

10 个 Docker 镜像安全最佳实践


原创：柳泉波 译 Docker 5月16日



柳泉波

华南师范大学教师，DockOne社区金牌翻译

《Docker 镜像安全最佳实践速查表[1]》列举了 10 个诀窍和指南，确保更安全和更高质量的 Docker 镜像处理。此外，还可以检视有关 Docker 安全的新报告《Docker 安全要趁早[2]》。

**snyk**

**Docker
Image
Security Best
Practices**

Authors:
 @liran_tal
Node.js Security WG & Developer Advocate at Snyk
 @omerlh
DevSecOps Engineer at Solutio

<https://snyk.io>

1. Prefer minimal base images

In Snyk's State of open source security report 2019, we found each of the top ten docker images to include as many as 580 vulnerabilities in their system libraries.

- Choose images with fewer OS libraries and tools lower the risk and attack surface of the container
- Prefer alpine-based images over full-blown system OS images

2. Least privileged user

Create a dedicated user and group on the image, with minimal permissions to run the application; use the same user to run this process. For example, Node.js image which has a built-in node generic user:

```
FROM node:10-alpine
USER node
CMD node index.js
```

3. Sign and verify images to mitigate MITM attacks

We put a lot of trust into docker images. It is critical to make sure the image we're pulling is the one pushed by the publisher, and that no one has tampered with it.

- Sign your images with the help of Notary
- Verify the trust and authenticity of the images you pull

4. Find, fix and monitor for open source vulnerabilities

Scan your docker images for known vulnerabilities and integrate it as part of your continuous integration. Snyk is an open source tool that scans for security vulnerabilities in open source application libraries and docker images.

Use Snyk to scan a docker image:
\$ snyk test --docker node:10 --file=path/to/Dockerfile

Use Snyk to monitor and alert to newly disclosed vulnerabilities in a docker image:
\$ snyk monitor --docker node:10

5. Don't leak sensitive information to docker images

It's easy to accidentally leak secrets, tokens, and keys into images when building them. To stay safe, follow these guidelines:

- Use multi-stage builds
- Use the Docker secrets feature to mount sensitive files without caching them (supported only from Docker 18.04).
- Use a .dockerignore file to avoid a hazardous COPY instruction, which pulls in sensitive files that are part of the build context

6. Use fixed tags for immutability

Docker image owners can push new versions to the same tags, which may result in inconsistent images during builds, and makes it hard to track if a vulnerability has been fixed. Prefer one of the following:

- A verbose image tag with which to pin both version and operating system, for example: FROM node:8-alpine
- An image hash to pin the exact content, for example: FROM node:<hash>

7. Use COPY instead of ADD

Arbitrary URLs specified for ADD could result in MITM attacks, or sources of malicious data. In addition, ADD implicitly unpacks local archives which may not be expected and result in path traversal and Zip Slip vulnerabilities.

Use COPY, unless ADD is specifically required.

8. Use labels for metadata

Labels with metadata for images provide useful information for users. Include security details as well.

Use and communicate a Responsible Security Disclosure policy by adopting a SECURITY.TXT policy file and providing this information in your images labels.

9. Use multi-stage builds for small secure images

Use multi-stage builds in order to produce smaller and cleaner images, thus minimizing the attack surface for bundled docker image dependencies.

