

We use cookies to ensure you get the best experience on our website. [Got it](#)



The screenshot shows the Snyk logo and the title "Docker Image Security Best Practices". It lists seven best practices:

- 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
- 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.

Enter your email address to subscribe

SUBSCRIBE NOW

10 Docker Security Best Practices



MARCH 6, 2019 | IN [CONTAINER SECURITY](#), [DEVSECOPS](#), [OPEN SOURCE](#)

BY LIRAN TAL, OMER LEVI HEVRONI



In this installment of our cheat sheets, we'd like to focus on Docker security and discuss docker security best practices and guidelines that ensure a more secure and

We use cookies to ensure you get the best experience on our website. [Got it](#)

Check out our [Docker security report: Shifting Docker security left](#)



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 contact, 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.

DOWNLOAD THE CHEAT SHEET!

Let's get started with our list of 10 Docker security best practices.

1. Prefer minimal base images

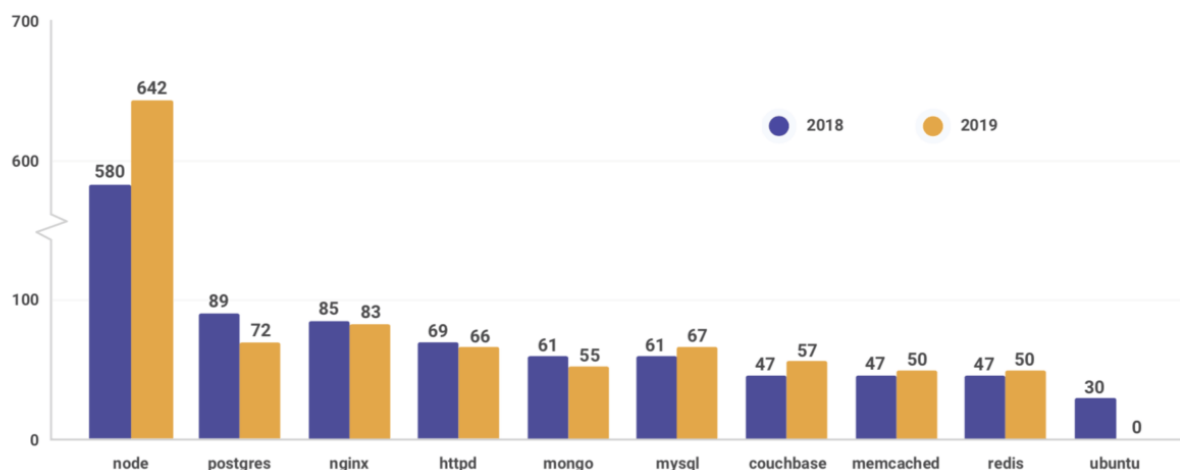
A common docker container security issue is that you end up with big images for your docker containers. Often times, you might start projects with a generic

We use cookies to ensure you get the best experience on our website. [Got it](#)

default . However, when specifying the node image, you should take into **snyk** tion that the fully installed Debian Stretch distribution is the underlying image that is used to build it. If your project doesn't require any general system libraries or system utilities then it is better to avoid using a full blown operating system (OS) as a base image.

In Snyk's [State of open source security report 2020](#) , we found that many of the popular Docker containers that are featured on the Docker Hub website are bundling images that contain many known vulnerabilities. For example, when you use a generic and popularly downloaded node image such as `docker pull node` , you are actually introducing a full blown operating system into your application that is known to have 642 vulnerabilities in its system libraries. This ends up adding unnecessary docker security issues from the get-go

Vulnerabilities in official container images



Top ten most popular docker images each contain at least 30 vulnerabilities

Taken from the [open source security report 2020](#), as can be seen, each of the top ten Docker images we inspected on Docker Hub contained known vulnerabilities, except for Ubuntu. By preferring minimal images that bundle only the necessary system tools and libraries required to run your project, you are also minimizing the

We use cookies to ensure you get the best experience on our website. [Got it](#)

 [Learn more about securing your docker images](#)

2. Least privileged user

When a `Dockerfile` doesn't specify a `USER`, it defaults to executing the container using the root user. In practice, there are very few reasons why the container should have root privileges and it could very well manifest as a docker security issue. Docker defaults to running containers using the root user. When that namespace is then mapped to the root user in the running container, it means that the container potentially has root access on the Docker host. Having an application on the container run with the root user further broadens the attack surface and enables an easy path to privilege escalation if the application itself is vulnerable to exploitation.

