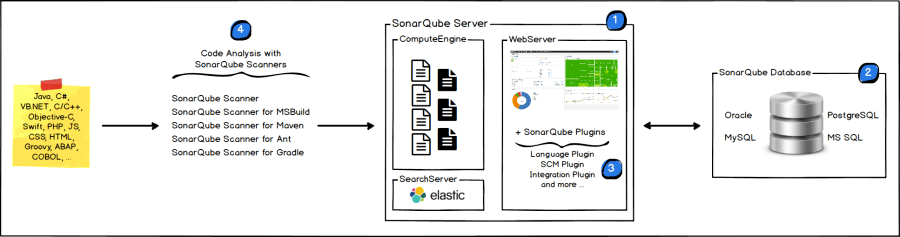
SonarQube是目前业界最流行，体验最好的代码检测工具（产品）。基于SonarQube的sonarcloud跟github一起，体验简直了。

首先，声明一下SonarQube工具不涉A。 涉A的定义是商业软件，所有开源软件和工具不涉及。Sonar为开源LGPL协议。（其大并发场景是收费心的。）

官方WIKI：<https://docs.sonarqube.org/latest/setup/install-server/>

**SonarQube的架构**

架构如下图：



四大件：

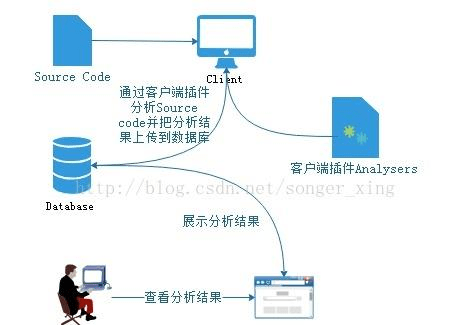
（1）server，包括webserver和执行引擎。

（2）存储，结果存储，支持多款数据库。

（3）插件，支持各种引擎扫描和功能的插件。

（4）Scanner client，代码扫描包。这里定义为包，是指它不是独立的进程，而是一个可执行的包，被jeckins或者其它构建工具引用，在构建节点上执行代码扫描，然后上传到Server，在server上分析展示。

流程如下：



**Sonar的优点：**

（1）支持**所有语言**的检测。一个工具，搞定所有。 <https://rules.sonarsource.com/java>

（2）**灵活扩展，插拔式使用**。自定义的代码检测规则，可自定义插件，独立打成JAR包放到SONARQUBE插件目录下，重启即生效，开发使用非常方便。而且**自带UT验证框架**，开发效率高。

