

A Practical Guide to Writing Secure Dockerfiles

WeAreDevelopers Live – Container Day

Madhu Akula

About Me

- Security Engineering @ Miro
- Creator of [Kubernetes Goat](#), [Hacker Container](#),
[tools.tldr.run](#), many other...
- Security (CloudNative, Containers, Kubernetes, Automation)
- Speaker & Trainer @ BlackHat, DEFCON, USENIX, OWASP, All Day DevOps, GitHub, DevSecCon, c0c0n, Nullcon, null, many other...
- Co-Author of Security Automation with Ansible 2
- Technical reviewer of Learn Kubernetes Security
- Never Ending Learner!



@madhuakula

<https://madhuakula.com>

What will you learn?

- What is Dockerfile?
- Why security of Dockerfiles?
- Best practices for writing Dockerfiles
- Linters, tools, techniques to validate
 - buildkit, dockle, hadolint, docker-slim, dive, plugins, etc.
- Introducing OPA, Rego & Conftest
- docker-security-checker
- Why custom security policies?
- Next steps
- Resources & Reference

What is Docker?

- Docker is an open source platform for building, deploying, and managing containerized applications
- Docker became the de facto standard to build and share containerized apps - from desktop, to the cloud, even edge devices
- Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere
- Docker can build images automatically by reading the instructions from a Dockerfile

What is Dockerfile?

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

```
24 lines (18 sloc) | 701 Bytes
1 # Using official python runtime base image
2 FROM python:3.9-slim
3
4 # add curl for healthcheck
5 RUN apt-get update \
6     && apt-get install -y --no-install-recommends \
7     curl \
8     && rm -rf /var/lib/apt/lists/*
9
10 # Set the application directory
11 WORKDIR /app
12
13 # Install our requirements.txt
14 COPY requirements.txt /app/requirements.txt
15 RUN pip install -r requirements.txt
16
17 # Copy our code from the current folder to /app inside the container
18 COPY . .
19
20 # Make port 80 available for links and/or publish
21 EXPOSE 80
22
23 # Define our command to be run when launching the container
24 CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-", "--access-logfile", "-", "--workers", "4", "--keep-alive", "0"]
```

Why should we focus on Dockerfile security?

- Dockerfiles are blueprint for building your docker container images
- Dockerfiles are codified version of what your application & infrastructure
- It is one of the key component for the entire supply chain security
- To maintain the highest level of security from the ground up
- Many possible security issues can happen due to insecure Dockerfiles

A screenshot of a GitHub search results page. The URL in the address bar is `github.com/search?l=&q=filename%3ADockerfile&type=code`. The search query is `filename:Dockerfile`. The results are filtered to show only code. There are 2,929,166 code results. The interface includes navigation buttons (back, forward, search), a pull requests button, an issues button, a marketplace button, and an explore button. A dropdown menu for sorting is visible.

Dockerfile best practices from Docker itself

- Start with small version of the image
- Create ephemeral containers
- Understanding the build context
- Exclude files with `.dockerignore` like `.gitignore`
- Use multi stage builds - Only we need artifacts and reduce attack surface
- Don't install unnecessary packages
- Minimize number of layers
- Create multi-line arguments in structured way and reduce the image layers
- Leverage the build cache
- Create your own base image like golden image

Generic Best Practices

- Order of the steps in Dockerfile (least to most frequently changing content)
- COPY only specific files required rather everything (don't do COPY ..)
- Perform same commands (RUN apt-get update && apt-get install -y curl)
- Only install what we need (--no-install-recommends)
- Remove package manager cache (&& rm -rf /var/lib/apt/lists/*)
- Don't use latest tag (be specific with with image tag)
- Non root user and group
- Disallow acquiring new privileges
- Only trusted and official base images
- Minimal base images, don't include not required software, packages, etc.
- Don't store secrets or sensitive information in Dockerfiles, file/directories
- Don't install SSH or similar services and expose your containers
- Image lifecycle management, updates if required
- Many others...