To minimize exposure, opt-in to create a dedicated user and a dedicated group in the Docker image for the application; use the `USER` directive in the `Dockerfile` to ensure the container runs the application with the least privileged access possible.

A specific user might not exist in the image; create that user using the instructions in the `Dockerfile`.

The following demonstrates a complete example of how to do this for a generic Ubuntu image:

```
FROM ubuntu
RUN mkdir /app
RUN groupadd -r lirantal && useradd -r -s /bin/false -g lirantal lirantal
WORKDIR /app
COPY . /app
```

We use cookies to ensure you get the best experience on our website. [Got it](#)

```
USER 1001:1001
snyk de index.js
```

The example above:

- creates a system user (-r), with no password, no home directory set, and no shell
- adds the user we created to an existing group that we created beforehand (using groupadd)
- adds a final argument set to the user name we want to create, in association with the group we created

If you're a fan of Node.js and alpine images, they already bundle a generic user for you called `node`. Here's a Node.js example, making use of the generic node user:

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

If you're developing Node.js applications, you may want to consult with the official [Docker and Node.js Best Practices](#).

Test your docker images for vulnerabilities

```
node / node:6-slim
```

We use cookies to ensure you get the best experience on our website. [Got it](#)



TEST FOR FREE

By submitting this form you consent to us emailing you occasionally about our products and services. You can unsubscribe from emails at any time, and we will never pass your email onto third parties. [Privacy Policy](#)

3. Sign and verify images to mitigate **MITM attacks**

Authenticity of Docker images is a challenge. We put a lot of trust into these images as we are literally using them as the container that runs our code in production. Therefore, it is critical to make sure the image we pull is the one that is pushed by the publisher, and that no party has modified it. Tampering may occur over the wire, between the Docker client and the registry, or by compromising the registry of the owner's account in order to push a malicious image to.

Verify docker images

Docker defaults allow pulling Docker images without validating their authenticity, thus potentially exposing you to arbitrary Docker images whose origin and author aren't verified.

Make it a best practice that you always verify images before pulling them in, regardless of policy. To experiment with verification, temporarily enable Docker Content Trust with the following command:

```
export DOCKER_CONTENT_TRUST=1
```

We use cookies to ensure you get the best experience on our website. [Got it](#)
and the image is not pulled.



Sign docker images

Prefer [Docker Certified](#) images that come from trusted partners who have been vetted and curated by Docker Hub rather than images whose origin and authenticity you can't validate.

Docker allows signing images, and by this, provides another layer of protection. To sign images, use [Docker Notary](#). Notary verifies the image signature for you, and blocks you from running an image if the signature of the image is invalid.

When Docker Content Trust is enabled, as we exhibited above, a Docker image build signs the image. When the image is signed for the first time, Docker generates and saves a private key in `~/docker/trust` for your user. This private key is then used to sign any additional images as they are built.

For detailed instructions on setting up signed images, refer to [Docker's official documentation](#).

How is signing docker images with Docker's Content Trust and Notary different from using GPG? Diogo Mónica has a [great talk](#) on this but essentially GPG helps you with verification, not with replay attacks.

