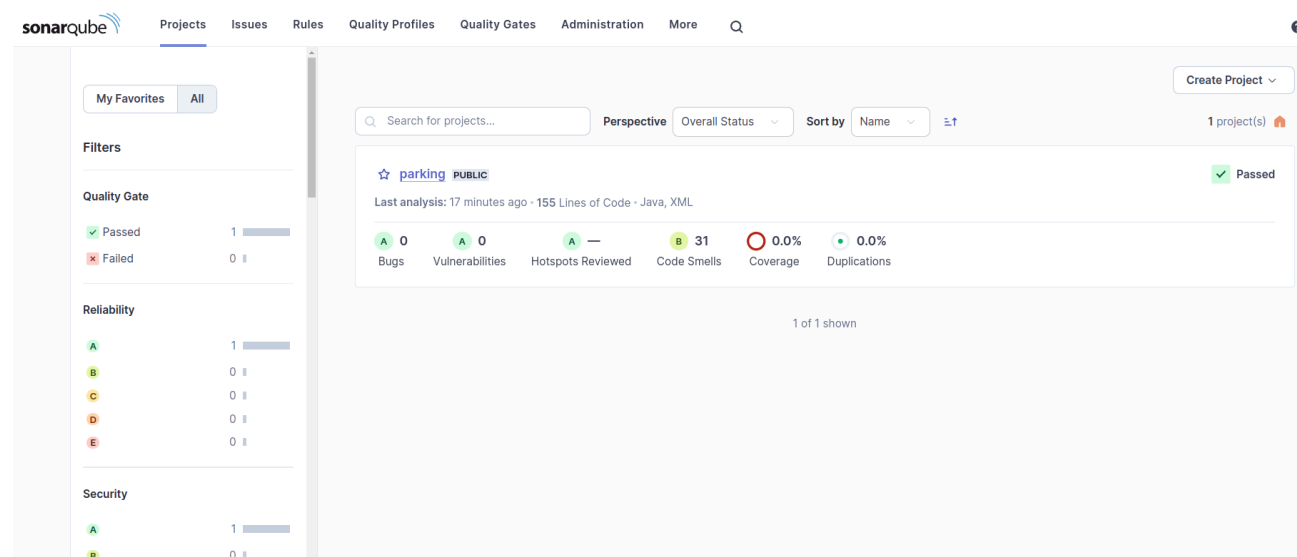
**sonar-scanner -D sonar.login=squ\_30a4c65317b290fc0f2daf226d4c69a1b6d5385a**

hay que darse cuenta que yo he puesto mi token, tú tendrás que poner el token que has generado en el apartado 4.

```
jmmedinac03@jmmedinac03-victus:~/Escritorio/Parking$ sonar-scanner -D sonar.logi
n=squ_30a4c65317b290fc0f2daf226d4c69a1b6d5385a
INFO: Scanner configuration file: /opt/sonar-scanner-5.0.1.3006-linux/conf/sonar
-scanner.properties
INFO: Project root configuration file: /home/jmmedinac03/Escritorio/Parking/sona
r-project.properties
INFO: SonarScanner 5.0.1.3006
INFO: Java 17.0.7 Eclipse Adoptium (64-bit)
INFO: Linux 5.15.0-101-generic amd64
INFO: User cache: /home/jmmedinac03/.sonar/cache
```

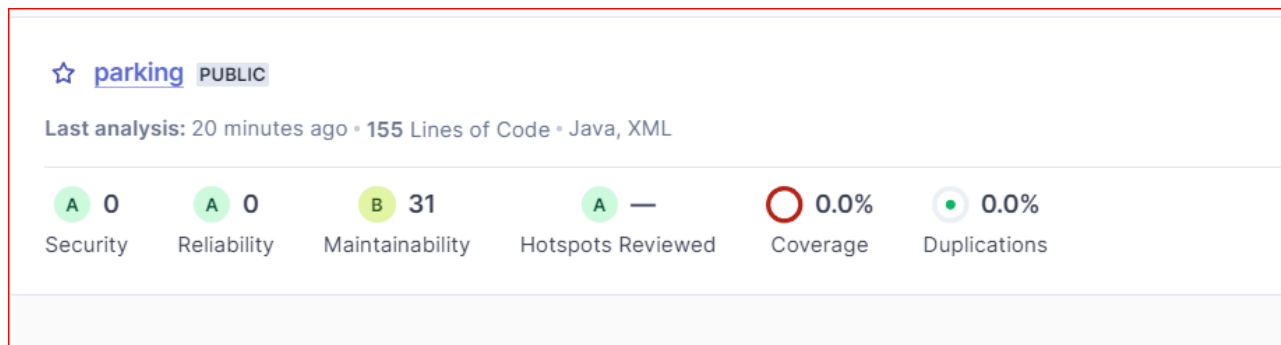
Si no tenemos ningún error ya podemos ir a ver los resultados del escaneo a nuestro servidor SonarQube.

