Procesos de Desarrollo de Software

Logging y SAST

Sesión Práctica

Contenidos

PARTE I. Logging y trazabilidad

1. Conceptos y motivación del logging.
2. Configuración de log en aplicaciones.
3. Explotación y análisis del log.

Parte II. SAST y calidad

1. Análisis estático de código (SAST).
2. Configuración de SonarQube.
3. Integración en el ciclo CI/CD.

PARTE I
Logging y trazabilidad

Trazabilidad en el software

Logging – Motivación

Mientras aprendemos a programar solemos usar la salida de consola para obtener retroalimentación, pero…

- La salida de consola no se guarda.
- No podemos distinguir entre niveles de log.
- No podemos usar configuraciones diferentes según el entorno.
- Puede saturar la salida estándar y generar problemas de concurrencia con otros procesos.

Nunca se debe usar la salida por consola en un desarrollo.
Se debe usar siempre un sistema de log.

Logging – Motivación

El **log** es el **registro de eventos relevantes** del sistema que ocurren durante la ejecución de una aplicación.

Mediante el logging:

- **Generamos** mensajes informativos sobre lo que ocurre.
- **Formateamos** los mensajes para que sean homogéneos.
- **Clasificamos** los mensajes según su importancia.
- **Explotamos** la información de los registros para conocer lo que ha pasado en una ejecución concreta.

Logging – Especificación

Los formatos, niveles y severidad de los logs siguen estándares definidos en RFCs como RFC 5424 (Syslog).

Mediante el log, indicaremos:

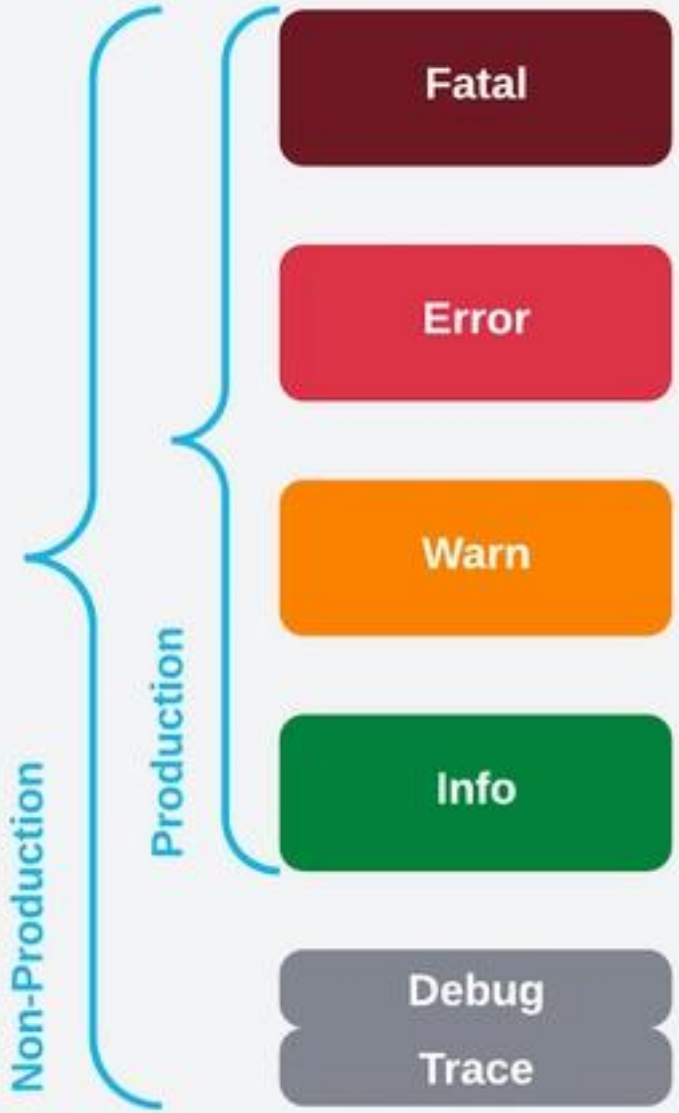
- **Nivel de log**
  Clasifica los mensajes según su importancia.
- **Fecha de log**
  Fecha en la que se emite el registro. Nos permite ordenar los mensajes.
- **Mensaje**
  El texto que acompañará a un mensaje, que puede incluir:
  - Sólo el texto que decidamos poner.
  - El texto y los valores de las variables que queramos registrar.
  - El texto y el mensaje de la excepción que se ha producido.

Logging – Niveles

- Los niveles de log sirven para priorizar y diferenciar los diferentes mensajes.
- Mediante configuración podemos indicar qué mensajes se registran y cuáles no.
  - En un **entorno de pruebas** querremos ver mensajes de `debug`.
  - En un **entorno de producción** los de `info`.
  - Los mensajes de `warning` o `error` siempre es interesante mostrarlos.
  - Podemos definir niveles diferentes a según las partes de nuestro código que queramos con más o menos detalle.

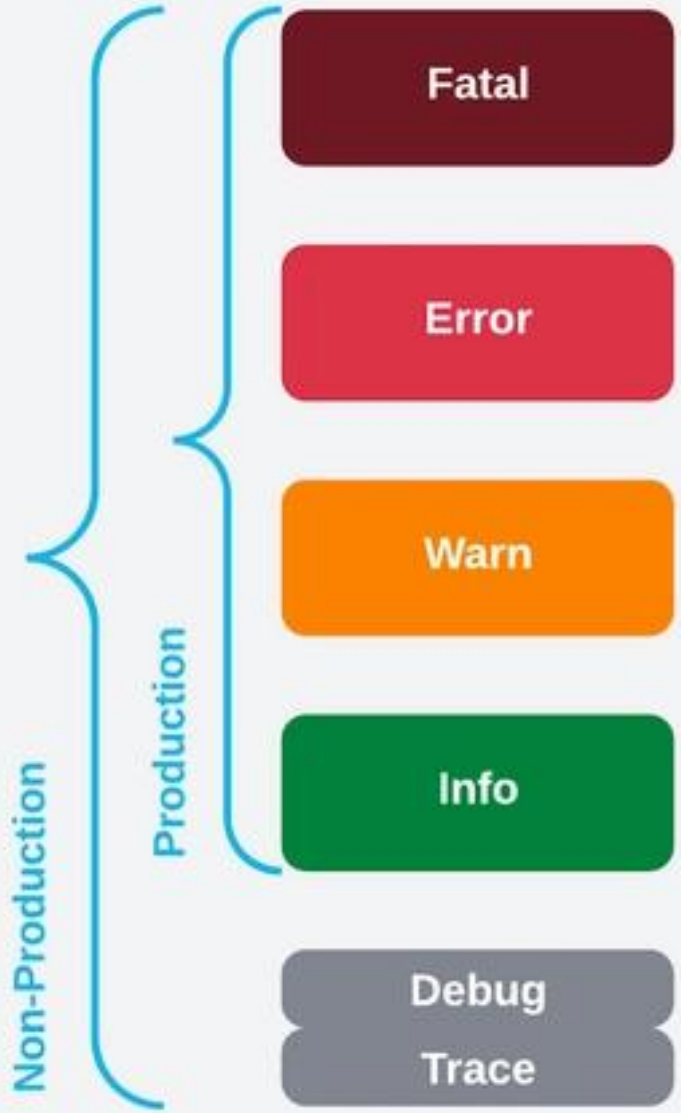
Logging – Niveles

- **TRACE**
  - Detalle muy profundo y minucioso de la ejecución.
  - Permite monitorizar todos los detalles del sistema.
- **DEBUG**
  - Información de depuración sobre la ejecución.
  - Normalmente introducido por los desarrolladores.
  - Mayor detalle del proceso seguido por el usuario.
- **INFO**
  - Mensajes con información general de la ejecución.
  - Información que se suele mostrar en producción.
  - Información para saber qué ha hecho un usuario en la aplicación.



Logging – Niveles

- WARN
  - Situaciones no críticas. Permiten al sistema continuar.
  - No deben enmascarar errores reales de ejecución.
- ERROR
  - Impide que la ejecución del usuario acabe satisfactoriamente
  - La aplicación no deja de funcionar, pero las acciones realizadas no se han podido ejecutar o no se han ejecutado de manera satisfactoria.
- FATAL
  - Fallo grave del sistema que impide que la aplicación siga ejecutándose.
  - Normalmente acaba con el cierre de la aplicación.



Logging – ¿Qué guardar?

- El log permite explotar de manera sencilla la información generada.
- Debemos definir un patrón homogéneo para todas las líneas de log.
- Se recomienda almacenar:
  - Fecha.
  - Identificador de sesión de usuario.
  - Fichero de código que contiene el log.
  - Tiempo de ejecución del método.
  - Etc.

```
20 nov 202 13:54:09,863 INFO MiCampusAuth:sessionCreated:126 - Usuario:
pedrody@um.es -1700434809863 - [96.249.89.158] - [Chrome-119.0.6045.123] -
                        [Android 6.0.1 - Nexus 5-ARM]
```

Logging – ¿Dónde guardar?

- Podemos volcar la información por diferentes vías.
  - Por la salida estándar.
  - A un fichero (es lo normal en cualquier aplicación).
  - A un servicio externo (email, mensaje de voz, …).
- El uso de ficheros y BBDD permite tener históricos de logs y rotarlos para evitar ficheros muy grandes.

El **rotado de log** es cuando un fichero donde se está escribiendo log se renombra automáticamente y se empieza a escribir el nuevo log en otro fichero

Configuración – Introducción

- Configurar el log en Java consta de dos pasos:
  1. Incluir las librerías de log necesarias.
  2. Configurar cómo se va a gestionar ese log.
- En asignaturas anteriores hemos visto cómo trabajar con Maven, por lo que gestionaremos las librerías en un proyecto Maven.

Configuración – Librerías

Librerías de log que podemos utilizar:

JUL (Java Util Logging)

- Viene por defecto en la JDK
- Configuración tediosa.
- Funcionalidad básica.

Logback

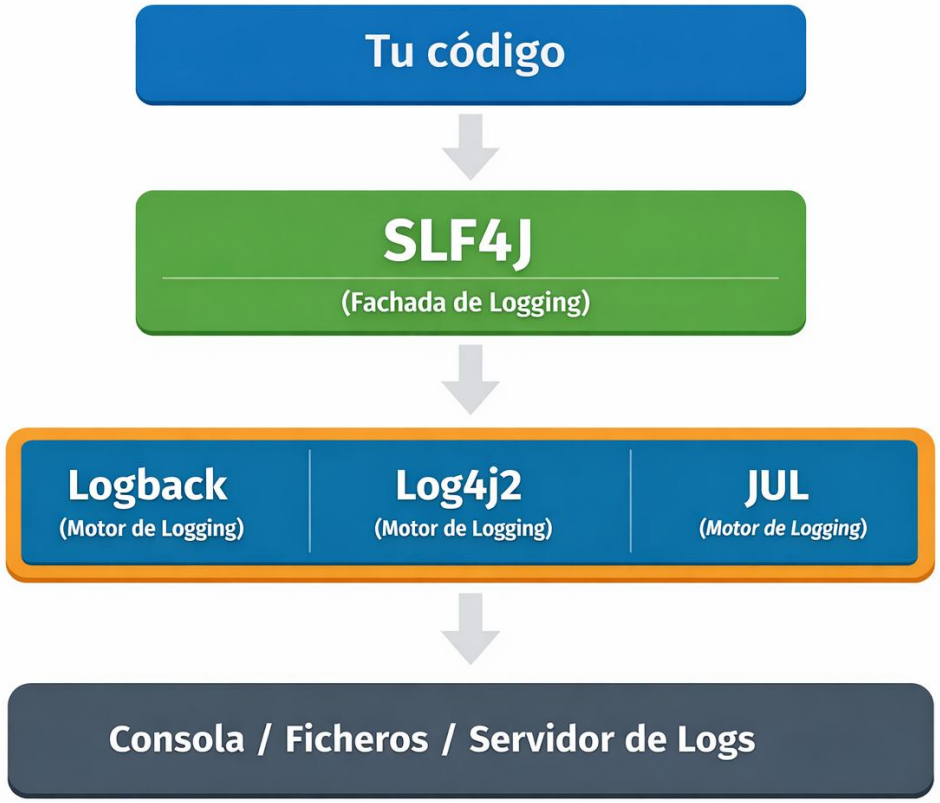
- Desarrollado por el creador de Log4j para corregir limitaciones de la versión 1.x
- Buen rendimiento.
- Configuración simplificada.

Log4j2

- Reescritura completa de log4j 1.x por otro equipo.
- Alto rendimiento.
- Logging asíncrono nativo.
- Muy configurable.

Configuración – Propuesta

- Nosotros usaremos **`log4j2`**, pero…
- Añadiremos SLF4J (Simple Logging Facade for Java)
  - Fachada de log que nos abstrae de la implementación.
  - Expone una interfaz que las diferentes implementaciones de logging cumplen.