Generic Best Practices - Example Dockerfile

33 lines (27 sloc) 1.26 KB

Raw Blame   

```
1 # Replace latest with a pinned version tag from https://hub.docker.com/_/alpine
2 #
3 # We suggest using the major.minor tag, not major.minor.patch.
4 FROM alpine:latest
5
6 # Non-root user for security purposes.
7 #
8 # UIDs below 10,000 are a security risk, as a container breakout could result
9 # in the container being ran as a more privileged user on the host kernel with
10 # the same UID.
11 #
12 # Static GID/UID is also useful for chown'ing files outside the container where
13 # such a user does not exist.
14 RUN addgroup -g 10001 -S nonroot && adduser -u 10000 -S -G nonroot -h /home/nonroot nonroot
15
16 # Install packages here with `apk add --no-cache`, copy your binary
17 # into /sbin/, etc.
18
19 # Tini allows us to avoid several Docker edge cases, see https://github.com/krallin/tini.
20 RUN apk add --no-cache tini
21 ENTRYPOINT ["/sbin/tini", "--", "myapp"]
22 # Replace "myapp" above with your binary
23
24 # bind-tools is needed for DNS resolution to work in *some* Docker networks, but not all.
25 # This applies to nslookup, Go binaries, etc. If you want your Docker image to work even
26 # in more obscure Docker environments, use this.
27 RUN apk add --no-cache bind-tools
28
29 # Use the non-root user to run our application
30 USER nonroot
31
32 # Default arguments for your app (remove if you have none):
33 CMD ["--foo", "1", "--bar=2"]
```

BuildKit

BuildKit is a toolkit for converting source code to build artifacts in an efficient, expressive and repeatable manner.

- From release of Docker 20.10 (Stable of BuildKit)
- Now by default enabled in latest release (export DOCKER_BUILDKIT=1)
- It improves a lot of performance, security features

BuildKit has support for securely passing secrets, forwarding SSH authentication agent from the host to the Docker build.

BuildKit - Security Use Case (Secrets usage in build)

```
FROM baseimage
RUN ...
ENV AWS_ACCESS_KEY=.....  
NO
ENV AWS_SECRET_ACCESS_KEY=.....
RUN ./get-data-from-aws.sh
RUN ...
```

```
FROM baseimage
RUN ...
ARG AWS_ACCESS_KEY  
NO
ARG AWS_SECRET_ACCESS_KEY
RUN ./get-data-from-aws.sh
RUN ...
```

Passing the secrets to the Dockerfile rather hard coding or buildargs using BuildKit

```
# syntax=docker/dockerfile:1.2
FROM baseimage
RUN ...
RUN --mount=type=secret,id=aws,target=/root/.aws/credentials,required ./get-data-from-aws.sh
RUN ...
```

BuildKit - Security Use Case (SSH Socket)

```
FROM baseimage
COPY ./keys/private.pem /root/.ssh/private.pem
RUN git clone git@github.com:org/repo /app && \
    cd /app && npm run build
```



Passing the SSH socket with mount by forwarding the SSH from host using BuildKit

```
FROM baseimage
RUN apk add --no-cache openssh-client
RUN mkdir -p ~/.ssh && ssh-keyscan github.com >> ~/.ssh/known_hosts
RUN --mount=type=ssh,required \
    git clone git@github.com:org/repo /app && \
    cd /app && npm run build
```

hadolint - Haskell Dockerfile Linter

A smarter Dockerfile linter that helps you build best practice Docker images. The linter is parsing the Dockerfile into an AST and performs rules on top of the AST. It is standing on the shoulders of ShellCheck to lint the Bash code inside RUN instructions.



Dockerfile Linter

A smarter Dockerfile linter that helps you build [best practice Docker images](#). The linter is parsing the Dockerfile into an AST and performs rules on top of the AST. It additionally is using the famous [ShellCheck](#) to lint the Bash code inside RUN instructions. Please [help me improve the linter](#) with your suggestions.

```
Always tag the version of an image explicitly
1 FROM debian
node_verion is referenced but not assigned (did you mean 'node_version'?).
Delete the apt-get lists after installing something
Avoid additional packages by specifying '--no-install-recommends'
2 RUN export node_version="0.10" \
3 && apt-get update && apt-get -y install nodejs="$node_verion"
4 COPY package.json usr/src/app
Use WORKDIR to switch to a directory
Pin versions in npm. Instead of `npm install <package>` use `npm install <package>@<version>`
5 RUN cd /usr/src/app \
6 && npm install node-static
7
Valid UNIX ports range from 0 to 65535
8 EXPOSE 8000
9 CMD [ "npm", "start" ]
```

