





































-  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...

🔍

 Code Smell
Package update should not be executed without installing it
 Code Smell
Cache should be cleaned after package installation
 Code Smell
Deprecated instructions should not be used
 Code Smell
Consent flag should be set to avoid manual input
 Code Smell
Environment variables should not be unset on a different layer than they were set
 Code Smell
Expanded filenames should not become options
 Code Smell
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

Specific version tag for image should be used

Intentionality - Logical

Maintainability

👉

 Code Smell

 Major

🔍

When a Dockerfile image is not tagged with a specific version, it is referred to as `latest`. This means that every time the image is built, deployed, or run, it will always use the latest version of the image.

Why is this an issue?

How can I fix it?

More Info

To avoid these issues, it is recommended to use specific version tags for Dockerfile images.

This can be done by appending the version number or tag to the Dockerfile image name. For example, instead of `my-image:latest`, it is better to use `my-image:1.2.3-alpine` or `my-image:1.2.3`.

For even more control and traceability, it is also possible to specify your image by digest using the sha256 of the image. This will pin your image to a specific version in time, but will also exclude it from eventual security updates. An example would be using `my-image@sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13ab40cbec258c68d`.

More information can be found in the documentation at the end.

Code examples

Noncompliant code example

```
FROM my-image
FROM my-image:latest
```

Compliant solution

```
FROM my-image:1.2.3
FROM my-image:1.2.3-alpine
```

How does this work?

This way, the same version of the Dockerfile image is used every time the application is built, deployed, or run, ensuring consistency and predictability across different environments. It is also not enough to use the latest tag, as this version also changes with each release.

Going the extra mile

Adhering to this can also make it easier to track which version of the Dockerfile image is being used, which can be useful for debugging and troubleshooting purposes.

Available In:

sonarlint



sonarcloud



sonarqube



Analyze your code