（3）**规则支持多租户隔离**。租户可定制自己的规则集。

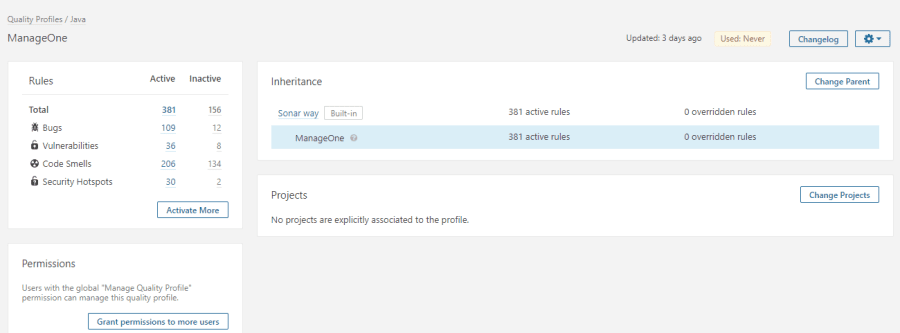
（4）**生态强大**，业界有诸多插件，与jenkins友好集成。

（5）部署使用便捷。

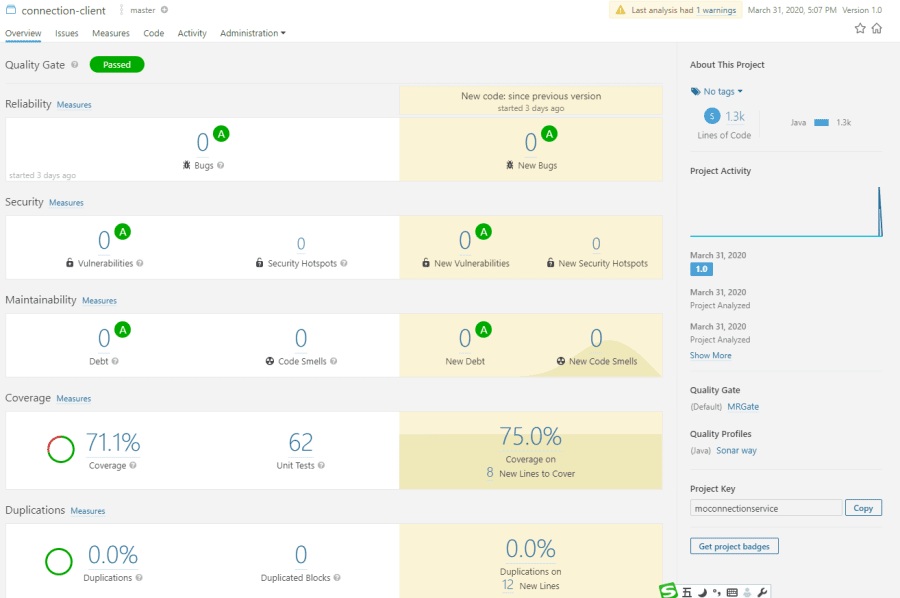
（6）架构松耦合，通过与maven/jenkins等集成，**将代码扫描的计算消耗迁移到业务或者构建方的资源上**，极大的提升了自身的吞吐能力。

**体验**

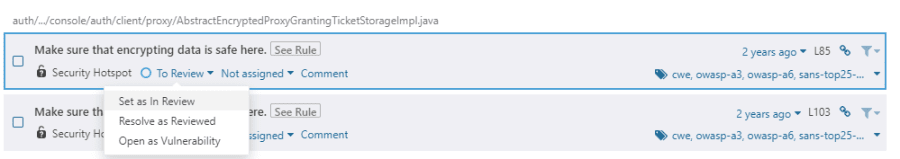
（1）规则自定义

  
自定义自己产品用的规则，应用于自己的项目。

（2）检测效果 -- 度量全面



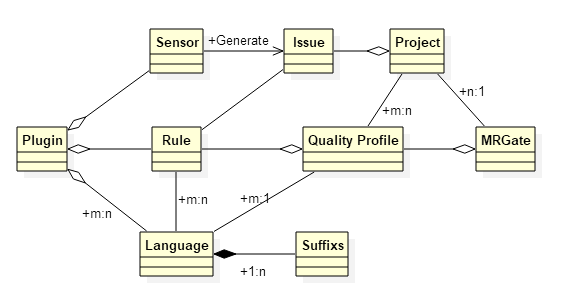
（3）误报屏蔽



（4）支持徽章：<https://gitlab.huawei.com/mo-architecture/connection-client>

http://image.huawei.com/tiny-lts/v1/images/133df26b780b65e06bef_268x49.png@900-0-90-f.png