[Lint](#) [Clear](#)

hadolint in action

```
> docker run --rm -i hadolint/hadolint < Dockerfile
/dev/stdin:1 DL3007 warning: Using latest is prone to errors if the image will ever update. Pin the version explicitly to a release tag
/dev/stdin:9 DL3020 error: Use COPY instead of ADD for files and folders
/dev/stdin:11 DL3020 error: Use COPY instead of ADD for files and folders
/dev/stdin:12 DL3004 error: Do not use sudo as it leads to unpredictable behavior. Use a tool like gosu to enforce root
/dev/stdin:14 DL3008 warning: Pin versions in apt get install. Instead of `apt-get install <package>` use `apt-get install <package>=<version>`
/dev/stdin:14 DL3009 info: Delete the apt-get lists after installing something
/dev/stdin:14 DL3015 info: Avoid additional packages by specifying `--no-install-recommends`
```

dockle

```
> dockle goodwittech/dockle-test:v2
FATAL  - CIS-DI-0009: Use COPY instead of ADD in Dockerfile
      * Use COPY : /bin/sh -c #(nop) ADD file:81c0a803075715d1a6b4f75a29f8a01b21cc170cf1cff6702317d1be2fe71a3 in /app/credentials.json
FATAL  - CIS-DI-0010: Do not store credential in ENVIRONMENT vars/files
      * Suspicious filename found : app/credentials.json
      * Suspicious ENV key found : MYSQL_PASSWD
FATAL  - DKL-DI-0005: Clear apt-get caches
      * Use 'rm -rf /var/lib/apt/lists' after 'apt-get install' : /bin/sh -c apt-get update && apt-get install -y git
FATAL  - DKL-LI-0001: Avoid empty password
      * No password user found! username : nopasswd
WARN   - CIS-DI-0001: Create a user for the container
      * Last user should not be root
INFO   - CIS-DI-0005: Enable Content trust for Docker
      * export DOCKER_CONTENT_TRUST=1 before docker pull/build
INFO   - CIS-DI-0006: Add HEALTHCHECK instruction to the container image
      * not found HEALTHCHECK statement
INFO   - CIS-DI-0008: Confirm safety of setuid/setgid files
      * setuid file: bin/mount urwxr-xr-x
      * setgid file: etc/passwd grw-r--r--
      * setuid file: app/suid.txt urw-r--r--
      * setgid file: usr/bin/chage grwxr-xr-x
      * setgid file: usr/bin/wall grwxr-xr-x
      * setgid file: usr/bin/ssh-agent grwxr-xr-x
      * setgid file: app/guid.txt grw-r--r--
      * setuid file: usr/lib/openssh/ssh-keysign urwxr-xr-x
      * setuid file: bin/su urwxr-xr-x
      * setuid file: etc/shadow urw-r-----
      * setuid file: usr/bin/chsh urwxr-xr-x
      * setuid file: usr/bin/passwd urwxr-xr-x
      * setuid file: bin/umount urwxr-xr-x
      * setuid file: usr/bin/chfn urwxr-xr-x
      * setgid file: usr/bin/expiry grwxr-xr-x
      * setgid file: sbin/unix_chkpwd grwxr-xr-x
```

Container Image Linter for Security,
Helping build the Best-Practice
Docker Image, Easy to start

dockle - CIS Benchmark Checks

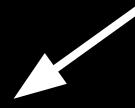
CIS Benchmark	Dockle	Docker Bench for Security	Hadolint	Clair
1. Create a user for the container	✓	✓	✓	-
2. Use trusted base images for containers	▲	▲	-	-
3. Do not install unnecessary packages in the container	-	-	-	-
4. Scan and rebuild the images to include security patches	-	-	-	✓
5. Enable Content trust for Docker	✓	✓	-	-
6. Add HEALTHCHECK instruction to the container image	✓	✓	-	-
7. Do not use update instructions alone in the Dockerfile	✓	✓	✓	-
8. Remove setuid and setgid permissions in the images	✓	-	-	-
9. Use COPY instead of ADD in Dockerfile	✓	✓	✓	-
10. Do not store secrets in Dockerfiles	✓	-	-	-
11. Install verified packages only	-	-	-	-

