





































-  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... 🔍

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

Cache should be cleaned after package installation

Analyze your code

Intentionality - Efficient

Maintainability 🟡

Code Smell

Major ⚠️

In Docker, when packages are installed via a package manager, an index is cached locally by default. This index should either be cleaned up or stored in a dedicated cache mount.

Why is this an issue?

How can I fix it?

More Info

Code examples

Noncompliant code example

For apk:

```
RUN apk add nginx
```

For apt-get:

```
RUN apt-get update \  
&& apt-get install nginx
```

For aptitude:

```
RUN aptitude update \  
&& aptitude install nginx
```

For apt:

```
RUN apt update \  
&& apt install nginx
```

For apt-get, without a cache mount:

```
RUN apt-get update \  
&& apt-get install nginx
```

Compliant solution

For apk:

```
RUN apk --no-cache add nginx  
  
RUN apk add nginx \  
&& apk cache clean  
  
RUN apk add nginx \  
&& rm -rf /var/cache/apk/*  
  
# This cache location is only used in specific distributions / configurations  
RUN apk add nginx \  
&& rm -rf /etc/apk/cache/*
```

For apt-get:

```
RUN apt-get update \  
&& apt-get install nginx \  
&& apt-get clean  
  
RUN apt-get update \  
&& apt-get install nginx \  
&& rm -rf /var/lib/apt/lists/* /var/cache/apt/archives/*
```

For aptitude:

```
RUN aptitude update \  
&& aptitude install nginx \  
&& aptitude clean  
  
RUN aptitude update \  
&& aptitude install nginx \  
&& rm -rf /var/lib/apt/lists/* /var/cache/apt/archives/*
```

For apt:

```
RUN apt update \  
&& apt install nginx \  
&& apt clean  
  
RUN apt update \  
&& apt install nginx \  
&& rm -rf /var/lib/apt/lists/* /var/cache/apt/archives/*
```

For apt-get, with a cache mount:

```
RUN \  
--mount=type=cache,target=/var/cache/apt,sharing=locked \  
--mount=type=cache,target=/var/lib/apt,sharing=locked \  
apt-get update \  
&& apt-get install nginx
```

How does this work?

When installing packages using `apt-get`, `aptitude` or `apt` these package managers store an index in the Docker image layer in `/var/lib/apt/lists`. Using `apk`, it will store an index in `/var/cache/apk/`. In some distributions and configurations, the cache will be created in `/etc/apk/cache`.

This index is not needed after installation, so it can be removed. To do that, execute the `clean` command, or run `rm -rf <location>` for the cache location of you package manager tool.

Additionally, for `apt-get`, `aptitude` and `apt` some lock files are stored in `/var/cache/apt/archives`, which can also be removed safely. They are not removed by the `clean` command, so they need to be removed manually.

Alternatively, store the cache in a dedicated cache mount. A cache mount can be created by adding a flag `--mount type=cache` to the `RUN` command. This will store the cache in a Docker volume, which will be persisted between builds making the build faster.

Available In:
sonarlint  | **sonarcloud**  | **sonarqube** 