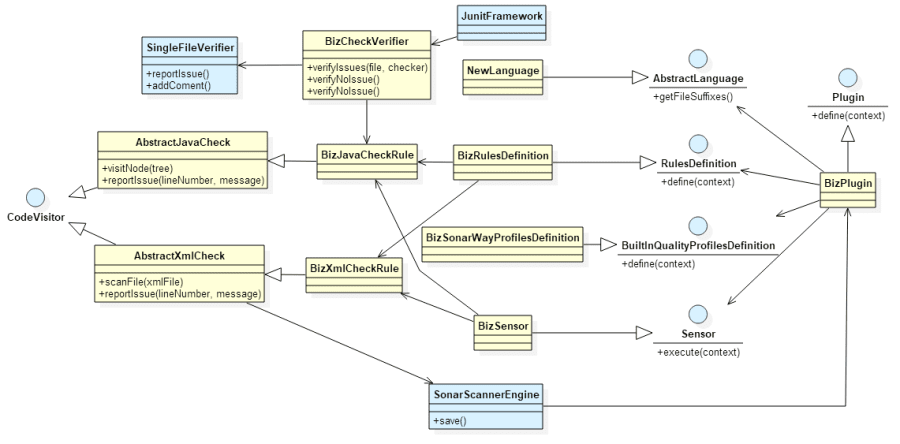
**Sonar的业务模型：**



Sensor表示规则执行引擎。

**规则扩展**

**Sonar插件机制类图如下：**

****

**对于新的语言检查扩展，主要实现插件注册类、规则定义、规则集配置、语言类型、执行代理五个基础公共类，在RulesDefinition和Sensor类中加载具体的检查规则类。**

**检查规则类，实现相对灵活，不同的语言用不同的方法，比如JAVA是用语法树，XML是用XPATH。**

详细过程参照：<https://gitlab.huawei.com/OpenTools/codecheckrules/wikis/home>

代码样例：

@Rule(key = "NotAllowPrintLogInLoop",

name = "Do not allow print log in loop",

description = "Do not allow print log in loop",

tags = {"coding-guideline"},

priority = Priority.MAJOR)

@SqaleConstantRemediation("10min")

public class MOLoopPrintRule extends IssuableSubscriptionVisitor {

private static Pattern pattern = Pattern.compile("((?i).\*LOG.\*)(\\.(fatal|error|warn|info|debug)\\(.\*\\)).\*");

private static Set<Tree.Kind> parentTokens = new HashSet<>();

static {

parentTokens.add(Tree.Kind.WHILE\_STATEMENT);

parentTokens.add(Tree.Kind.DO\_STATEMENT);

parentTokens.add(Tree.Kind.FOR\_EACH\_STATEMENT);

parentTokens.add(Tree.Kind.FOR\_STATEMENT);

}

@Override

public List<Tree.Kind> nodesToVisit() {

return ImmutableList.of(Tree.Kind.EXPRESSION\_STATEMENT);

}

@Override

public void visitNode(Tree tree) {

if (checkLoopPrint(super.context.getFileLines().get(tree.firstToken().line() - 1)) && checkParentToken(tree)) {

super.addIssue(tree.firstToken().line(), "Do not allow print log in loop");

}

}

private boolean checkLoopPrint(String code) {

return pattern.matcher(code).matches();

}

private boolean checkParentToken(Tree tree) {

while (tree != null && tree.kind() != Tree.Kind.METHOD) {

if (tree.parent() != null && parentTokens.contains(tree.parent().kind())) {

return true;

}

tree = tree.parent();

}

return false;

}

}

开发完成后，把规则类注册到插件中就行了。

测试代码样例：

@Test

public void should\_report\_issues\_when\_visit\_whole\_file() {

JavaCheckVerifier.verify("src/test/java/com/huawei/codecheck/rule/MOLoopPrintRuleTest.java", new MOLoopPrintRule());

}

注意被检测代码中，要在有问题要检测的行的后面加上 // Noncomplaint

检查工具会自动加载该行号，作为自检项。

**特殊场景说明：**

场景一：对于明文密码扫描，需要扫全部文件，匹配文件内容。

SONAR同一般代码检测工具一样，是基于文件后缀白名单进行扫描的。如果此文件是小众的，比如config.abc, 或者没有后缀的，那是没办法的。 只能尽量在描述要扫描的文件后缀时列全了。

实现思路： 新开发一个插件，将要扫描的所有文件后缀罗列到该插件中扫描。 但是，像java, xml, go, python等语言的文件，该插件是无法直接扫描的，因为一个文件只能适用于一种语言。 此时需要变通一下，将该扫描规则加载到其它语言的QulityFile中。也就是在该插件的Sensor扩展类中，扩展支持JAVA等其它要扫描 的语言类型即可。

**关于SonarQube的并发检测能力解决方案**

（1）非问题，因为前面说了，它的server只是分析，很大部分的检测负载分到构建资源上了。Sonar几十M的报告，也是秒级完成。实在不行，买sonar的高并发能力，法国的软件，也不涉A。

（2）多套实例部署，分库分实例。像ManageOne产品，微服务架构，大的微服务也就几万行，不存在任何问题。如果仓库多了，就再部署一套，跟分库分表一个思路。 如果有产品百万级代码都在一个仓库里，建议拆库吧。如今git盛行的当下，还把所有东西放一个仓库，实在不可取。

    如果库实在拆不了，SONAR也有办法，就是分开构建。按库下面的目录进行构建和检查，SONAR支持配置代码目录和测试目录。

（3）在SonarQube上盖个帽子，做层分发，解决下访问口令和利旧原则应该就可以了。这个我司是很擅长的。