dockle - Checks

Original Checkpoints	Dockle	Docker Bench for Security	Hadolint	Clair
1. Avoid null password user (CVE-2019-5021)	✓	-	-	-
2. Be unique UID/GROUPs	✓	-	-	-
3. Avoid <code>sudo</code> command	✓	-	✓	-
4. Avoid sensitive directory mounting	✓	-	-	-
5. Avoid <code>apt-get upgrade</code> , <code>apk upgrade</code> , <code>dist-upgrade</code>	✓	-	✓	-
6. Use <code>apk add</code> with <code>--no-cache</code>	✓	-	✓	-
7. Clear <code>apt-get</code> caches	✓	-	✓	-
8. Avoid <code>latest</code> tag	✓	-	✓	-

DockerSlim - Minify and Secure Docker Containers

```
$ head -n 1 Dockerfile
FROM ubuntu:14.04

$ docker images my/sample-node-app
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
my/sample-node-app  latest   31be09316a19  4 minutes ago  432MB 
```

DockerSlim - Usage and Features

```
> docker-slim --help
Incorrect Usage. flag: help requested

NAME:
  docker-slim - optimize and secure your Docker containers!

USAGE:
  docker-slim [global options] command [command options] [arguments...]

VERSION:
  darwin|Transformer|1.34.0|a5cb54043b3ab3cf747165aad745f19db680434e|2021-01-29_10:00:49PM

COMMANDS:
  build, b      Analyzes, profiles and optimizes your container image auto-generating Seccomp and AppArmor security profiles
  containerize, c Containerize the target artifacts
  convert, k    Convert container image
  edit, e       Edit container image
  help, h       Show help info
  lint, l       Analyzes container instructions in Dockerfiles
  probe, prb   Probe target
  profile, p   Collects fat image information and generates a fat container report
  run, r       Run one or more containers
  server, s    Run as an HTTP server
  update, u   Updates docker-slim
  version, v   Shows docker-slim and docker version information
  xray, x     Shows what's inside of your container image and reverse engineers its Dockerfile
```

DockerSlim - Security Profiles

- The goal is to auto-generate Seccomp, AppArmor, (and potentially SELinux) profiles based on the collected information
 - AppArmor Profiles
 - Seccomp Profiles
- Generating the Seccomp profiles (I have tried it, doesn't work for all cases)
 - Run DockerSlim
 - docker-slim build your-name/your-app
 - Use the generated Seccomp profile
 - docker run --security-opt seccomp:<docker-slim directory>/images/<YOUR_APP_IMAGE_ID>/artifacts/your-name-your-app-seccomp.json <your other run params> your-name/your-app

dive

A tool for exploring
a docker image,
layer contents, and
discovering ways to
shrink the size of
your Docker/OCI
image.

Your code editor linters

The screenshot shows the Visual Studio Marketplace page for the "Docker Linter" extension. At the top, there's a navigation bar with "Visual Studio | Marketplace" and a search bar. Below the header, the URL "Visual Studio Code > Linters > Docker Linter" is visible, along with a "Sign in" button and a "New to Visual Studio Code? Get it now." link.

The main content area features a large blue header with the extension's name "Docker Linter" and a circular icon depicting a ship on water. Below the header, the developer "Henrik Sjööh" is listed, showing 118,076 installs and a 5-star rating. A note states: "Lint perl, python and/or ruby in your docker containers."

Key buttons include a green "Install" button and a "Trouble Installing?" link. Below these, a navigation menu offers links to "Overview", "Version History", "Q & A", and "Rating & Review".

The "Functionality" section lists supported languages and tools:

- perl
 - perl
 - perl -c
 - perlcritic
- PHP
 - php -l
- python
 - flake8
- ruby
 - rubocop

The "Categories" section includes a single category "Linters". The "Tags" section lists "linters", "perl", "perl+mojo", "perl+mojolicious", "php", "python", and "ruby".