10. Use a linter

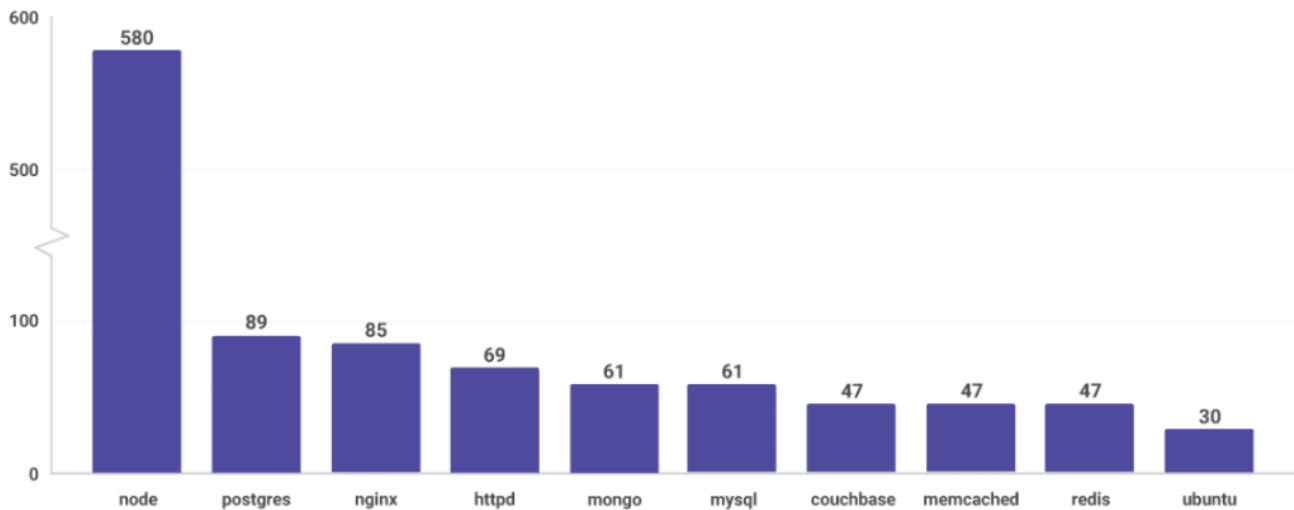
Enforce Dockerfile best practices automatically by using a static code analysis tool such as hadolint linter, that will detect and alert for issues found in a Dockerfile.

1. 选用最小化基础镜像

人们编写项目的 Dockerfile 时，经常使用一个通用的 Docker 容器镜像作为基础，例如 From Node 。 Node 镜像实际上是以一个完整安装的 Debian Stretch 发行版为基础，这意味着构建得到的项目容器镜像将包含一个完整的操作系统。如果该项目不需要任何通用的系统库或者系统工具应用，最好不要使用完整的操作系统作为基础镜像。

Synx 发布的《开源安全报告-2019[3]》指出，Docker Hub 上流行的很多容器镜像，都用到了包含大量已知安全漏洞的基础镜像。例如，执行 `docker pull node`，下载并使用 Node 镜像，相当于在应用中引入了一个包含 580 个已知漏洞的操作系统。

Number of OS vulnerabilities by docker image



从上图（摘自《开源安全报告-2019》）可知，Docker Hub 上最流行的 10 个镜像都包含已知的安全漏洞。选用最小化基础镜像，即只包含项目确实需要的系统工具和库的镜像，就能最小化系统的攻击面，确保所用操作系统是安全的。

了解更多 Docker 镜像安全的知识[4]。

2. 设定最小权限的 USER

如果 Dockerfile 中没有指定 USER，Docker 默认将会以超级用户 root 的身份运行容器，容器所属的命名空间（namespace）因此映射为 root 所有，这意味着容器有可能获取 Docker 宿主机的超级管理权限。不仅如此，以 root 用户身份运行容器，还扩大了攻击面，如果容器应用中存在安全漏洞，很容易造成权限提升。

在实践中，一般不需要容器拥有 root 权限。为了尽量降低安全威胁，创建专门的用户和用户组，在 Dockerfile 中使用 USER 指定用户，确保以最小权限的用户身份运行容器应用。

如果基础镜像中不包含专门的用户，那么就在 Dockerfile 中直接创建。下面就是一个这样的例子，它用到的基础镜像是 Ubuntu：

```
FROM ubuntu
RUN mkdir /app
RUN groupadd -r lirantal && useradd -r -s /bin/false -g lirantal lirantal
WORKDIR /app
COPY . /app
RUN chown -R lirantal:lirantal /app
USER lirantal
CMD node index.js
```

在上例中：

- 创建一个系统用户（-r 选项），没有密码、没有主目录且没有 shell；
- 将该用户添加到前面（使用 groupadd）创建的用户组；
- 最后一段参数设定了用户名以及所属的用户组。

如果你使用的是 Node.js 和 alpine 镜像，已经包含了一个用户 node，直接使用即可：

```
FROM node:10-alpine
RUN mkdir /app
COPY . /app
RUN chown -R node:node /app
USER node
CMD ["node", "index.js"]
```