Configuración – SLF4J

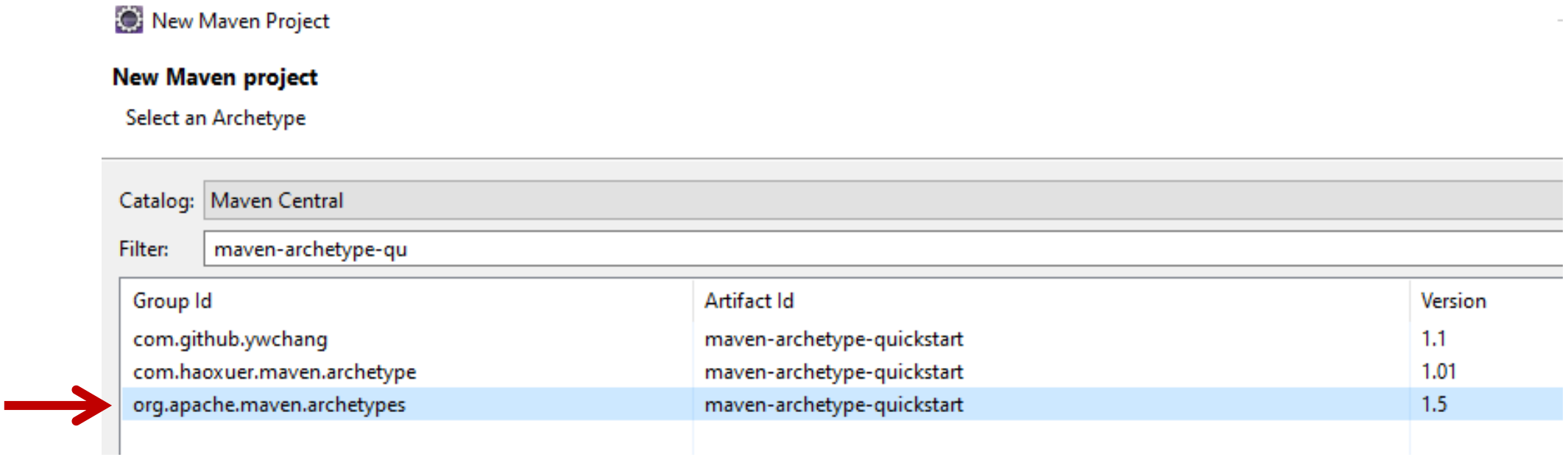
SLF4J (Simple Logging Facade for Java)

- Gracias a esta fachada podemos cambiar implementaciones sin que afecte nuestro código.
- Sí que deberemos cambiar los ficheros de configuración ya que cada librería tiene su formato.

Configuración – Creación del proyecto

1. Creación del proyecto

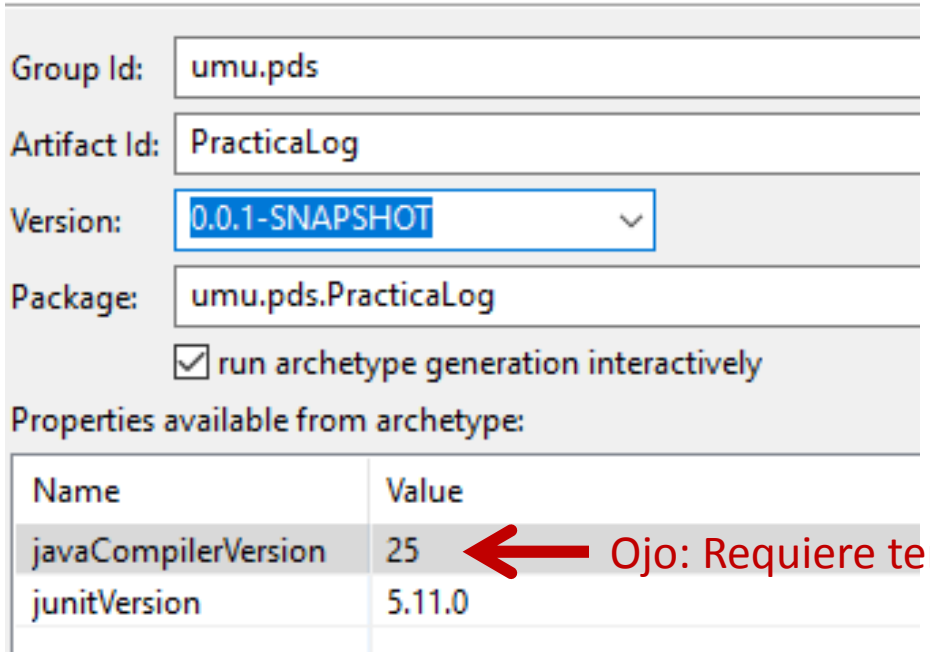
Crear proyecto Java a partir de un arquetipo Maven.



Configuración – Creación del proyecto

1. Creación del proyecto

Crear proyecto Java a partir de un arquetipo Maven.



Ojo: Requiere tener instalado JDK 25

Configuración – Dependencias

2. Añadir dependencias Maven:

- Log4j2
  https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core
- SLF4J
  https://mvnrepository.com/artifact/org.slf4j/slf4j-api
- Conectores Log4J2 y SLF4J
  https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j2-impl

```
<!-- Dependencias log-->
<!-- Log4j2 -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.25.3</version>
</dependency>

<!-- SLF4J -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>2.0.17</version>
</dependency>

<!-- Conector log4j y slf4j -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j2-impl</artifactId>
    <version>2.25.3</version>
    <scope>compile</scope>
</dependency>
<!-- Fin dependencias log-->
```

Configuración – Configuración de log4j

Log4j requiere un fichero de configuración.

- Dicho fichero obligatoriamente se llamará **`log4j2.xml`**
- Ruta en el proyecto: **`src/main/resources/log4j2.xml`**

Este fichero define:

- ¿Qué contenido se escribe en el log?
- ¿Cómo se escribe dicho contenido?
- ¿Dónde se escriben los logs?

El fichero se divide en dos secciones:

- **Appenders**. Escritores de log.
- **Loggers**. Reglas de qué se escribe.

```
<Configuration>
    <Appenders>...</Appenders>
    <Loggers>...</Loggers>
</Configuration>
```

Configuración – log4j – Appenders

Diferentes tipos

`Console, File, Database, Network, …`

- Podemos tener todos los que queramos a la vez.

```
<Appenders>
    <!-- Appender de consola, se escribira por la consola -->
    <Console name="Console">
        <PatternLayout pattern="[%d{HH:mm:ss}] [%level] %logger - %msg%n" />
    </Console>

    <!-- Apender de fichero, esta configuracion es obligatoria para entornos de produccion -->
    <File name="FileLogger" fileName="logs/app.log" append="true">
        <PatternLayout pattern="[%d{yyyy-MM-dd HH:mm:ss}] [%level] %logger - %msg%n" />
    </File>

</Appenders>
```

Configuración – log4j – Appenders – File Appender

Para almacenar en ficheros.

- **fileName**
  Ruta donde estará el fichero de log. La crea si no existe.
- **Append**
  Añadir cada línea de log al final, en caso contrario sobrescribe.

```
<!-- Apender de fichero, esta configuracion es obligatoria para entornos de produccion -->
<File name="FileLogger" fileName="logs/app.log" append="true">
    <PatternLayout  pattern="[%d{yyyy-MM-dd HH:mm:ss}] [%-5level] %logger - %msg%n" />
</File>
```

Problema:
El log no deja de crecer.

Solución:
Rotado de logs -> RollingFile

Configuración – log4j – File Appender

`RollingFile`:
Establece las políticas (**`Policies`**) indican cómo se rota.

`filePattern`:
Indica el formato que se le dará al nombre del fichero cada vez que rote.

```
<RollingFile name="RollingFile"
    fileName="logs/app.log"
    filePattern="logs/app-%d{yyyy-MM-dd}-%i.log.gz">
    <PatternLayout
        pattern="[%d{yyyy-MM-dd HH:mm:ss}] [%level] %logger - %msg%n" />
    <Policies>
        <!-- Rota cada 10 MB -->
        <SizeBasedTriggeringPolicy size="10 MB" />
        <!-- Rota cada día -->
        <TimeBasedTriggeringPolicy />
    </Policies>
</RollingFile>
```

Configuración – log4j – Appenders – PatternLayout

PatternLayout define qué información se imprime en cada línea de log

La lista completa de posibilidades puedes consultarla en el manual de log4j

```
<PatternLayout  pattern="[%d{yyyy-MM-dd HH:mm:ss}] [%level] %logger - %msg%n" />
```

Token	Descripción	Ejemplo
%d{yyyy-MM-dd HH:mm:ss}	Fecha y hora completa: año-mes-día horas:minutos:segundos	2026-01-08 14:45:30
%level	Nivel del log (INFO, DEBUG, WARN, ERROR…)	INFO, ERROR
%logger	Nombre del logger (normalmente la clase)	com.miapp.Main
%msg	Mensaje que envías en el código (log.info("…"))	Aplicación iniciada
%n	Salto de línea	Nueva línea al final

Configuración – log4j – Loggers

- Define qué ***niveles*** de log se utilizan de manera global o por paquetes.
- Define qué ***appenders*** voy a usar y con qué configuración.
- Se debe definir un ***logger Root*** que será el general de la aplicación.
- Podemos definir ***loggers adicionales*** para precisar qué reglas aplican a qué partes de nuestro código.

```
<Loggers>
    <!-- Log general -->
    <!-- Damos de alta los appenders que queremos usar -->
    <Root level="DEBUG">
        <!-- Appender de consola -->
        <AppenderRef ref="Console" />
        <!-- Appender de fichero -->
        <AppenderRef ref="FileLogger" />
    </Root>

    <!-- Log sobre "umu.pds.trace" -->
    <Logger name="umu.pds.trace" level="TRACE" additivity="false">
        <!-- Appender de consola -->
        <AppenderRef ref="Console" />
    </Logger>
</Loggers>
```

Configuración – log4j – Loggers

- De manera general se imprimirán los logs de tipo `Debug` o mayor prioridad.
- Para "`umu.pds.trace`" se imprimirán los logs de tipo `Trace` o mayor prioridad.

```
<Loggers>
    <!-- Log general -->
    <!-- Damos de alta los appenders que queremos usar -->
    <Root level="DEBUG">
        <!-- Appender de consola -->
        <AppenderRef ref="Console" />
        <!-- Appender de fichero -->
        <AppenderRef ref="FileLogger" />
    </Root>

    <!-- Log sobre "umu.pds.trace" -->
    <Logger name="umu.pds.trace" level="TRACE" additivity="false">
        <!-- Appender de consola -->
        <AppenderRef ref="Console" />
    </Logger>
</Loggers>
```

Configuración – log4j – Loggers

Obtención de una instancia de log

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LogDebugClass {

    private static final Logger log = LoggerFactory.getLogger(LogDebugClass.class);

    public static void imprimeDebug() {
        log.debug("Imprimo mensaje de debug");
        log.trace("Imprimo trace desde el metodo de Debug");
        log.error("Error con codigo ",new Exception("Excepcion de error"));
    }
```

slf4j: Fachada para uso de log independiente de la implementación

Factoría de log

Clase que usará esta instancia de log

Uso de log

Configuración – Ejercicios

Ejercicio 1

Configurar log4j2 en un proyecto y definir una estructura de clases que:

- Imprima log de nivel "`INFO`" para el paquete `umu.pds.info`
- Imprima log de nivel "`DEBUG`" para el paquete `umu.pds.debug`
- No imprima log de nivel "`TRACE`" en ningún caso.
- Se deben invocar a los métodos que impriman mensajes de debug de todos los niveles mencionados anteriormente independientemente de que el log lo imprima o no.

Configuración – Ejercicios

Ejercicio 2

Escribir mensajes de log concatenando valores es una mala práctica:

```
log.debug ("variable1: “ + variable1 + "; variable 2: “ +  variable2);
```

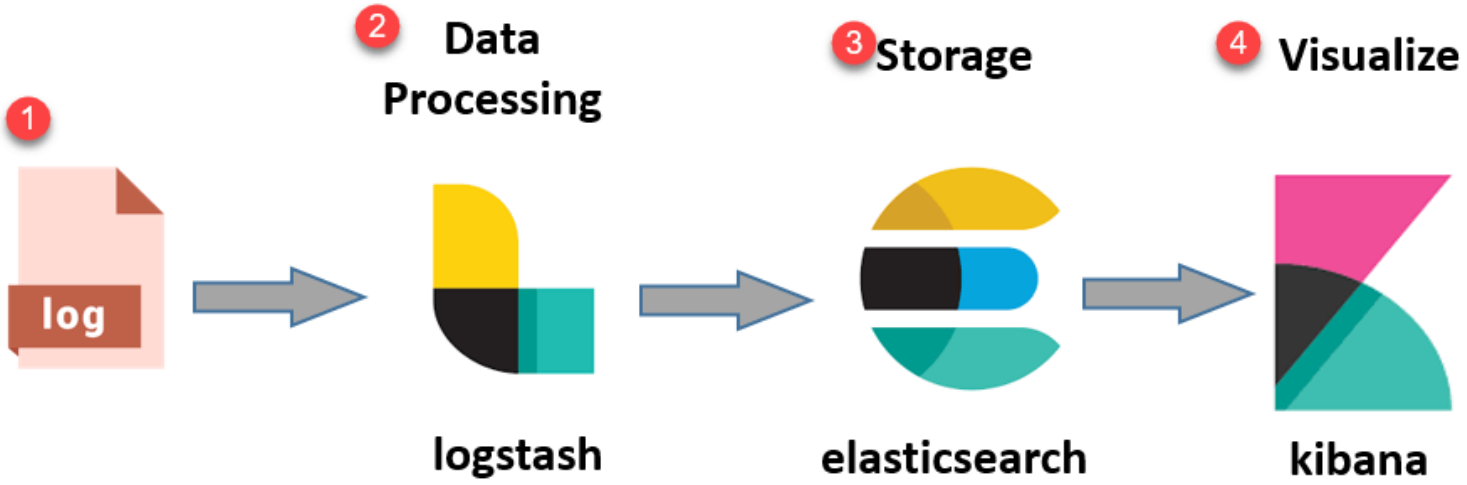
Investigar de qué manera es posible especificar el valor de las variables sin hacer concatenaciones.

Manual

https://logging.apache.org/log4j/2.12.x/manual

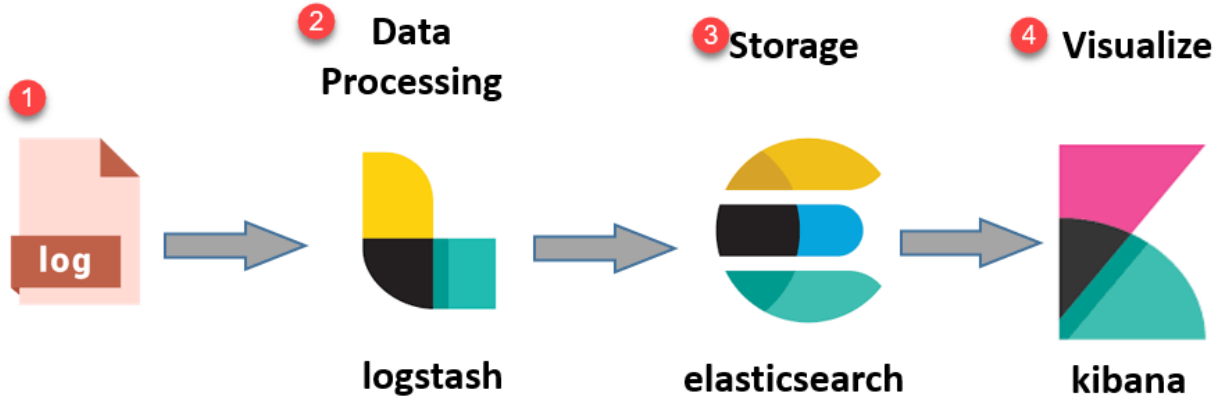
Explotación del log – Pipeline

En aplicaciones con un uso masivo, como por ejemplo aplicaciones o servicios web, hay herramientas especializadas para explotar logs.



Pila ELK: **E**lasticSearch **L**ogstash **K**ibana

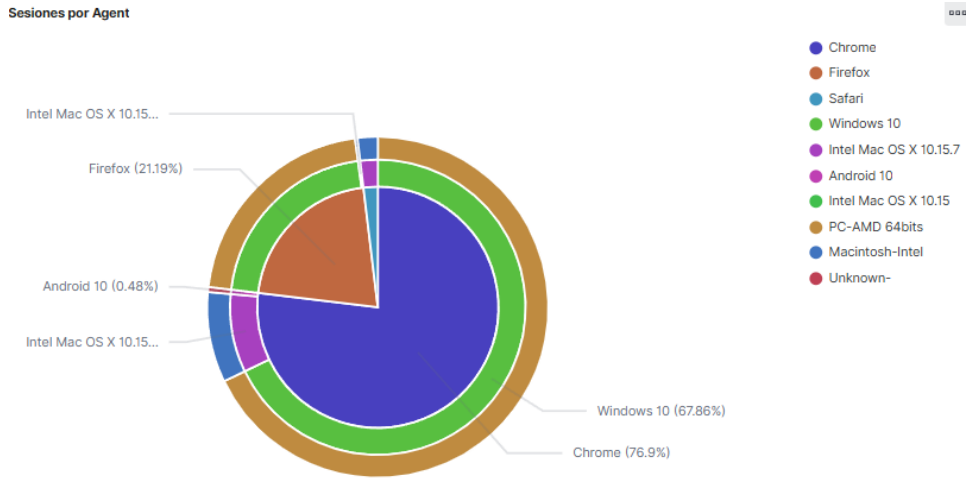
Explotación del log – Pipeline

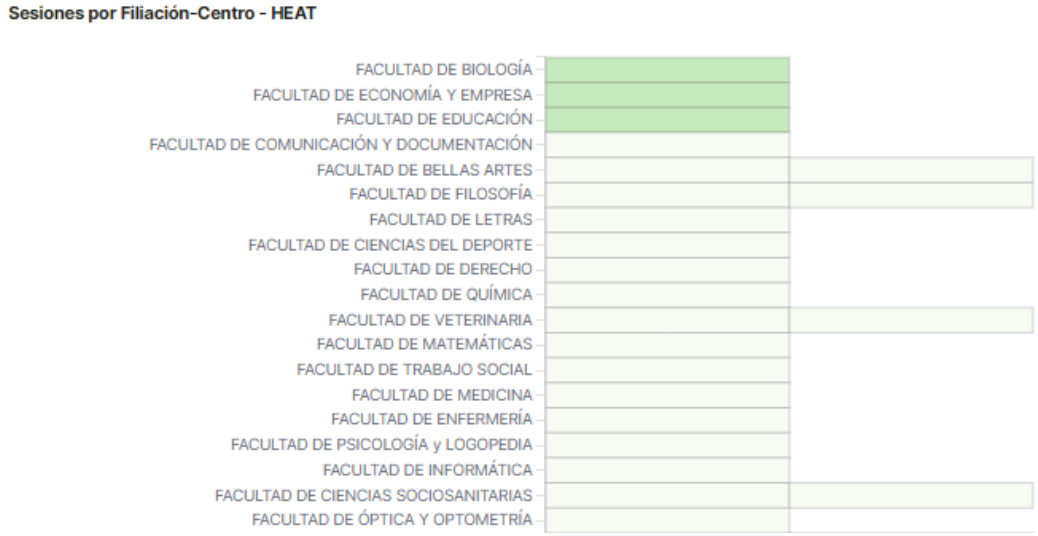


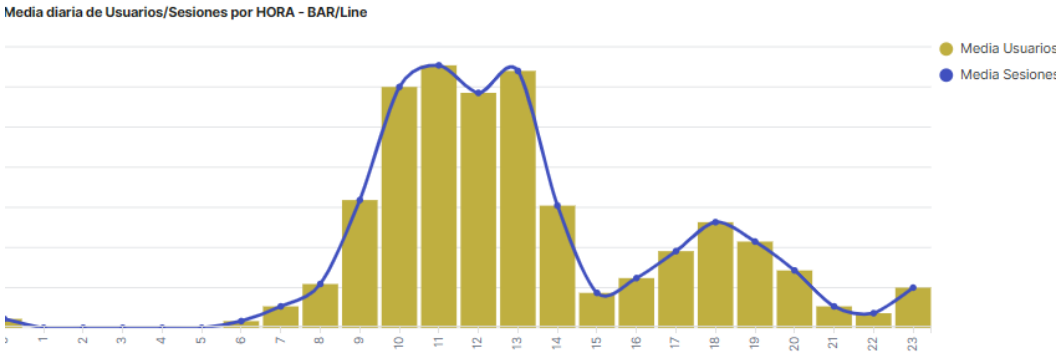
- Logstash
  A partir de la línea de log en bruto, identifica los fragmentos y los separa.
- ElasticSearch
  Almacena para cada registro índices para facilitar las búsquedas.
- Kibana
  Visualiza la información almacenada por ElasticSearch.

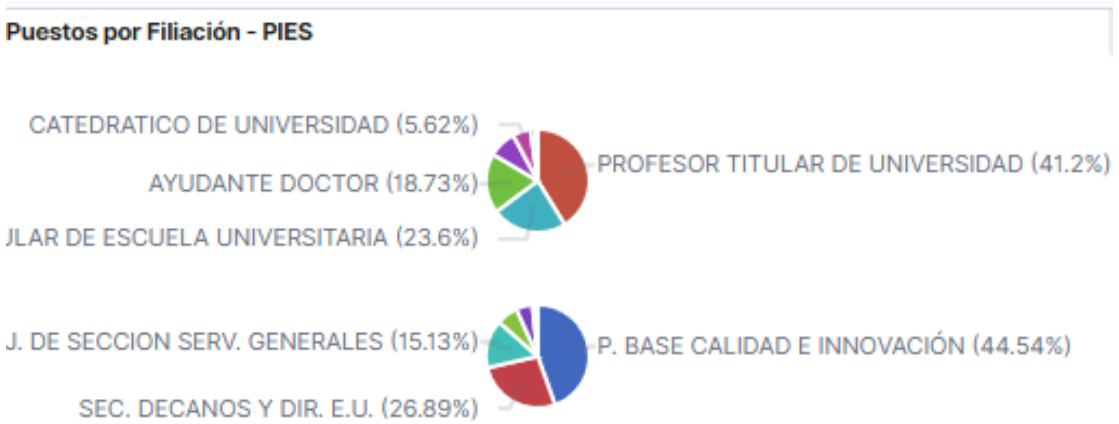
Explotación del log – Pipeline

Con Kibana podemos, parte de ver los logs, crear visualizaciones









Explotación del log – Buenas prácticas

- La pila ELK es sólo un ejemplo. Existen diferentes sistemas para tratar la información que genera una aplicación.
  - No obstante, todas requieren registrar logs y el sistema no usa la salida estándar.
- Nosotros somos los responsables de nuestros logs.
  Guardar información:
  - Útil.
  - Que aporte valor.
  - En su justa medida.
  - Que no comprometa la seguridad el sistema o la información de los usuarios.

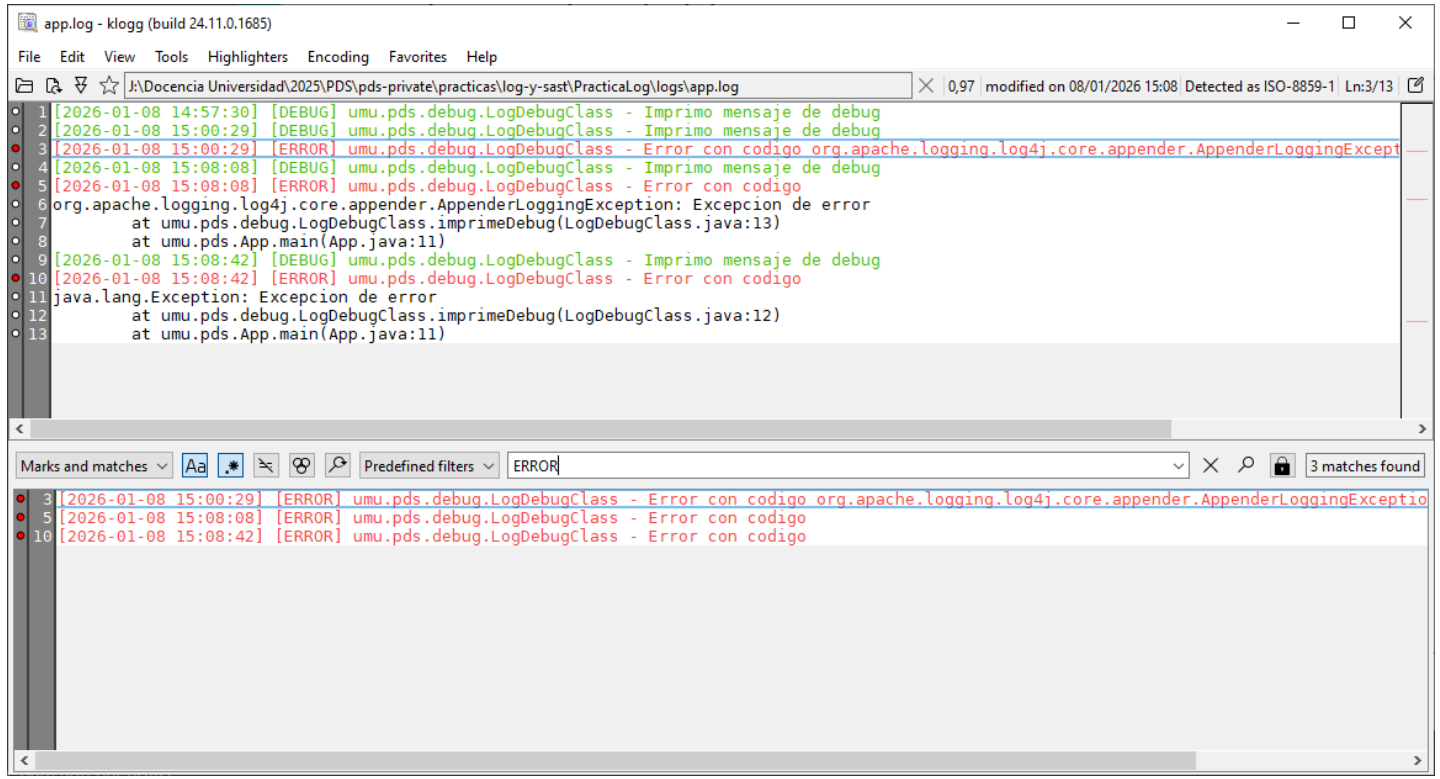
Explotación del log – Pipeline

- Normalmente el log se explota mediante la pila ELK (ElasticSearch, Logstash y Kibana) o similares.
- Existen muchas aplicaciones de escritorio que ayudan a la interpretación y explotación del log.
- Para esta práctica usaremos **Klogg**:
  - Tiene una versión portable para Windows
  - Versiones para Linux
  - Versiones para Mac

(No obstante, otras muchas aplicaciones tienen funcionalidades similares)

Explotación del log – Klogg

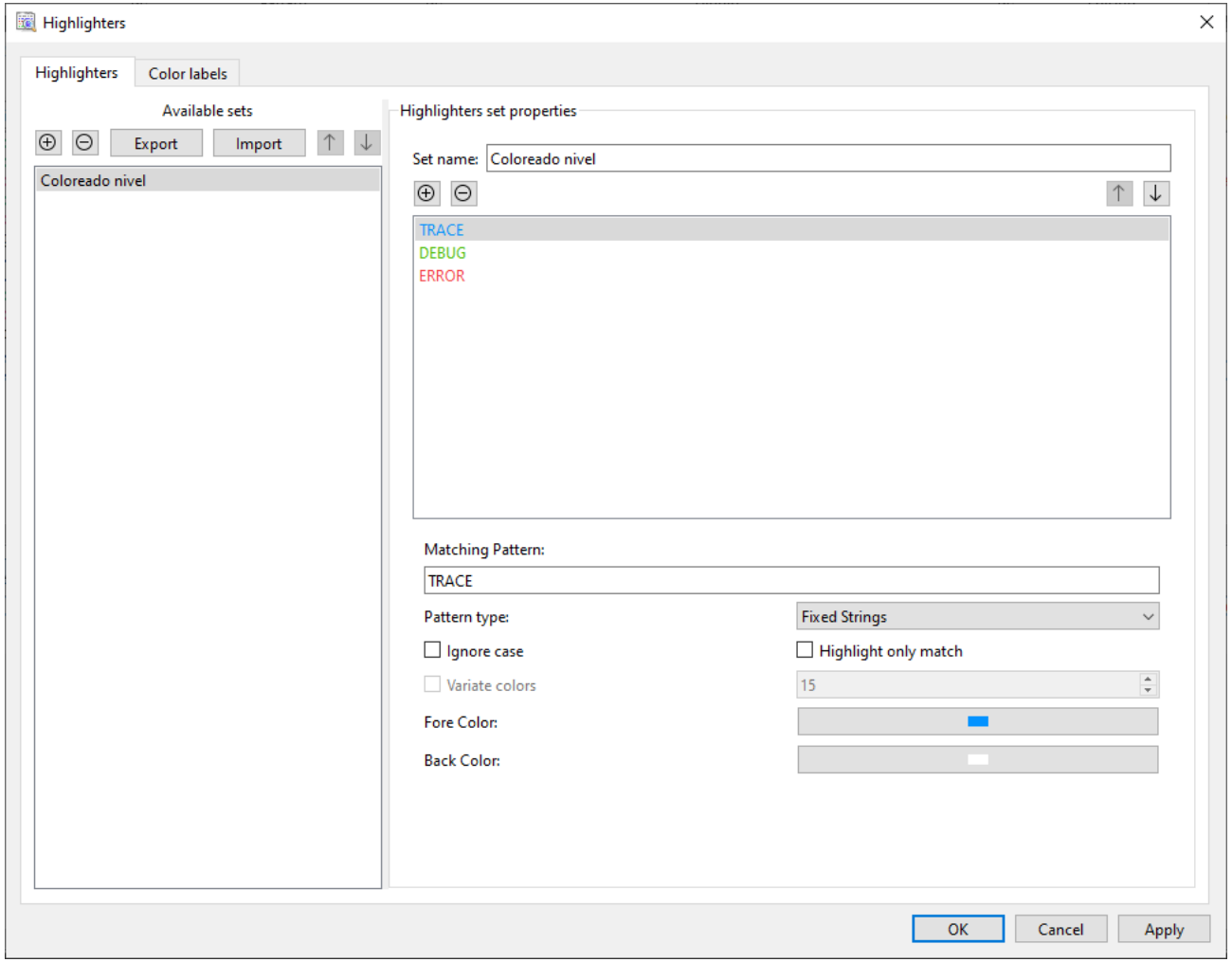
- Permite abrir diferentes ficheros de log a la vez
- Ideal para ficheros de log muy extensos
- Monitoriza en tiempo real.
- Soporte para:
  - Expresiones regulares.
  - Coloreado.
  - Filtrado.