En este caso vemos que obtenemos buenos resultados:



Metric	Value
Bugs	0
Vulnerabilities	0
Hotspots Reviewed	0
Code Smells	31
Coverage	0.0%
Duplications	0.0%

## 12. Analizando los resultados



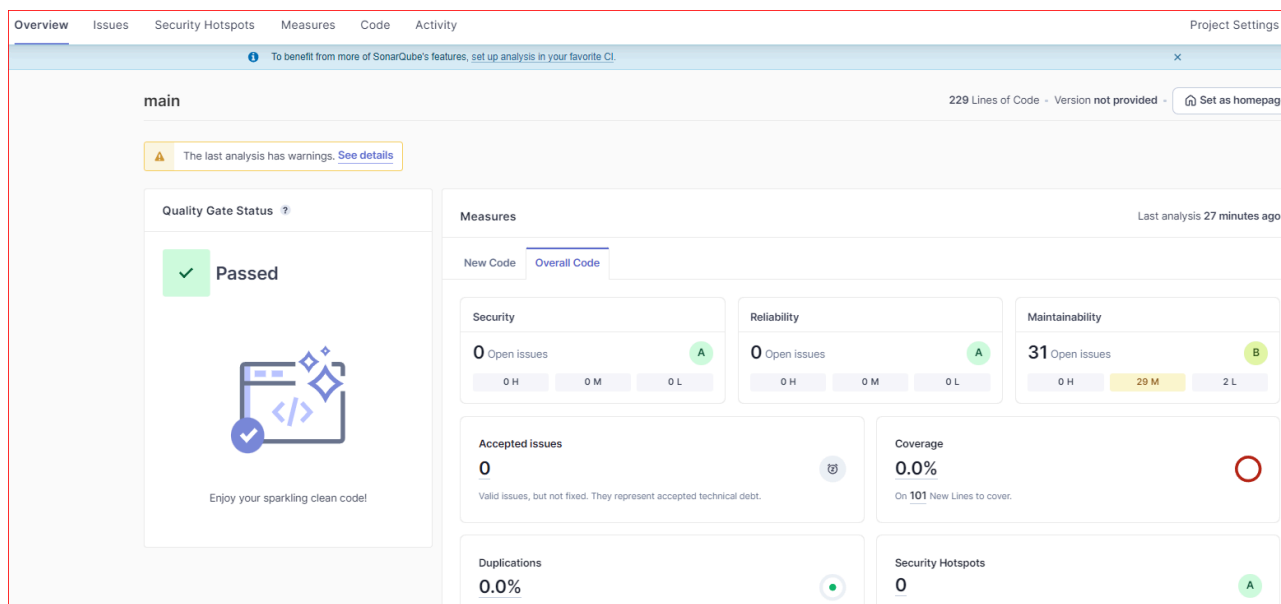
Dentro de la pestaña o sección de **Projects** podemos ver los resultados del escaneo.

Tenemos información sobre 5 métricas o apartados: **Seguridad**, **Fiabilidad**, **Mantenibilidad**, **Hotspots** o Puntos de Accesos asegurados, Porcentaje de **cobertura de las pruebas** de software y Porcentaje de líneas de código duplicadas.

En cada una de estas métricas podemos ver el número de problemas o avisos encontrados, así como el grado o calificación que se le otorga (A excelente, B bueno, etc.)