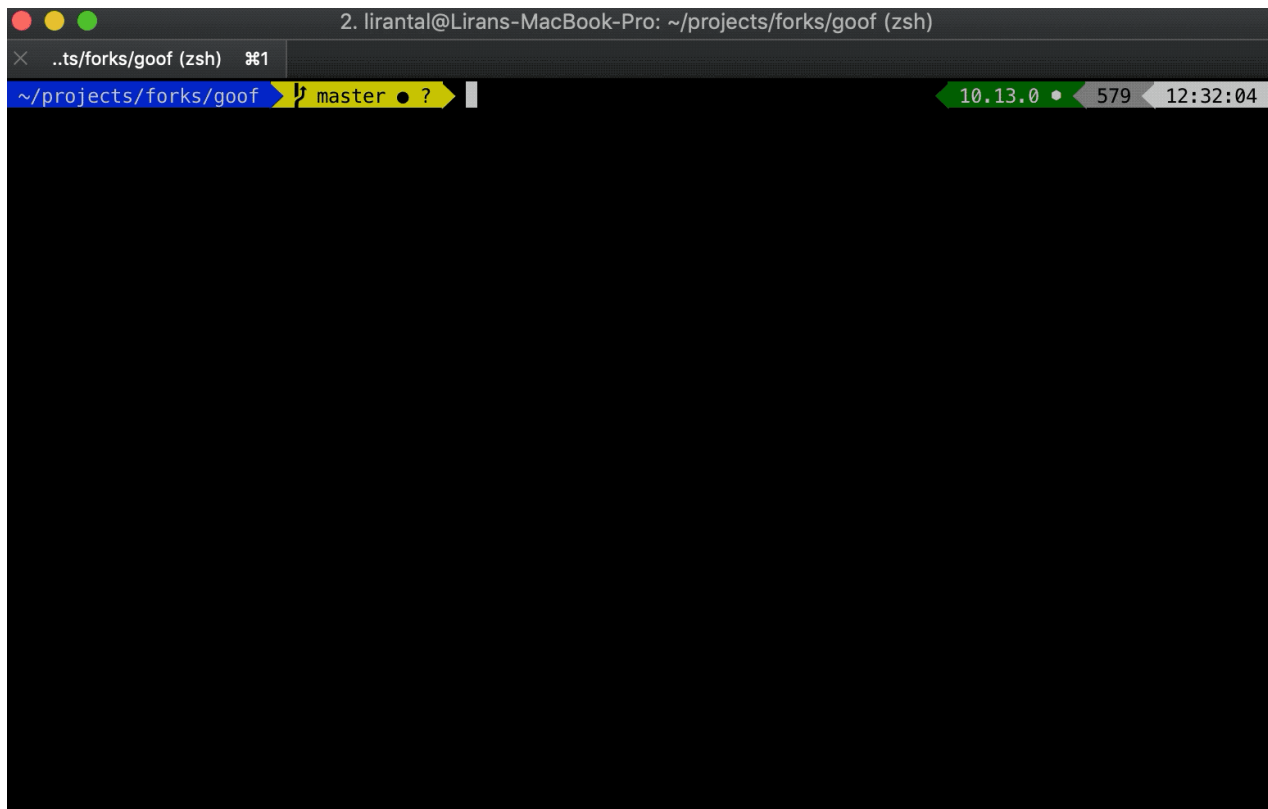
4. Find, fix and monitor for open source vulnerabilities

When we choose a base image for our Docker container, we indirectly take upon ourselves the risk of all the [container security](#) concerns that the base image is bundled with. This can be poorly configured defaults that don't contribute to the security of the operating system, as well as system libraries that are bundled with the base image we chose.

We use cookies to ensure you get the best experience on our website. [Got it](#)

being able to run your application without issues. This helps reduce the attack surface by limiting exposure to vulnerabilities; on the other hand, it doesn't run any audits on its own, nor does it protect you from future vulnerabilities that may be disclosed for the version of the base image that you are using.

Therefore, one way of protecting against vulnerabilities in [open source security software](#) is to use tools such as [Snyk](#), to add continuous docker security scanning and monitoring of vulnerabilities that may exist across all of the Docker image layers that are in use.



Scan a Docker image for known vulnerabilities with these commands:

```
# fetch the image to be tested so it exists locally
$ docker pull node:10
```


We use cookies to ensure you get the best experience on our website. [Got it](#)

```
$ snyk test --docker node:10 --file=path/to/dockerfile
```



1 Docker image for known vulnerabilities so that once newly discovered vulnerabilities are found in the image, Snyk can notify and provide remediation advice:

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

Based on scans performed by Snyk users, we found that 44% of Docker image scans had known vulnerabilities, and for which there were newer and more secure base images available. This remediation advice is unique to Snyk, based on which developers can take action and upgrade their Docker images.

Snyk also found that for 20% out of all Docker image scans, only a rebuild of the Docker image would be necessary in order to [reduce the number of vulnerabilities](#).

How to audit docker container security?

It is an essential task to scan your Linux-based container project for known vulnerabilities to ensure the security your environment. To achieve this, Snyk scans the base image for its dependencies: The operating system (OS) packages installed and managed by the package manager and key binaries—layers that were not installed through the package manager.

Use Snyk, a [free tool for container security](#). Based on the scan results, Snyk offers remediation advice and guidance for public Docker Hub images by indicating base image recommendation, Dockerfile layer in which a vulnerability was found and more.

We use cookies to ensure you get the best experience on our website. [Got it](#)



Sometimes, when building an application inside a Docker image, you need secrets such as an SSH private key to pull code from a private repository, or you need tokens to install private packages. If you copy them into the Docker intermediate container they are cached on the layer to which they were added, even if you delete them later on. These tokens and keys must be kept outside of the

`Dockerfile` .