Explotación del log – Klogg – Patrones de coloreado

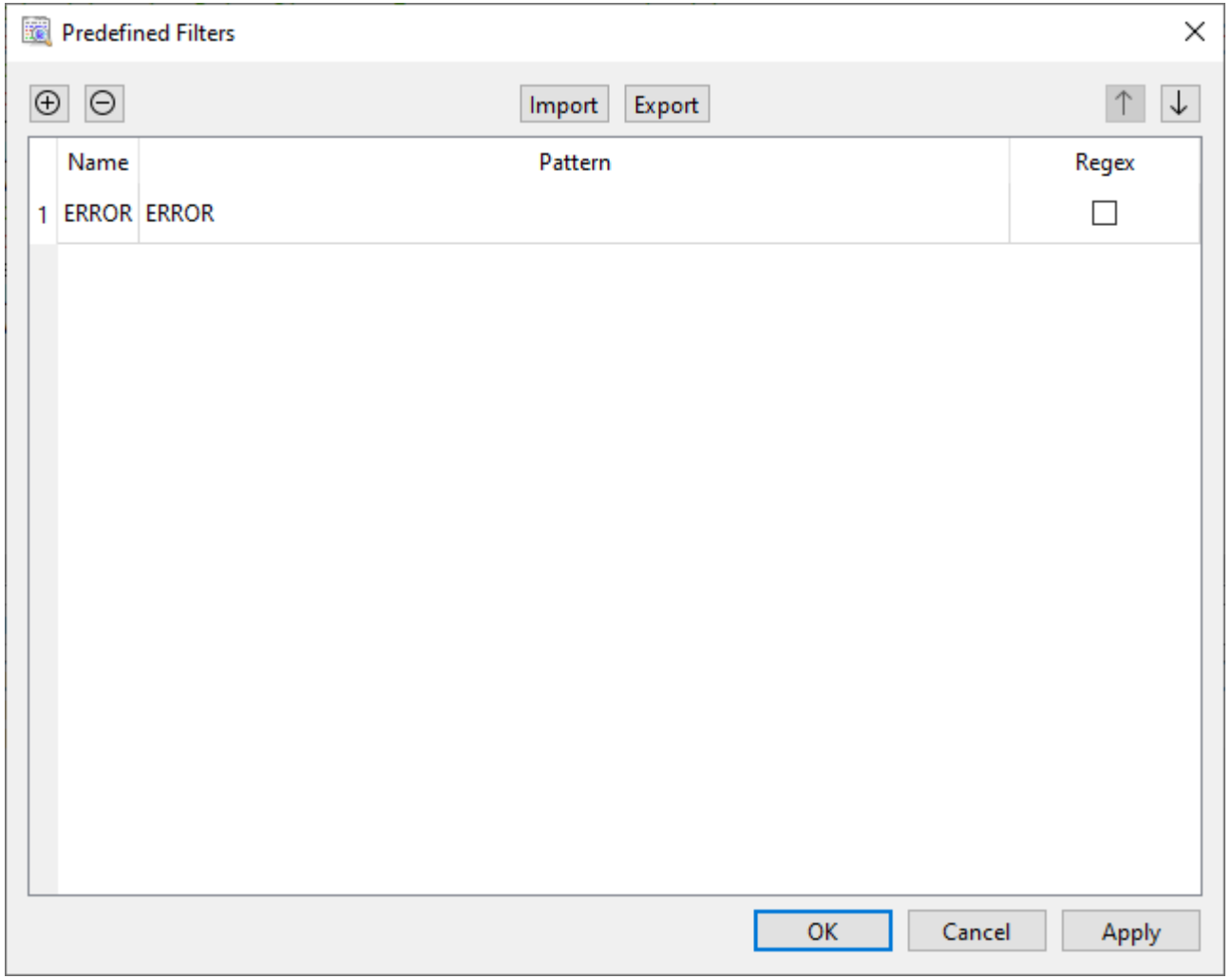
Establecer un patrón de coloreado.

- Texto directo o Expresiones Regulares.
- Múltiples sets de pintado.



Explotación del log – Klogg – Filtros predefinidos

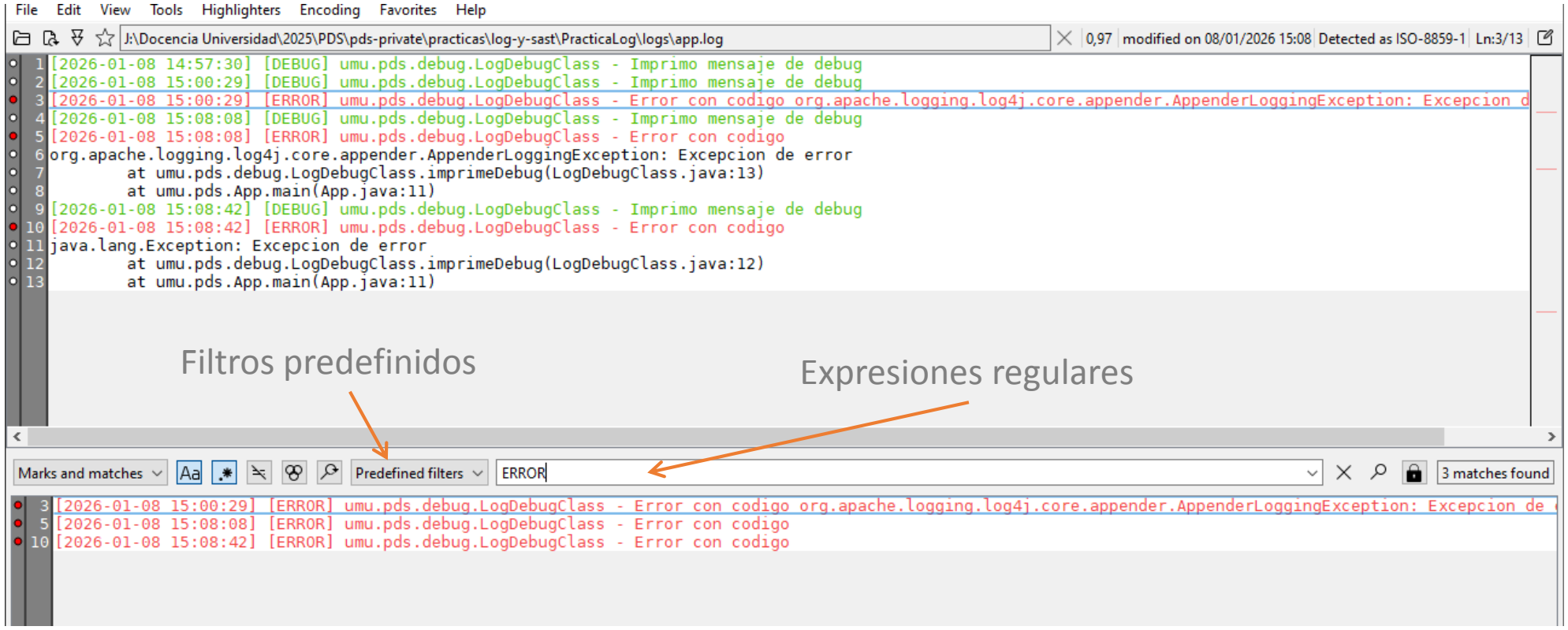
- Texto directo o Expresiones Regulares.
- Múltiples filtros.



Explotación del log – Klogg – Área de trabajo

Log en bruto

Área de filtrado



Explotación del log – Klogg – Resumen

- Herramienta sencilla y ágil.
- Gran potencia a la hora de trabajar en local con ficheros de log.
- El motor de expresiones regulares permite precisar mucho los resultados.
- Fácil instalación.

Referencias

- Apache Log4j2
  https://logging.apache.org/log4j/2.12.x/
- Log4j2 manual
  https://logging.apache.org/log4j/2.x/manual/index.html
- SLF4J: Simple Log Facade for Java
  https://www.slf4j.org/manual.html
- Klogg
  https://klogg.filimonov.dev/

PARTE II
SAST y calidad

Static Application Security Testing

SAST en desarrollos software – Motivación

Con **análisis estático de código** podemos prevenir:

- Problemas de programación que pueden dar lugar a errores.
- Vulnerabilidades en nuestro código.
- Malas prácticas de programación.
- Cobertura de los test.

- Diferentes herramientas:
  - SonarQube
  - Gitlab SAST
  - Github CodeQL
  - …

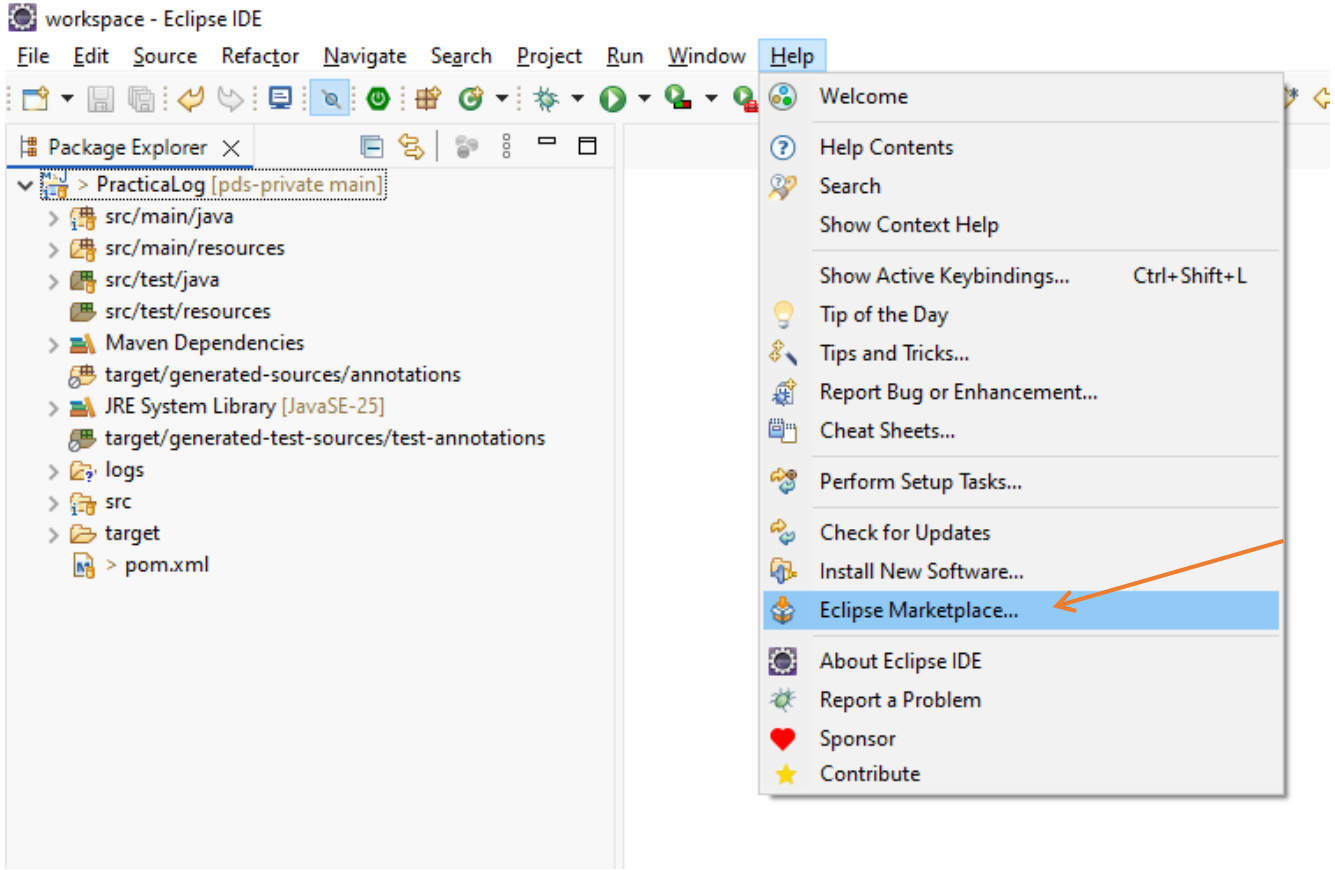
SAST en desarrollos software – SonarQube

En PDS vamos a usar **SonarQube** porque:

- Permite análisis en local ya que se integra con:
  - Eclipse
  - Visual Studio Code
  - IntelliJ
- Disponemos de un servidor SonarQube para la asignatura
- Puede integrarse con Maven en cualquier ciclo CI/CD

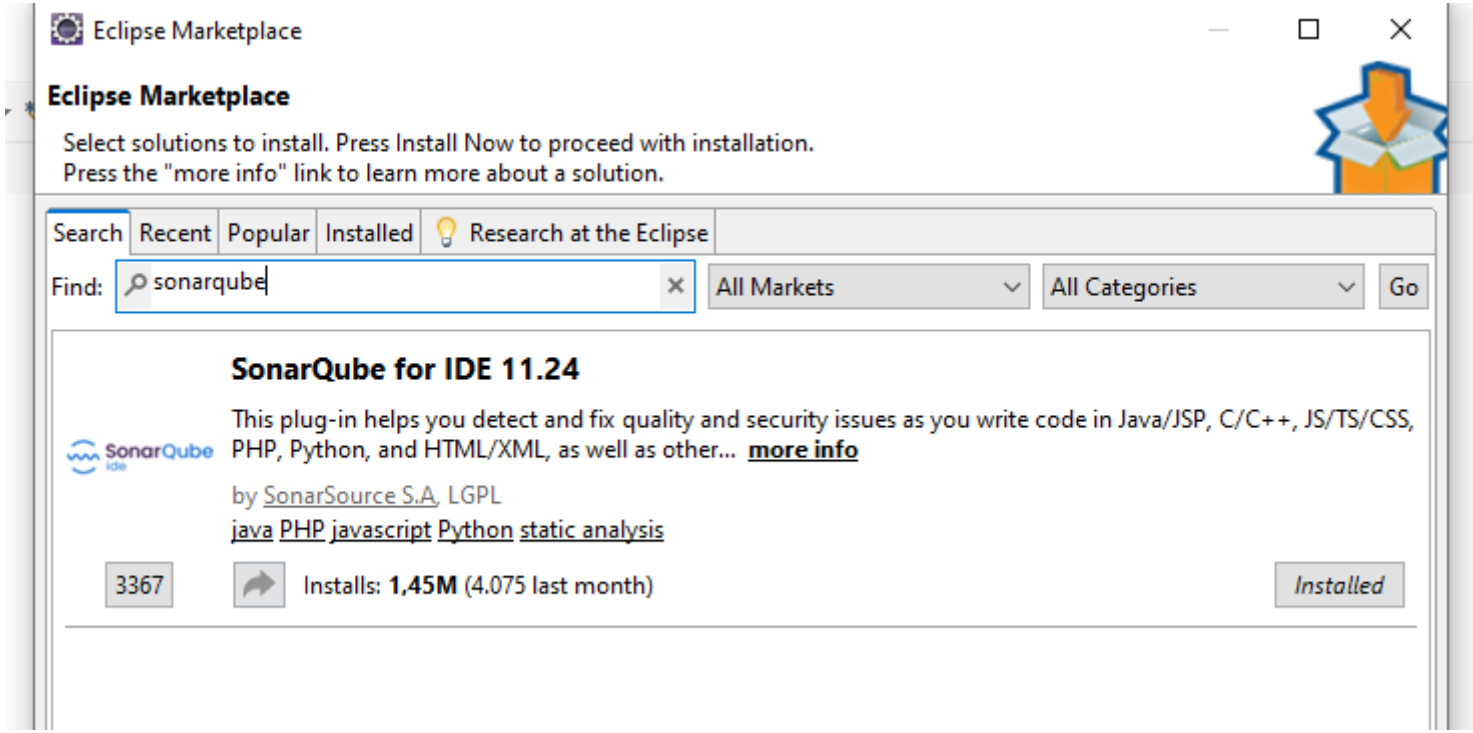
SAST – SonarQube – Configuración

- Instalación en eclipse:
  Accedemos al Marketplace



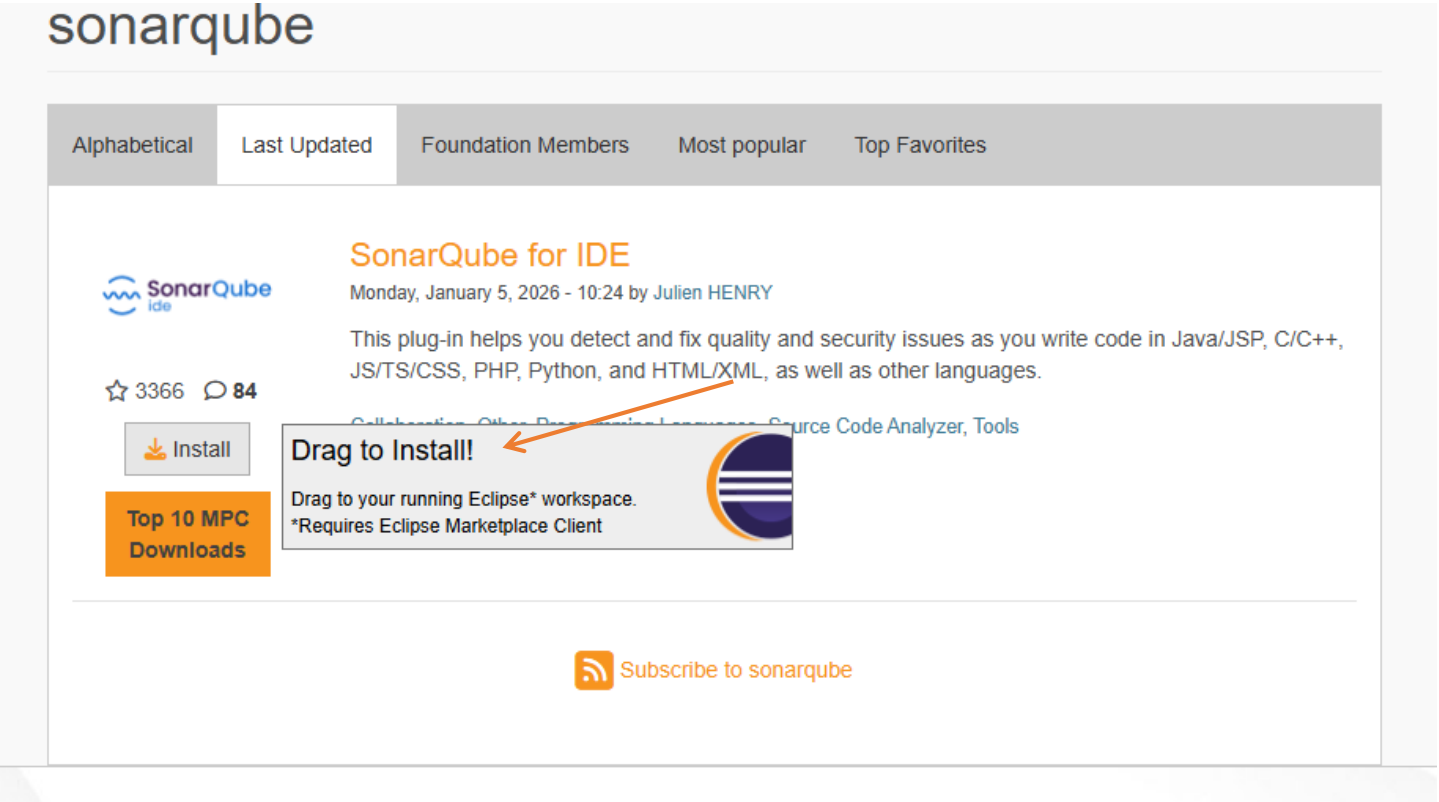
SAST – SonarQube – Configuración

- Instalación en eclipse:
  Buscamos "sonarqube" e instalamos el plugin.



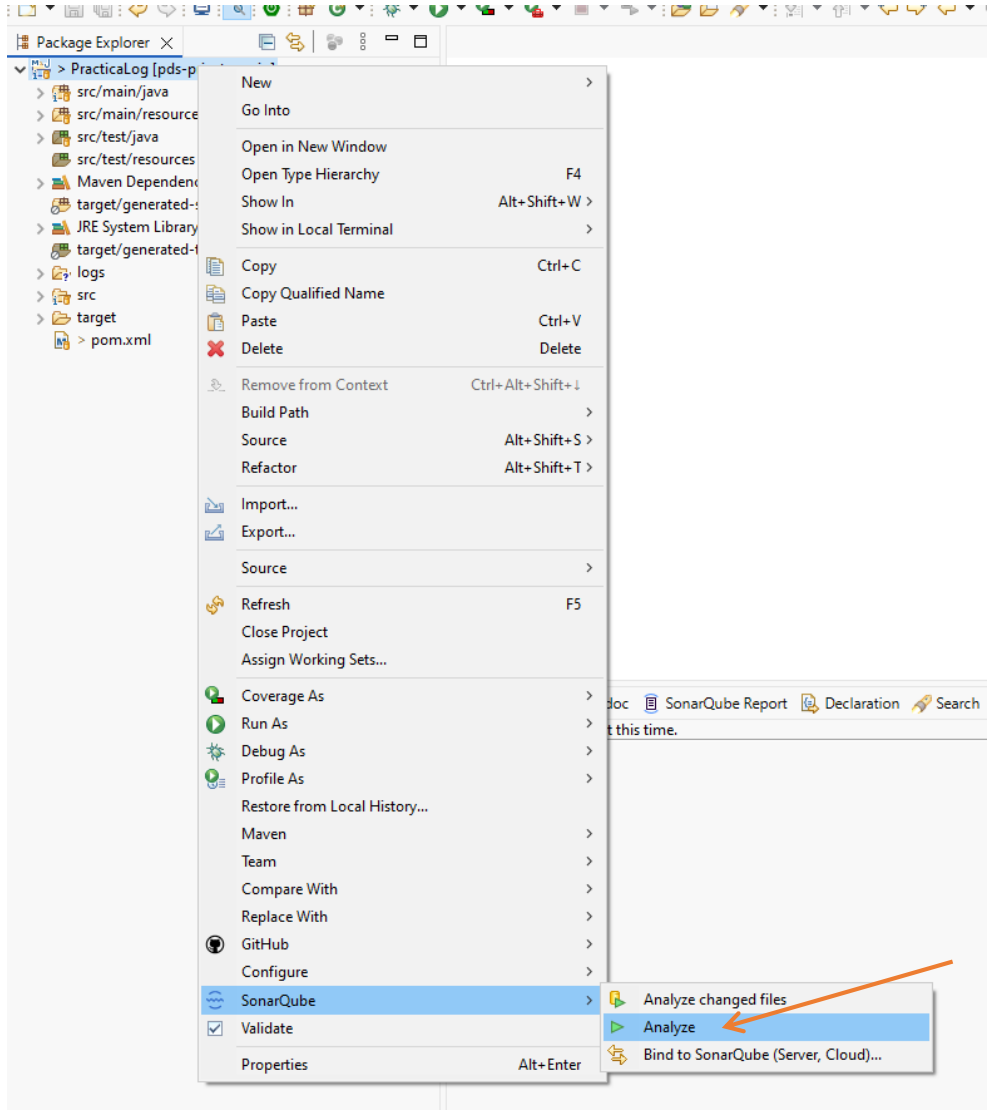
SAST – SonarQube – Configuración

- Instalación en eclipse:
  Instalación alternativa por si no aparece en el market
  Accedemos a: https://marketplace.eclipse.org/free-tagging/sonarqube



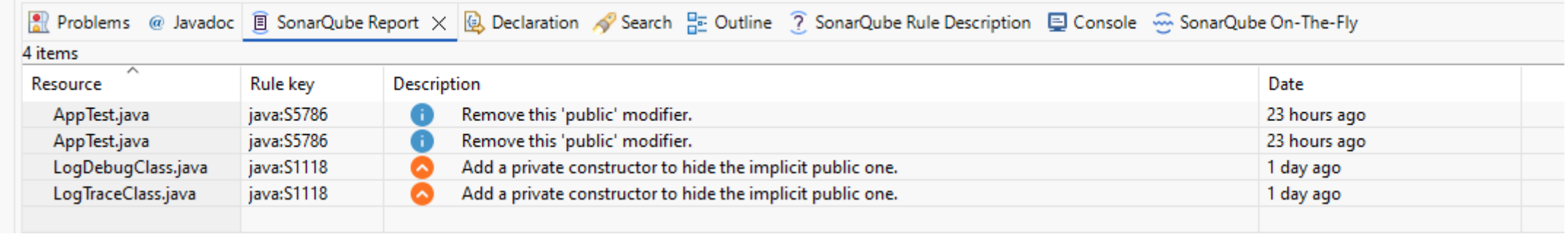
SAST – SonarQube – Uso

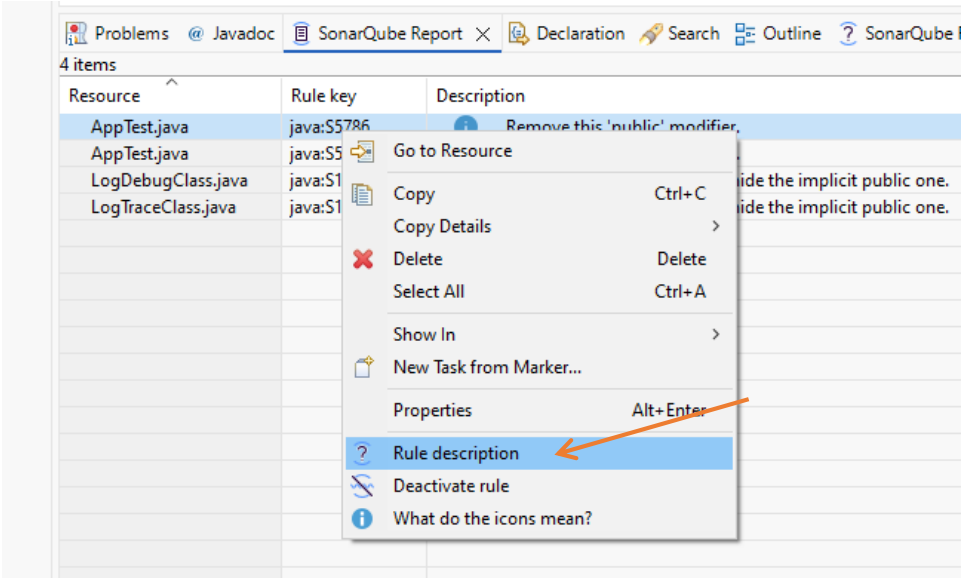
