Secrets
ABAP
Apex
AzureResourceManager
C
C#
C++
CloudFormation
COBOL
CSS
Dart
Docker
Flex
Go
HTML
Java
JavaScript
JCL
Kotlin
Kubernetes
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

Docker static code analysis

Unique rules to find Vulnerabilities, Security Hotspots, and Code Smells in your DOCKER code

All rules 44 | Vulnerability 4 | Bug 4 | Security Hotspot 15 | Code Smell 21

Tags | Impact | Clean code attribute | Search by name...

Double quote to prevent globbing and word splitting
Code Smell

Instructions should be upper case
Code Smell

Allowing non-root users to modify resources copied to an image is security-sensitive
Security Hotspot

Automatically installing recommended packages is security-sensitive
Security Hotspot

Running containers as a privileged user is security-sensitive
Security Hotspot

Delivering code in production with debug features activated is security-sensitive
Security Hotspot

Use ADD instruction to retrieve remote resources
Code Smell

Arguments in long RUN instructions should be sorted
Code Smell

Track uses of "TODO" tags
Code Smell

Descriptive labels are mandatory
Code Smell

Use digest to pin versions of base images
Code Smell

Dockerfile parsing failure
Code Smell

Pulling an image based on its digest is security-sensitive
Security Hotspot

Track uses of "TODO" tags

Analyze your code

Intentionality - Complete | Maintainability

Code Smell | Info | cwe

Why is this an issue? | More Info

Developers often use `TODO` tags to mark areas in the code where additional work or improvements are needed but are not implemented immediately. However, these `TODO` tags sometimes get overlooked or forgotten, leading to incomplete or unfinished code. This rule aims to identify and address unattended `TODO` tags to ensure a clean and maintainable codebase. This description explores why this is a problem and how it can be fixed to improve the overall code quality.

What is the potential impact?

Unattended `TODO` tags in code can have significant implications for the development process and the overall codebase.

Incomplete Functionality: When developers leave `TODO` tags without implementing the corresponding code, it results in incomplete functionality within the software. This can lead to unexpected behavior or missing features, adversely affecting the end-user experience.

Missed Bug Fixes: If developers do not promptly address `TODO` tags, they might overlook critical bug fixes and security updates. Delayed bug fixes can result in more severe issues and increase the effort required to resolve them later.

Impact on Collaboration: In team-based development environments, unattended `TODO` tags can hinder collaboration. Other team members might not be aware of the intended changes, leading to conflicts or redundant efforts in the codebase.

Codebase Bloat: The accumulation of unattended `TODO` tags over time can clutter the codebase and make it difficult to distinguish between work in progress and completed code. This bloat can make it challenging to maintain an organized and efficient codebase.

Addressing this code smell is essential to ensure a maintainable, readable, reliable codebase and promote effective collaboration among developers.

Noncompliant code example

```
# TODO
FROM ubuntu
```

Available In:
sonarlint | sonarcloud | sonarqube