Node.js 应用开发者请参阅官方的 Docker 和 Node.js 最佳实践[5]。

3. 签名和校验镜像，防范中间人攻击

Docker 镜像的认证颇具挑战性。在生产环境使用这些镜像运行我们的代码，意味着我们对这些镜像的极大信任。因此，必须保证我们拉取的容器镜像确实是发布者发布的镜像，没有被任何人篡改。发生镜像篡改，有可能是因为 Docker 客户端和镜像中心之间的中间人攻击，或者是发布者的身份被人盗用并在镜像中心发布了恶意镜像。

校验 Docker 镜像

Docker 默认直接拉取容器镜像，不会校验镜像的来源和发布者。这意味着你有可能使用来源和发布者不明的任何镜像。

无论采用何种策略，最佳实践都是先校验容器镜像，通过验证后再拉取镜像。为了体验镜像校验功能，执行下列暂时开启 Docker Content Trust 的命令：

```
export DOCKER_CONTENT_TRUST=1
```

现在，尝试拉取一个没有签名的容器镜像——请求会被拒绝，不会拉取镜像。

签名 Docker 镜像

优先使用 Docker 认证的镜像，即这些镜像来自经过 Docker Hub 检查和选择的可信提供者。不要使用无法检验来源和发布者的容器镜像。

Docker 支持镜像签名，提供了额外一层的保护。使用 Docker Notary 签名镜像。Notary 会检验镜像的签名，如果签名不合法，它会阻止运行该镜像。

如果开启了 Docker Content Trust，构建 Docker 镜像的同时也会对镜像签名。如果是第一次签名，Docker 会为当前用户生成一个私钥，保存在 `~/docker/trust`。后续所有的镜像都会使用这个私钥签名。

请参考 Docker 官方文档[6]，了解签名镜像的详细指令。

4. 找出、修正和监控开源漏洞

指定容器的基础镜像，同时也引入了该镜像包含的操作系统及系统库有可能存在的所有安全风险。

最好选用能够正常运行应用代码的最小化镜像，这有助于减少攻击面，因为限制了可能的安全漏洞数量。不过，这么做并没有对镜像进行安全审计，也不能防范将来发现的新漏洞。

因此，防范安全软件漏洞的一种方法是使用像 Snyk 这样的工具，持续扫描和监控 Docker 镜像各层可能存在的漏洞。

使用下列命令扫描容器镜像，检查是否存在已知漏洞：

```
# fetch the image to be tested so it exists locally
$ docker pull node:10
# scan the image with snyk
$ snyk test --docker node:10 --file=path/to/Dockerfile
```

Snyk 能够监控指定的容器镜像，一旦有新发现的安全漏洞，通知用户并给出修补建议：

```
$ snyk monitor --docker node:10
```

根据 Snyk 用户执行的镜像扫描，我们发现大约 40% 的 Docker 镜像包含已知漏洞，实际上弥补这些漏洞的新版本基础镜像已经有了。Synx 提供了绝无仅有的修正建议功能，用户可以根据建议采取行动，升级 Docker 镜像。

Snyk 还发现在扫描的所有镜像中，为了减少漏洞的数量，大约 20% 的镜像需要重新构建。更多信息请参阅《开源安全报告-2019[3]》。

5. 不要在容器镜像中包含机密信息

有时候，构建包含应用的容器镜像时，需要用到一些机密信息，例如从私有仓库拉取代码所需的 SSH 私钥，或者安全私有软件包所需的令牌。如果 Dockerfile 中包含复制机密信息的命令，构

建镜像时，这行命令对应的中间容器会被缓存，导致机密数据也被缓存，有可能造成机密信息泄漏。因此，像令牌和密钥这样的机密信息必须保存在 Dockerfile 之外。

使用多阶段构建

利用 Docker 的多阶段构建功能，用一个中间镜像层获取和管理机密信息，然后清除中间镜像，这样在应用镜像构建阶段不涉及敏感数据。如下面例子所示，使用代码将机密信息添加到中间层：

```
FROM: ubuntu as intermediate

WORKDIR /app
COPY secret/key /tmp/
RUN scp -i /tmp/key build@acme/files .

FROM ubuntu
WORKDIR /app
COPY --from intermediate /app .
```