Using multi-stage builds

Another aspect of improving docker container security is through the use of multi-stage builds. By leveraging Docker support for multi-stage builds, fetch and manage secrets in an intermediate image layer that is later disposed of so that no sensitive data reaches the image build. Use code to add secrets to said intermediate layer, such as in the following example:

```
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 .
```

Using Docker secret commands

Use an alpha feature in Docker for managing secrets to mount sensitive files

We use cookies to ensure you get the best experience on our website. [Got it](#)

```
snyk_ax = 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
```

Read more about Docker secrets on their site.

Beware of recursive copy

You should also be mindful when copying files into the image that is being built. For example, the following command copies the entire build context folder, recursively, to the Docker image, which could end up copying sensitive files as well:

```
COPY . .
```

If you have sensitive files in your folder, either remove them or use `.dockerignore` to ignore them:

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

How do you protect a docker container?

Ensure you use multi-state builds so that the container image built for production is free of development assets and any secrets or tokens.

We use cookies to ensure you get the best experience on our website. [Got it](#)

Furthermore, ensure you are using a [container security tool](#), such as Snyk which **snyk**se for free to scan your docker images from the CLI, directly from Docker Hub, or those deployed to production using Amazon ECR, Google GCR or others.

6. Use fixed tags for immutability

Each Docker image can have multiple tags, which are variants of the same images. The most common tag is *latest*, which represents the latest version of the image. Image tags are not immutable, and the author of the images can publish the same tag multiple times.

This means that the base image for your Docker file might change between builds. This could result in inconsistent behavior because of changes made to the base image. There are multiple ways to mitigate this issue and improve your Docker security posture:

We use cookies to ensure you get the best experience on our website. [Got it](#)

and `.o.o.r` or even `.o.o.r-alpine`, prefer the latter, as it is the most specific image **snyk**ice. Avoid using the most generic tags, such as `latest`. Keep in mind that when pinning a specific tag, it might be deleted eventually.

- To mitigate the issue of a specific image tag becoming unavailable and becoming a show-stopper for teams that rely on it, consider running a local mirror of this image in a registry or account that is under your own control. It's important to take into account the maintenance overhead required for this approach—because it means you need to maintain a registry. Replicating the image you want to use in a registry that you own is good practice to make sure that the image you use does not change.
- Be very specific! Instead of pulling a tag, pull an image using the specific SHA256 reference of the Docker image, which guarantees you get the same image for every pull. However notice that using a SHA256 reference can be risky, if the image changes that hash might not exist anymore.

Are Docker images secure?

Docker images might be based on open source Linux distributions, and bundle within them open source software and libraries. A recent state of open source security research conducted by Snyk found that [the top most popular docker images contain at least 30 vulnerabilities](#).

7. Use COPY instead of ADD

Docker provides two commands for copying files from the host to the Docker image when building it: `COPY` and `ADD`. The instructions are similar in nature, but differ in their functionality and can result in a Docker container security issues for the image:

- `COPY` — copies local files recursively, given explicit source and destination files or directories. With `COPY`, you must declare the locations.

We use cookies to ensure you get the best experience on our website. [Got it](#)

when it doesn't exist, and accepts archives as local or remote URLs as its source, **snyk** it expands or downloads respectively into the destination directory.

While subtle, the differences between ADD and COPY are important. Be aware of these differences to avoid potential security issues:

- When remote URLs are used to download data directly into a source location, they could result in [man-in-the-middle attacks](#) that modify the content of the file being downloaded. Moreover, the origin and authenticity of remote URLs need to be further validated. When using COPY the source for the files to be downloaded from remote URLs should be declared over a secure TLS connection and their origins need to be validated as well.
- Space and image layer considerations: using COPY allows separating the addition of an archive from remote locations and unpacking it as different layers, which optimizes the image cache. If remote files are needed, combining all of them into one RUN command that downloads, extracts, and cleans-up afterwards optimizes a single layer operation over several layers that would be required if ADD were used.
- When local archives are used, ADD automatically extracts them to the destination directory. While this may be acceptable, it adds the risk of zip bombs and [Zip Slip vulnerabilities](#) that could then be triggered automatically.

8. Use metadata labels

Image labels provide metadata for the image you're building. This helps users understand how to use the image easily. The most common label is "maintainer", which specifies the email address and the name of the person maintaining this image. Add metadata with the following `LABEL` command:

We use cookies to ensure you get the best experience on our website. [Got it](#)

snyk to a maintainer contact, add any metadata that is important to you. This metadata could contain: a commit hash, a link to the relevant build, quality status (did all tests pass?), source code, a reference to your SECURITY.TXT file location and so on.