- Una vez instalado podremos escanear los proyectos en el menú contextual.
- Deberemos hacerlo sobre la raíz del proyecto para que lo escanee todo.
- Una vez acabe mostrará los resultados.



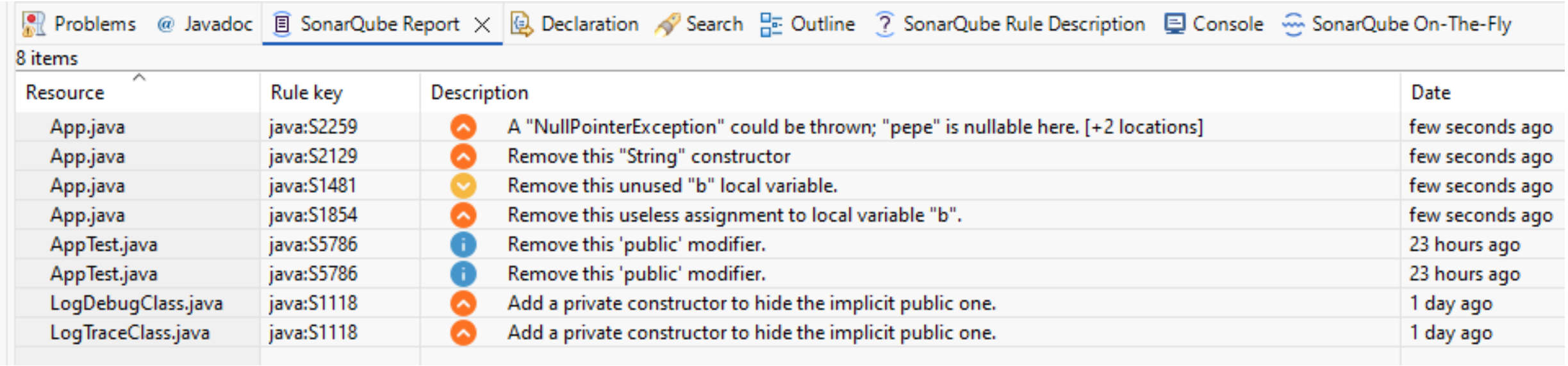
SAST – SonarQube – Uso

Nos listará los resultados y podremos consultar la regla que aplica.





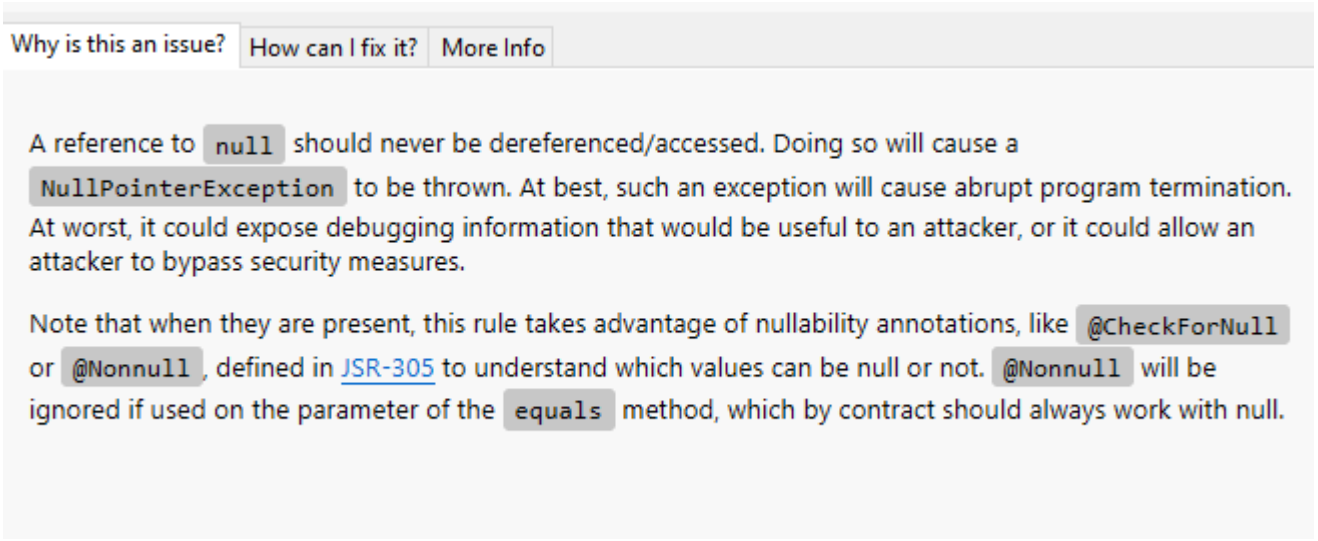
SAST – SonarQube – Uso



Resource	Rule key	Description	Date
App.java	java:S2259	A "NullPointerException" could be thrown; "pepe" is nullable here. [+2 locations]	few seconds ago
App.java	java:S2129	Remove this "String" constructor	few seconds ago
App.java	java:S1481	Remove this unused "b" local variable.	few seconds ago
App.java	java:S1854	Remove this useless assignment to local variable "b".	few seconds ago
AppTest.java	java:S5786	Remove this 'public' modifier.	23 hours ago
AppTest.java	java:S5786	Remove this 'public' modifier.	23 hours ago
LogDebugClass.java	java:S1118	Add a private constructor to hide the implicit public one.	1 day ago
LogTraceClass.java	java:S1118	Add a private constructor to hide the implicit public one.	1 day ago

SAST – SonarQube – Uso

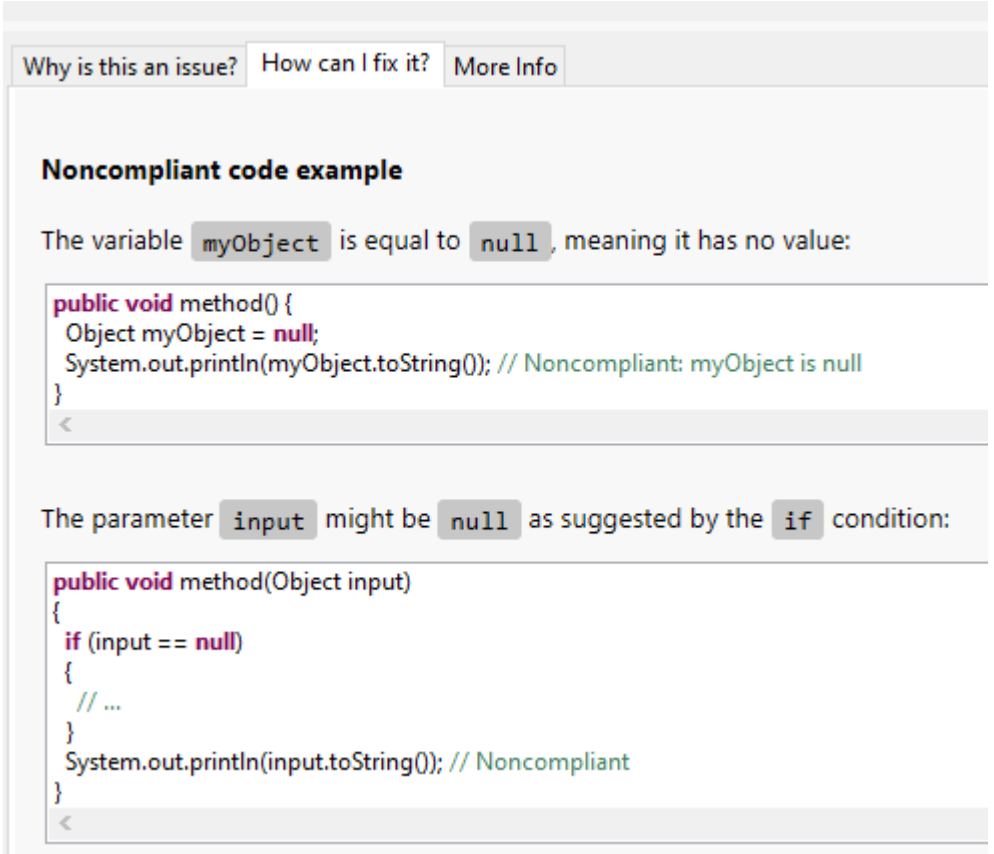
Podemos conocer por qué salta la regla y cómo corregirlo.



Why is this an issue? How can I fix it? More Info

A reference to `null` should never be dereferenced/accessed. Doing so will cause a `NullPointerException` to be thrown. At best, such an exception will cause abrupt program termination. At worst, it could expose debugging information that would be useful to an attacker, or it could allow an attacker to bypass security measures.

Note that when they are present, this rule takes advantage of nullability annotations, like `@CheckForNull` or `@Nonnull`, defined in JSR-305 to understand which values can be null or not. `@Nonnull` will be ignored if used on the parameter of the `equals` method, which by contract should always work with null.



Why is this an issue? How can I fix it? More Info

Noncompliant code example

The variable `myObject` is equal to `null`, meaning it has no value:

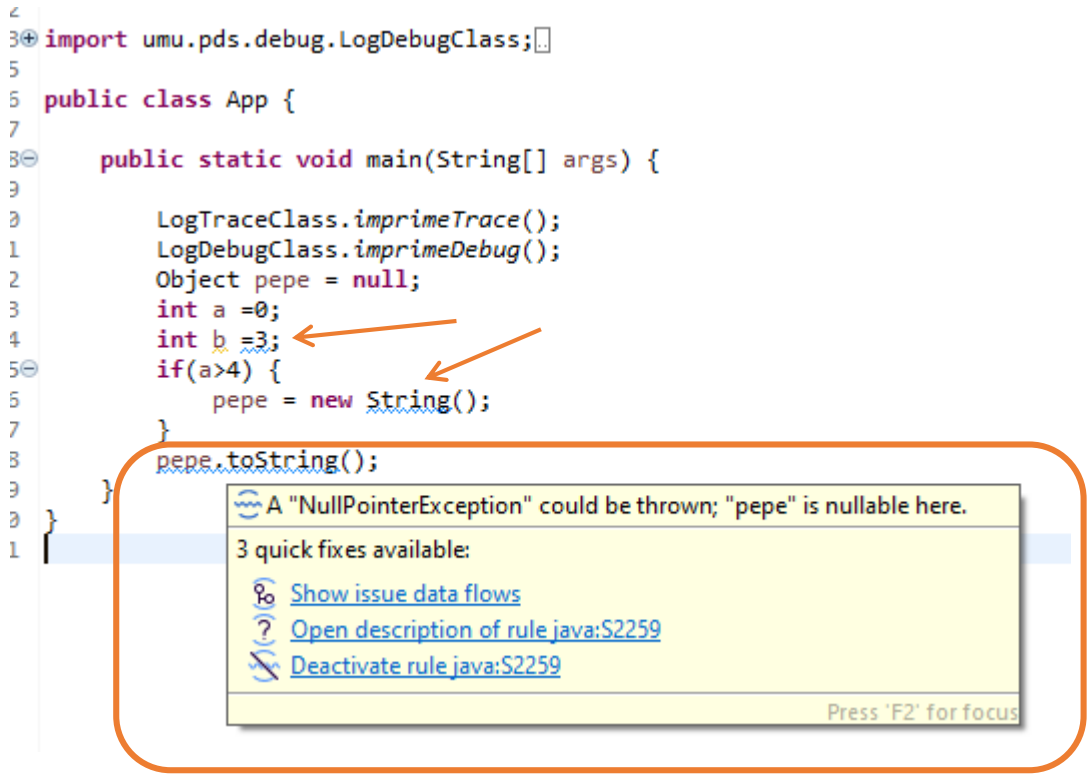
```
public void method() {
  Object myObject = null;
  System.out.println(myObject.toString()); // Noncompliant: myObject is null
}
```

The parameter `input` might be `null` as suggested by the `if` condition:

```
public void method(Object input)
{
  if (input == null)
  {
    // ...
  }
  System.out.println(input.toString()); // Noncompliant
}
```

SAST – SonarQube – Uso

SonarQube también subrayará en azul, sobre nuestro código, los problemas potenciales que detecte



SAST – SonarQube – Integración

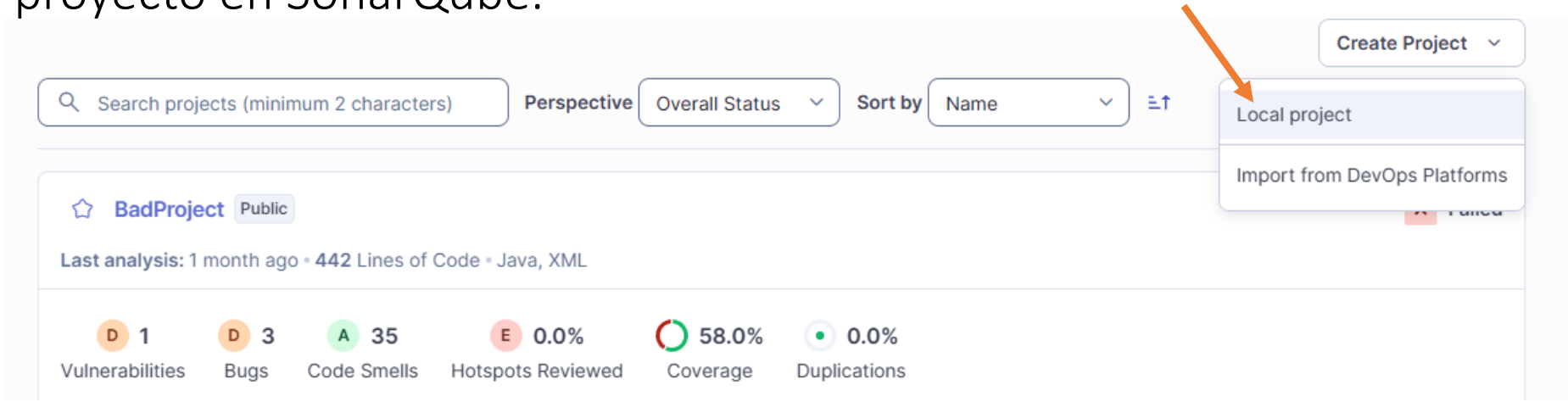
- Aunque el plugin de SonarQube es muy útil, en entornos de trabajo reales se limita a los entornos locales.
- Es necesario integrarlo en un servidor donde se centralice todos los desarrollos de la empresa o grupo de trabajo.
- Usado las herramientas CI/CD de las que se dispongan, podremos integrar el análisis de código dentro del ciclo de desarrollo de cualquier equipo.

SAST – SonarQube – Integración

- Vamos a analizar nuestro proyecto de dos maneras diferentes.
  - Enviándolo desde local al servidor SonarQube.
  - Añadiéndolo como una Github Action (CI/CD).
- Debemos subir nuestro proyecto a nuestro repositorio Github para poder hacer el segundo paso.

SAST – SonarQube – Integración

- El primer paso es configurar nuestro proyecto para que se analice en https://sast.inf.um.es
- Una vez hecho login con nuestra cuenta de la Universidad de Murcia creamos nuestro proyecto en SonarQube.

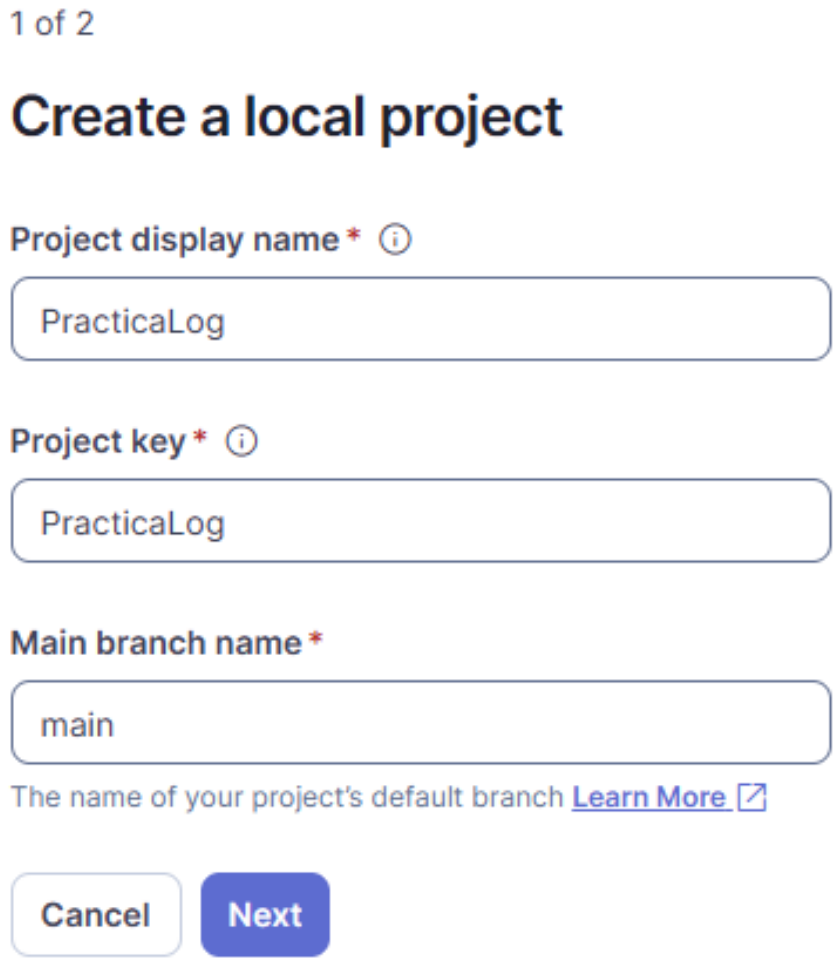


- Lo creamos como local porque no vamos a hacer un proceso de importación desde github.

SAST – SonarQube – Integración

Dentro del menú que nos aparece le tendremos que dar un nombre.

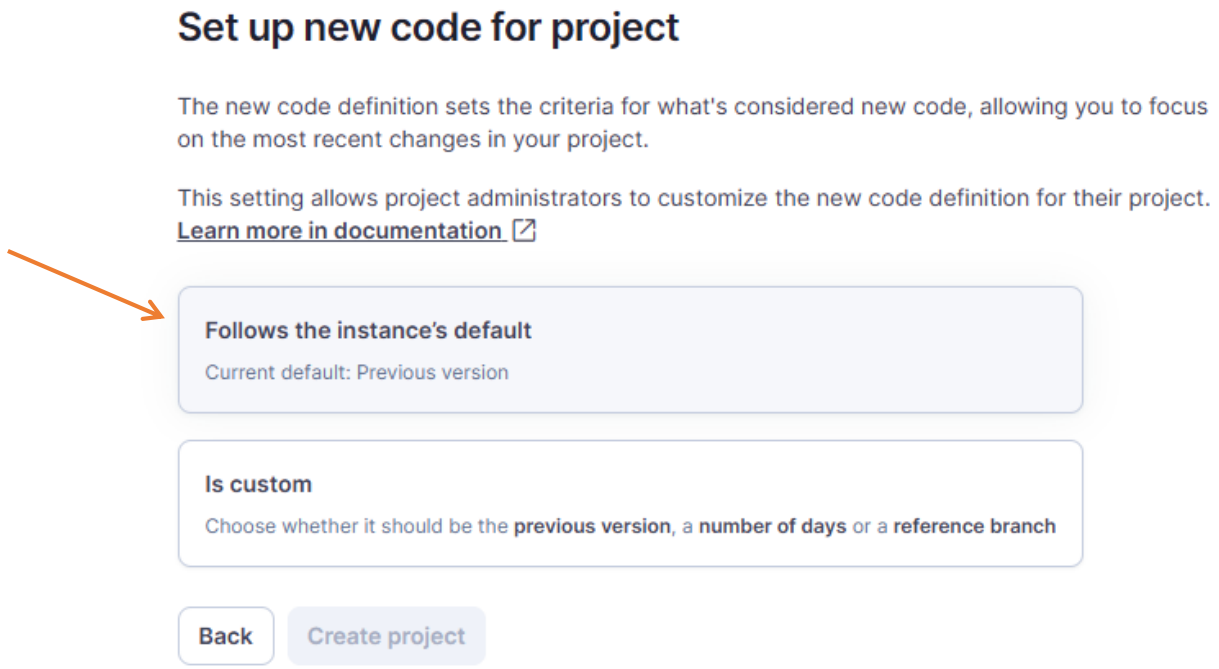
- Deberemos tener nombres de proyectos diferentes para que no colisionen
- Dejamos la rama principal con el valor por defecto "**`main`**"



SAST – SonarQube – Integración

Una vez hemos creado el proyecto, deberemos establecer el criterio para decidir qué es código nuevo

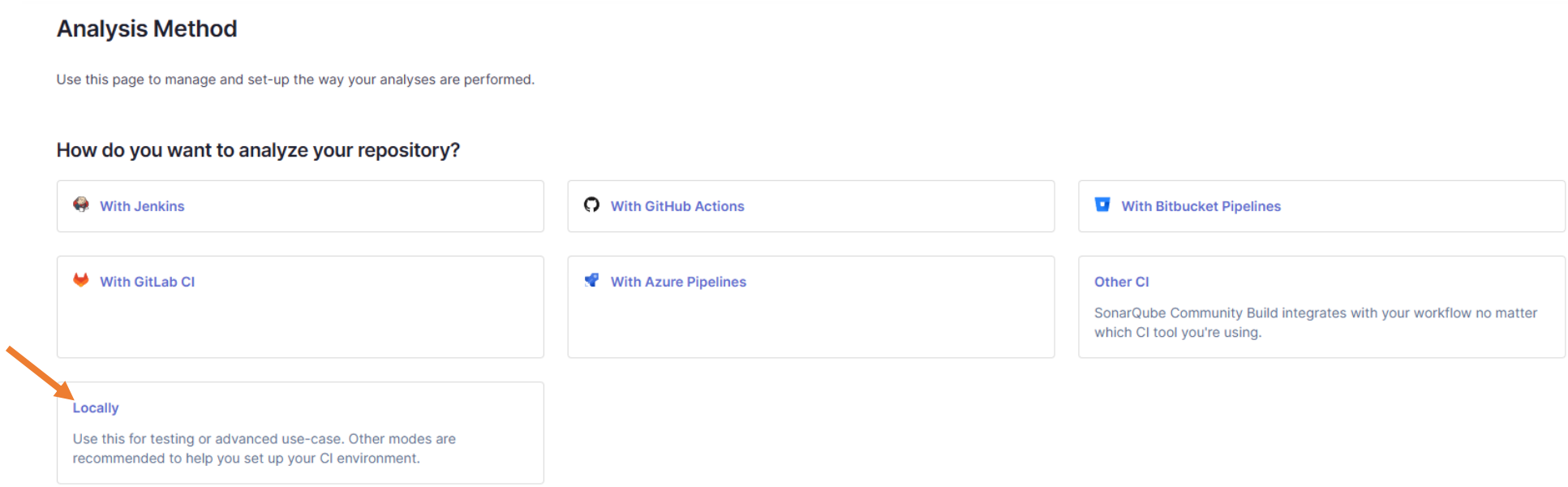
- Dejaremos por defecto el valor que tiene SonarQube



SAST – SonarQube – Integración

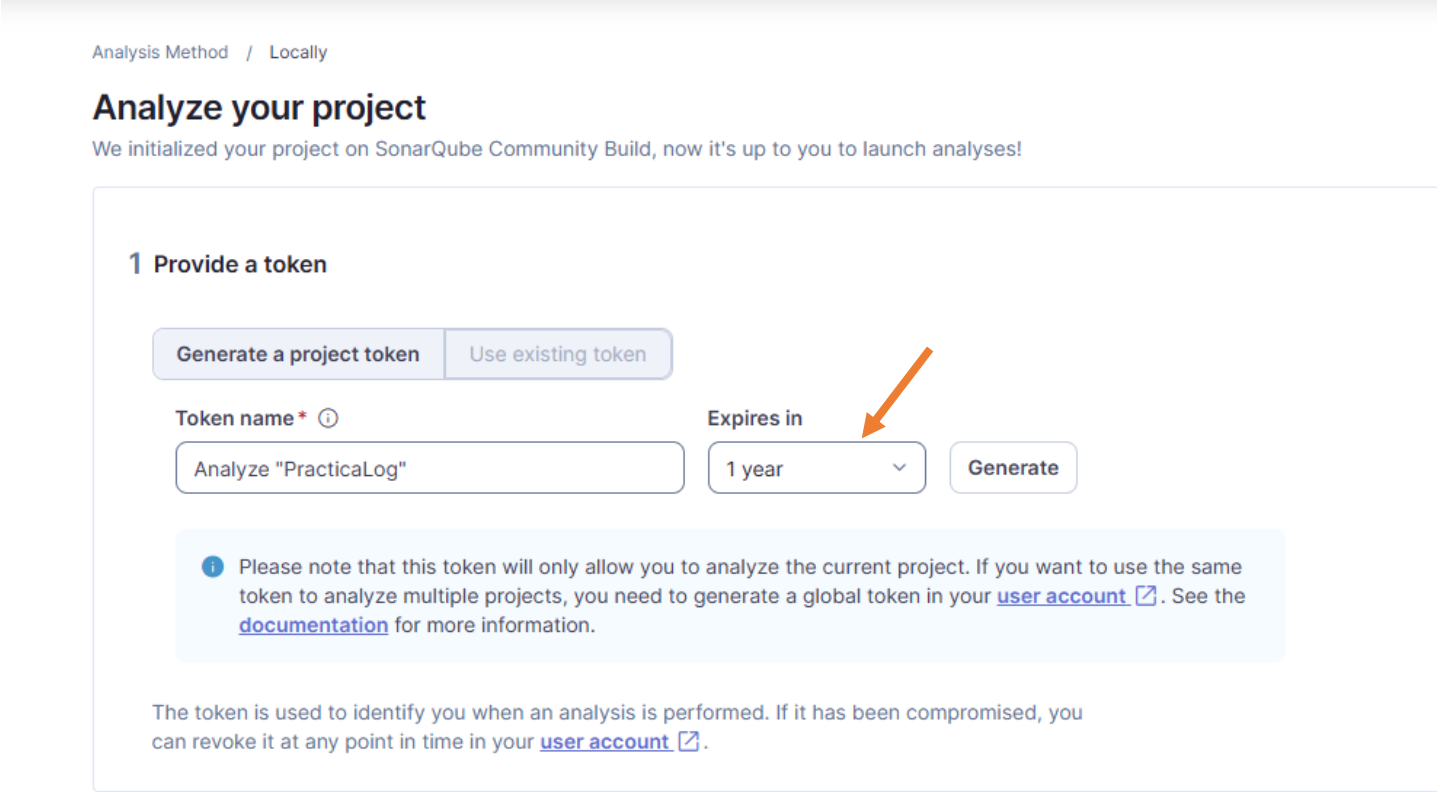
Con el proyecto configurado nos preguntará si queremos usar algún CI/CD existente para analizar el código.

- Nosotros marcaremos local ya que lo haremos vía Maven.



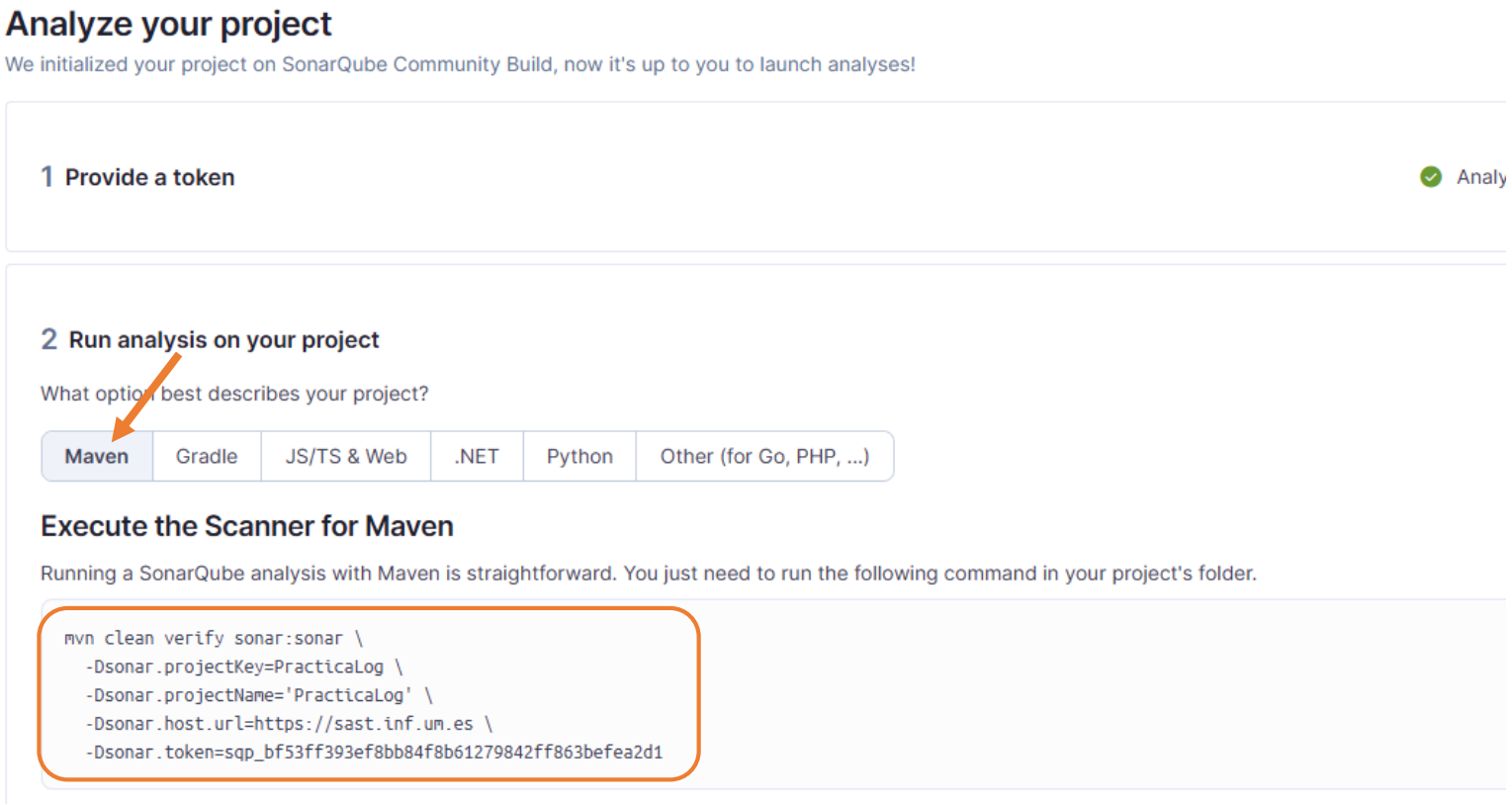
SAST – SonarQube – Integración

- El siguiente paso será generar un token para que podamos invocar a SonarQube para enviarle el código



SAST – SonarQube – Integración

- Por último, nos pregunta cómo queremos generar el análisis. Indicamos que con Maven



SAST – SonarQube – Integración

Lanzando el comando indicado desde local debería funcionarnos, pero en las últimas versiones hay que hacer un ajuste.

NOTA:

Si obtenemos el error:

```
[ERROR] No plugin found for prefix 'sonar' in the current project and in the plugin groups
[org.apache.maven.plugins, org.codehaus.mojo]
```

Debemos añadir el grupo de plugins a sonar. Para ello buscamos el fichero `settings.xml` de la instalación de Maven y añadimos dentro de `<settings>`

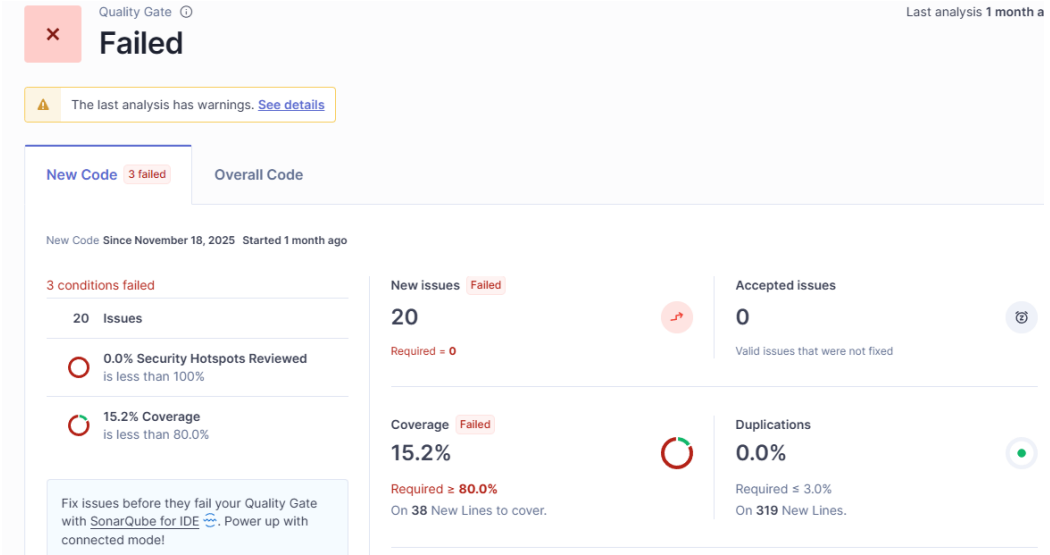
```
<pluginGroups>
    <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
</pluginGroups>
```

SAST – SonarQube – Integración

- Si continuamos teniendo problemas, podemos ejecutarlo indicando el plugin exacto.