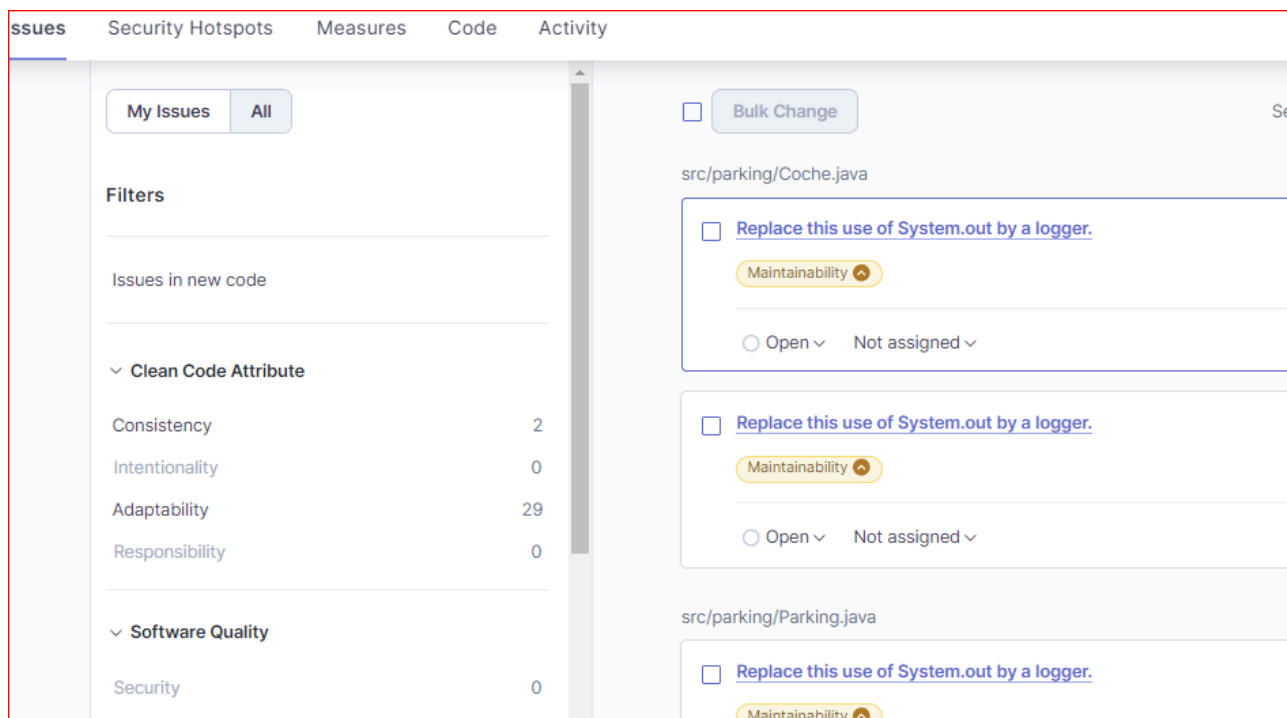
Como vemos esta información nos da una medida de la calidad del software.

Si queremos ver información más detallada, entramos dentro del proyecto y podremos ver la información en determinados aspectos. En **Overview** vemos información general:



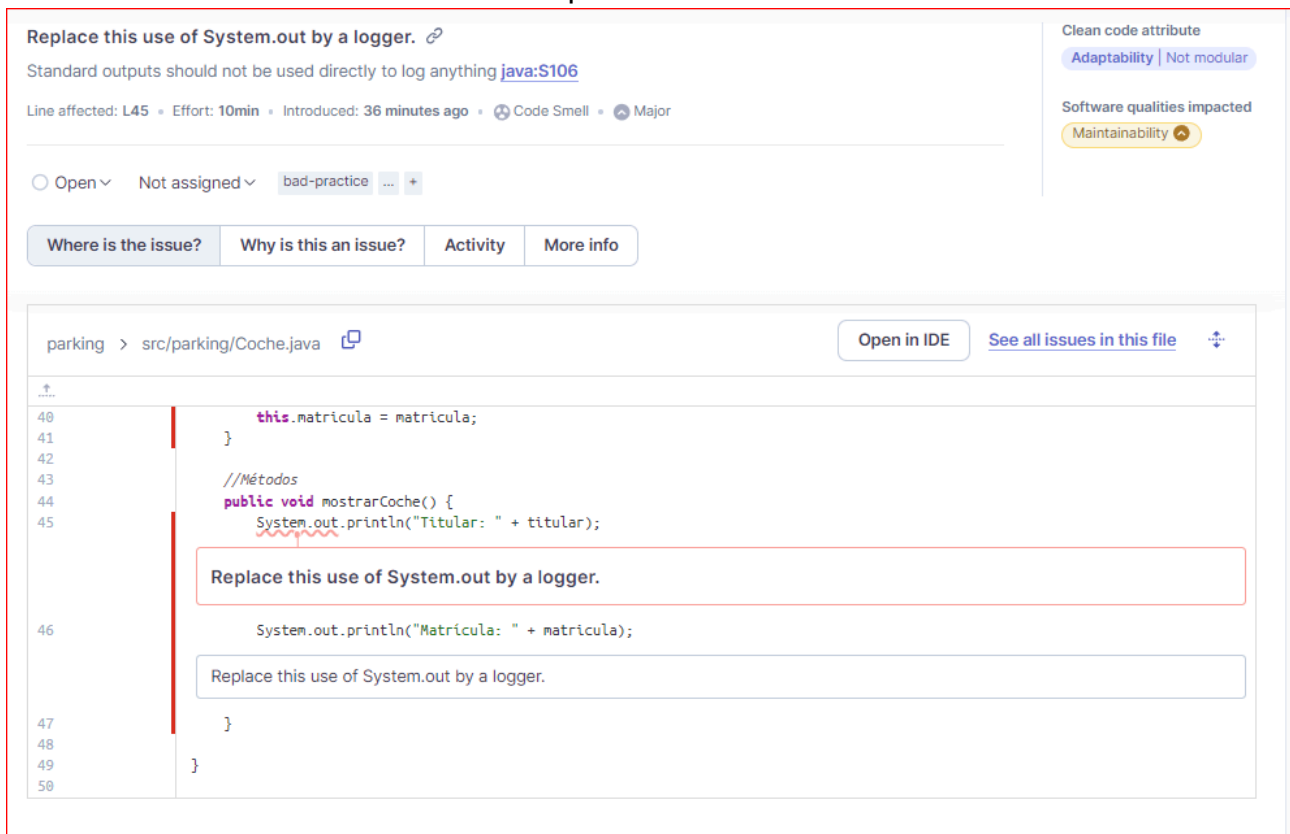
Junto a cada una de las métricas a las que nos referíamos, encontramos información de cuánto tiempo tardaríamos en solucionar estos problemas. En el caso de ejemplo nos encontramos con 31 avisos que se estima se tardaría 29 minutos en reescribirlos de una forma más correcta.

Si queremos ver información más detallada acerca de los avisos o problemas encontrados nos vamos a **Issues**, donde se muestra en forma de lista.



Si clicamos en ellos podemos ver información sobre el aviso, en cuatro pestañas distintas que nos da información sobre: **¿Dónde está el problema?, ¿Por qué es un problema?, Actividad y Más información.**

Dentro de esas secciones nos aportan la información de cómo solucionarlo.



The screenshot shows a SonarQube issue page. At the top, it says "Replace this use of System.out by a logger." with a link to the rule. Below that, it states "Standard outputs should not be used directly to log anything" and "java:S106". It also shows "Line affected: L45", "Effort: 10min", "Introduced: 36 minutes ago", "Code Smell", and "Major". On the right, there are tabs for "Clean code attribute" (Adaptability, Not modular) and "Software qualities impacted" (Maintainability). Below the issue details, there are tabs for "Where is the issue?", "Why is this an issue?", "Activity", and "More info". The main part of the page shows the code in a file named "src/parking/Coche.java". The code is as follows:

```
40      this.matricula = matricula;
41    }
42
43    //Métodos
44    public void mostrarCoche() {
45        System.out.println("Titular: " + titular);
46
47        System.out.println("Matricula: " + matricula);
48    }
49
50 }
```

There are two red boxes highlighting the lines of code that trigger the issue: line 45 and line 46. Each box contains the text "Replace this use of System.out by a logger."

## 13. Eliminando todos los rastros.

Eliminamos el contenedor docker: **docker rm -f sonarqube**

Eliminamos la imagen descargada: **docker image rm -f sonarqube**

## 14. Webgrafía

Aparte de las páginas web de docker: <https://www.docker.com/> y SonarQube: <https://www.sonarsource.com/products/sonarqube/>, en la siguiente web se puede ver todo el proceso <https://elwillie.es/2022/12/11/sonarqube-introduccion-e-instalacion-de-sonarqube-sonarscanner-cli-y-sonarlint/>