It is good practice to adopt a [SECURITY.TXT](#) (RFC5785) file that points to your responsible disclosure policy for your Docker label schema when adding labels, such as the following:

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

See more information about labels for Docker images: <https://label-schema.org/rc1/>


9. Use multi-stage build for small and secure images

While building your application with a `Dockerfile`, many artifacts are created that are required only during build-time. These can be packages such as development tooling and libraries that are required for compiling, or dependencies that are required for running unit tests, temporary files, secrets, and so on.

Keeping these artifacts in the base image, which may be used for production, results in an increased Docker image size, and this can badly affect the time spent downloading it as well as increase the attack surface because more packages are installed as a result. The same is true for the Docker image you're using—you might need a specific Docker image for building, but not for running the code of your application.

Golang is a great example. To build a Golang application, you need the Go compiler. The compiler produces an executable that runs on any operating system,

We use cookies to ensure you get the best experience on our website. [Got it](#)

 Good reason why Docker has the multi-stage build capability. This feature allows you to use multiple temporary images in the build process, keeping only the latest image along with the information you copied into it. In this way, you have two images:

- First image—a very big image size, bundled with many dependencies that are used in order to build your app and run tests.
- Second image—a very thin image in terms of size and number of libraries, with only a copy of the artifacts required to run the app in production.

10. Use a linter

Adopt the use of a linter to avoid common mistakes and establish best practice guidelines that engineers can follow in an automated way. This is a helpful docker security scanning task to statically analyze Dockerfile security issues.

One such linter is [hadolint](#). It parses a Dockerfile and shows a warning for any errors that do not match its best practice rules.

```
/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 ➤
```

Hadolint is even more powerful when it is used inside an integrated development environment (IDE). For example, when using [hadolint as a VSCode extension](#), linting errors appear while typing. This helps in writing better Dockerfiles faster.

[Learn more about securing your docker images](#)

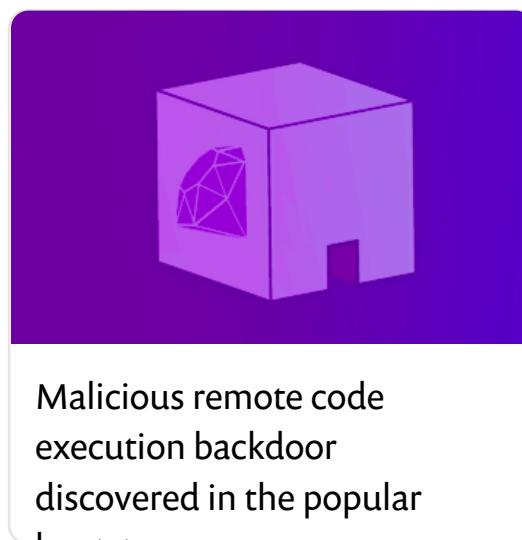
Be sure to print out the cheat sheet and pin it up somewhere to remind you of

We use cookies to ensure you get the best experience on our website. [Got it](#)
and working with docker images:

How do you harden a docker container image?

You may use linters such as hadolint or dockle to ensure the Dockerfile has secure configuration. Make sure you also scan your container images to avoid vulnerabilities with a severe security impact in your production containers. A recommended read is the following [10 Docker image security best practices](#) to ensure a secure image.

RELATED CONTENT



Ready to get started?

Snyk helps you develop fast and stay secure.

We use cookies to ensure you get the best experience on our website. [Got it](#)



[BOOK A DEMO](#)

PRODUCTCOMPANSECURITY

- Partners
- About
- JavaScript
- Developersus
- Security
- &
- Customers
- Kubernetes
- DevOps
- Jobs
- Security
- Features
- at
- Container
- Enterprise
- Snyk
- Security
- Features
- Legal
- Open
- Pricing
- Terms
- Source
- Test
- Privacy
- Security
- with
- Press
- Secure
- GitHub
- Kit
- SDLC
- Test
- Events
- with
- Secure
- CLI
- by
- API
- Design
- Status

RESOURCES

- Vulnerability DB
- Blog
- Learn
- Documentation
- Snyk API
- Research
- FAQs

FIND US ONLINE

TRACK OUR DEVELOPMENT



We use cookies to ensure you get the best experience on our website. [Got it](#)