使用 Docker 的 secret 管理功能

使用 Docker 的 secret 管理功能（alpha 阶段），加载敏感信息文件且不会缓存这些信息：

```
# syntax = docker/dockerfile:1.0-experimental
FROM alpine

# shows secret from default secret location
RUN --mount=type=secret,id=mysecret cat /run/secrets/mysecre

# shows secret from custom secret location
RUN --mount=type=secret,id=mysecret,dst=/foobar cat /foobar
```

想了解有关 Docker secret 的更多信息，请访问 Docker 官方站点[6]。

避免无意中复制机密信息

往镜像中复制文件时，也要当心，避免无意中添加了机密信息。例如，下面的命令将整个构建上文文件夹复制到 Docker 镜像，有可能把敏感文件也复制进去了：

```
COPY . .
```

如果文件夹中有敏感文件，要么先移除这些文件，要么将这些文件包含在 `.dockerignore` 中，复制时会忽略这些文件：

```
private.key  
appsettings.json
```

6. 设定镜像的标签，保证镜像的不可更改性

每个 Docker 镜像可以有多个标签（tag），代表该镜像的不同变体。最常见的标签是 `latest`，表示这是该镜像的最新版本。镜像标签是可更改的，也就是说镜像的作者可以多次发布相同标签的镜像。

因此，即使你的 `Dockerfile` 明确指定了使用的基础镜像及其标签，这次镜像构建和下次镜像构建仍然可能用到了不同的基础镜像。解决这个问题，有多种办法：

- 优先选用最详细的镜像标签。例如，镜像有 `:8`、`:8.0.1` 和 `:8.0.1-alpine` 等标签，选择最后一个，因为它提供了最详细的信息。不要使用像 `latest` 这样过于泛泛的标签。
- 记住，镜像的发布者有可能删除镜像的某个标签。如果设定了所用镜像的标签，一旦这个标签被删除，镜像构建会因为找不到基础镜像而失败。为了避免这个问题，可以提前把该镜像复制到私有镜像中心或者公有镜像中心的私人账户下面。这么做，保证了镜像的不可更改性，同时也带来了维护私有镜像中心的负担。
- 使用比签名更具体的 SHA256 引用指明要使用的镜像，这能保证每次拉取都是相同内容的镜像。这么做也有风险，如果镜像改变了，以前的 SHA256 引用（散列值）也不存在了。

7. 使用 COPY，不要使用 ADD

译者警告：这部分对 `ADD` 和 `COPY` 的描述，与 Docker 官方文档并不吻合，译者按照自己的理解修改了这部分内容。如果要了解作者原意，请阅读英文原文[7]。

从宿主机复制文件到容器镜像中的 Docker 命令有两个：`COPY` 和 `ADD`，这两个命令本质上很相似，但具体功能并不相同：

- **COPY** - 将本地文件或者目录（递归）复制到容器镜像中的目标目录，复制来源和目标都必须明确指定。
- **ADD** - 与 **COPY** 类似的功能，有两个不同：（1）如果复制来源是本地压缩文件，**ADD** 将该文件解压缩到目标目录；（2）**ADD** 也可以将远程 URL 指定的文件下载到目标目录。

为了避免可能导致的安全问题，请记住 **COPY** 和 **ADD** 的不同：

- 使用 **ADD** 从远程 URL 下载文件，存在中间人攻击的风险，文件内容有可能因此被篡改。必须确保远程 URL 必须是安全的 TLS 链接，校验远程 URL 的来源和身份。译者注：实际上，官方文档并不鼓励使用 **ADD** 添加远程文件。
- 如果复制的是本地压缩文件，**ADD** 自动将它解压缩到目标目录，这有可能触发 zip 炸弹或者 zip 任意文件覆盖漏洞。
- 相比较而言，使用 **COPY** 复制文件或目录，会创建一个缓存的中间镜像层，优化镜像构建的速度。

8. 使用 LABEL 指定镜像元数据

镜像元数据有助于用户更好地理解和使用该镜像。最常见的元数据是 **maintainer**，它说明了镜像维护者的电邮地址和名字。使用 **LABEL** 命令添加镜像的元数据：

```
LABEL maintainer="me@acme.com"
```

除了镜像的维护者信息，添加其他你认为重要的元数据，包括提交对象的散列值、相关构建的链接、质量状态（通过所有测试了吗？）、源代码链接、**SECURITY.TXT** 文件的位置等。