```
mvn clean verify org.sonarsource.scanner.maven:sonar-maven-plugin:3.11.0.3922:sonar
-Dsonar.projectKey=[CLAVE_PROYECTO] -Dsonar.projectName=[NOMBRE_PROYECTO]
-Dsonar.host.url=https://sast.inf.um.es -Dsonar.token=[PONER_TOKEN]
```

- Una vez analizado, en SonarQube tendremos acceso a toda la información de ese código



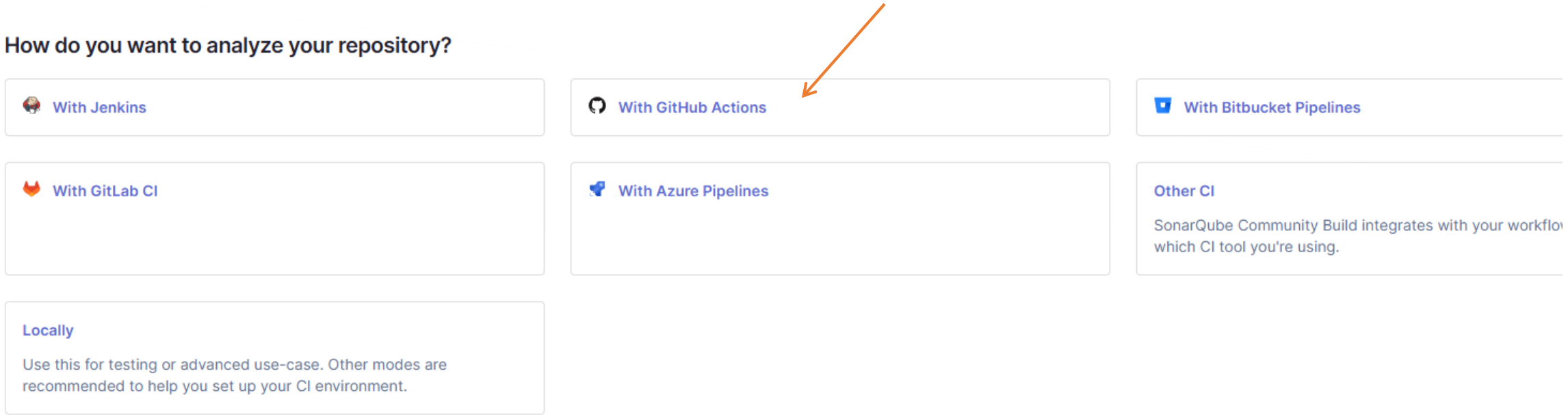
SAST – SonarQube – Integración

Una vez que sabemos enviar nuestro código al servidor SonarQube podemos integrarlo en Github Actions.

- SonarQube por defecto nos da una guía de los pasos a seguir.
- Si ya los conocemos podemos aplicarlo a nuestro proyecto existente, en caso contrario crearemos un nuevo proyecto para acceder a las indicaciones.

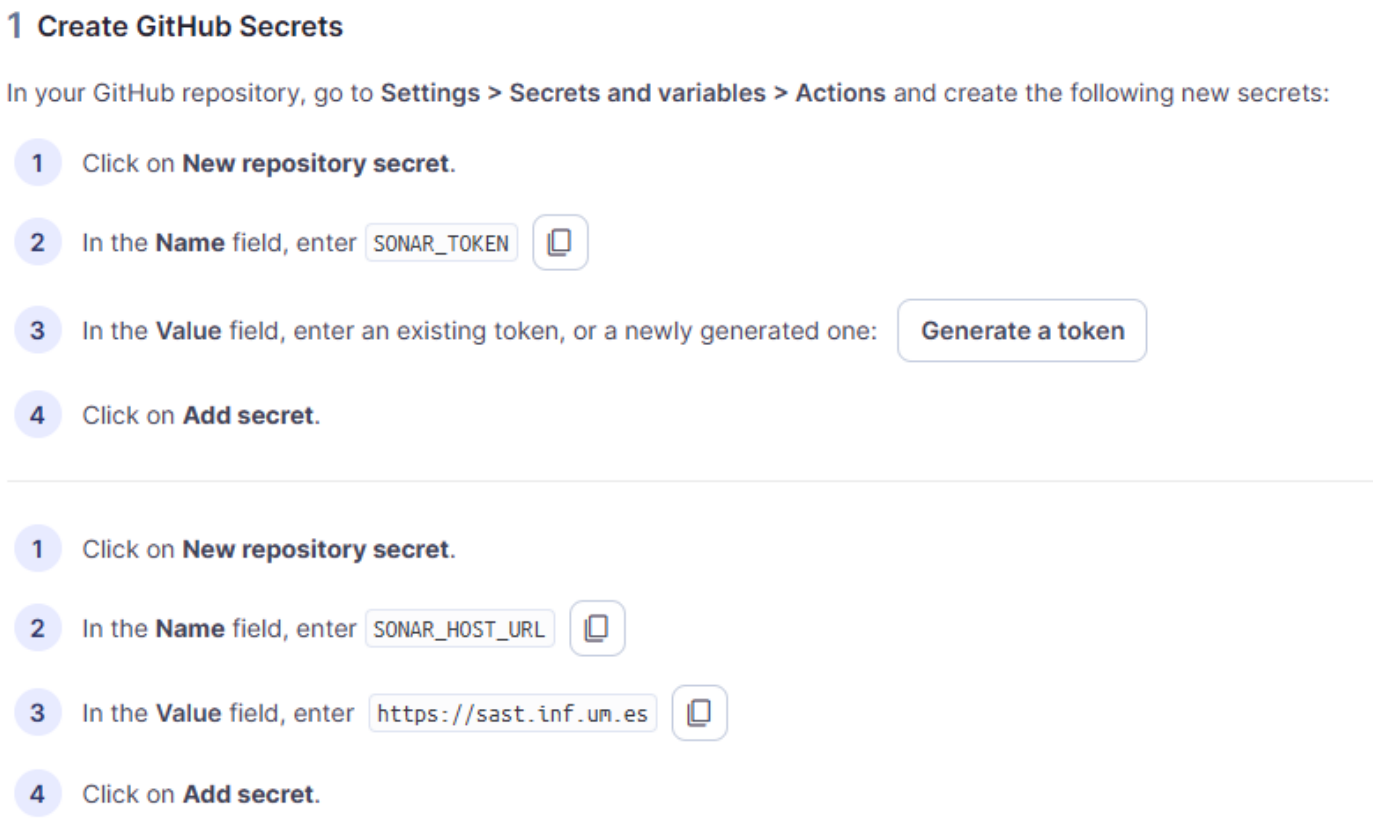
SAST – SonarQube – Integración

En la configuración del análisis, donde indicamos "Locally" ahora elegiremos "With GitHub Actions"



SAST – SonarQube – Integración

1. El primer paso es crear la configuración en Github



1 Create GitHub Secrets

In your GitHub repository, go to **Settings > Secrets and variables > Actions** and create the following new secrets:

1. Click on **New repository secret**.
2. In the **Name** field, enter `SONAR_TOKEN`
3. In the **Value** field, enter an existing token, or a newly generated one: **Generate a token**
4. Click on **Add secret**.

1. Click on **New repository secret**.
2. In the **Name** field, enter `SONAR_HOST_URL`
3. In the **Value** field, enter `https://sast.inf.um.es`
4. Click on **Add secret**.

SAST – SonarQube – Integración

1. El segundo paso es indicar cómo lo vamos a lanzar y SonarQube nos dará directamente el código de la **gitHub Action**.