SECURITY.TXT (RFC5785)[8] 文件说明了镜像维护者的安全披露政策。最好在镜像元数据中加上 **SECURITY.TXT** 的链接，例如：

```
LABEL securitytxt="https://www.example.com/.well-known/security.txt"
```

想了解镜像元数据的更多信息，请访问 <https://label-schema.org/rc1/>。

译者注：这个规范好像已经废止了，请直接访问 OCI 镜像规范[9]。

9. 使用多阶段构建小而安全的镜像

使用 Dockerfile 构建应用容器镜像时，会生成很多只是构建时需要的镜像层，包括编译时所需的开发工具和库，运行单元测试所需的依赖、临时文件、机密信息等等。

如果保留这些镜像层，不仅会增加镜像的大小，影响镜像下载速度，而且会因为安装更多软件包而面临更大的攻击危险。这对用到的镜像也是成立的——需要使用一个专门构建应用的镜像，但不会用它来运行应用代码。

Go 语言就是一个很好的例子。构建一个 Go 应用需要用到 Go 编译器。编译得到的 Go 应用能够在任何操作系统上直接运行，没有任何依赖，包括 scratch 镜像。

Docker 因此提供了多阶段构建的功能，允许在构建过程中使用多个临时镜像，只保留最后一个镜像。这样，用户得到两个镜像：

- 第一个镜像——非常大的镜像，包含了构建应用和运行测试所需的所有依赖；
- 第二个镜像——非常小的镜像，只包含运行应用所需的极少数依赖。

10. 使用静态分析工具

使用静态分析工具，能够避免常见的错误，建立工程师自动遵循的最佳实践指南。

例如，hadolint 分析 Dockerfile 并列出不符合最佳实践规则的地方。

```
/tmp$ docker run --rm -i hadolint/hadolint < Dockerfile-test
/dev/stdin:3 DL4000 MAINTAINER is deprecated
/dev/stdin:6 DL3008 Pin versions in apt get install. Instead of `apt-get install <package>` use `apt-get install <package>=<version>`
/dev/stdin:6 DL3009 Delete the apt-get lists after installing something
/dev/stdin:6 DL3016 Pin versions in npm. Instead of `npm install <package>` use `npm install <package>@<version>`
/dev/stdin:6 DL3015 Avoid additional packages by specifying `--no-install-recommends`
/dev/stdin:17 DL3020 Use COPY instead of ADD for files and folders
/dev/stdin:18 DL3020 Use COPY instead of ADD for files and folders
/tmp$
```

在集成开发环境（IDE）中使用 hadolint 更好。例如，安装 VS Code 的 hadolint 扩展后，编写 Dockerfile 时，边写边检查，既快又好。

了解更多 Docker 镜像安全的知识[10]。

把速查表打印出来，贴在某处，时刻提醒自己构建容器镜像时应该遵循的最佳安全实践！

相关链接：

1. https://res.cloudinary.com/snyk/image/upload/v1551798390/Docker_Image_Security_Best_Practices_.pdf
2. <https://snyk.io/blog/shifting-docker-security-left/>
3. <https://snyk.io/blog/top-ten-most-popular-docker-images-each-contain-at-least-30-vulnerabilities/>
4. <https://snyk.io/container-vulnerability-management/>
5. <https://github.com/nodejs/docker-node/blob/master/docs/BestPractices.md>
6. https://docs.docker.com/engine/security/trust/content_trust/
7. <https://snyk.io/blog/10-docker-image-security-best-practices/>
8. <https://securitytxt.org/>
9. <https://github.com/opencontainers/image-spec>
10. <https://snyk.io/container-vulnerability-management/>

原文链接：<https://snyk.io/blog/10-docker-image-security-best-practices/>

Kubernetes入门与进阶实战培训

Kubernetes入门与进阶实战培训将于2019年6月14日在北京开课，3天时间带你系统掌握Kubernetes，学习效果不好可以继续学习。本次培训包括：Docker基础、容器技术、Docker镜像、数据共享与持久化、Docker三驾马车、Docker实践、Kubernetes基础、Pod基础与进阶、常用对象操作、服务发现、Helm、Kubernetes核心组件原理分析、Kubernetes服务质量保证、调度详解与应用场景、网络、基于Kubernetes的CI/CD、基于Kubernetes的配置管理等，点击下方图片或者点击阅读原文了解详情。



[阅读原文](#)