The "Resources" section provides links to "Issues", "Repository", "Homepage", "Changelog", and "Download Extension".

The "Project Details" section shows the GitHub repository "henriik/vscode-docker-linter", noting "No Pull Requests", "Last Commit: 3 years ago", and "1 Open Issues".

<https://marketplace.visualstudio.com/items?itemName=henriik.docker-linter>

Introducing OPA

Open Policy Agent is a Policy-based control for cloud native environments.

OPA is an open source, general-purpose policy engine that unifies policy enforcement across the stack. It provides a high-level declarative language that lets you specify policy as code and simple APIs to offload policy decision-making from your software.



Introduction to rego policies

- OPA policies are expressed in a high-level declarative language called Rego. Rego (pronounced “ray-go”) is purpose-built for expressing policies over complex hierarchical data structures
- Rego was inspired by Datalog, which is a well understood, decades old query language. Rego extends Datalog to support structured document models such as JSON
- Rego queries are assertions on data stored in OPA. These queries can be used to define policies that enumerate instances of data that violate the expected state of the system

Sample Rego policy example

The screenshot shows the The Rego Playground interface with the following components:

- Top Bar:** Includes the logo, "The Rego Playground", "Examples ▾", "Coverage" (unchecked), "Evaluate", and "Publish".
- Left Panel (Code Editor):** Displays the Rego policy code. The code implements Role-based Access Control (RBAC) with grants and role mappings.
- Input Panel:** Shows the input data for evaluation, including a user action and a data object containing user roles and role grants.
- Output Panel:** Shows the evaluation results, indicating 1 result found in 148.408 μs, with the output JSON being:

```
1 {
2   "allow": true,
3   "user_is_admin": true,
4   "user_is_granted": []
5 }
```

What is conftest?

Conftest is a utility to help you write tests against structured configuration data. For instance you could write tests for your Kubernetes configurations, Terraform code, Serverless configs or any other structured data. In our context, we will use it to write validation policies for Dockerfiles.

Conftest relies on the Rego language from Open Policy Agent for writing the assertions.

Supported formats by Conftest

- YAML
- JSON
- INI
- TOML
- HOCON
- HCL
- HCL2
- CUE
- Dockerfile
- EDN
- VCL
- XML
- Jsonnet

```
package main

deny[msg] {
    input.kind == "Deployment"
    not input.spec.template.spec.securityContext.runAsNonRoot

    msg := "Containers must not run as root"
}

deny[msg] {
    input.kind == "Deployment"
    not input.spec.selector.matchLabels.app

    msg := "Containers must provide app label for pod selectors"
}
```

```
$ conftest test deployment.yaml
FAIL - deployment.yaml - Containers must not run as root
FAIL - deployment.yaml - Containers must provide app label for pod selectors

2 tests, 0 passed, 0 warnings, 2 failures, 0 exceptions
```

Sample Dockerfile

```
16 lines (11 sloc) | 319 Bytes

1 FROM ubuntu:latest
2 LABEL MAINTAINER "Madhu Akula"
3
4 ENV SECRET AKIGG23244GN2344GHG
5 ENV GITLAB_API_ID gig32oig3bgi34gb43gb43uigb43i
6
7 WORKDIR /app
8
9 ADD app /app
10 COPY README.md /app/README.md
11 ADD code /tmp/code
12 RUN sudo apt-get update
13
14 RUN apt-get update && apt-get install -y htop
15
16 CMD ["/bin/bash", "/app/entrypoint.sh"]
```

Policy for checking ADD usage instead of COPY

Here in the below example input contains the Dockerfile in a JSON format and we are looking for any command we find ADD and if we find in the Dockerfile we are returning the deny message.

```
warn[msg] {
    input[i].Cmd == "add"
    val := concat(" ", input[i].Value)
    msg = sprintf("Use COPY instead of ADD: %s", [val])
}
```

Running docker-security-checker

```
› conftest test Dockerfile

WARN - Dockerfile - main - Do not use latest tag with image: ["ubuntu:latest"]
FAIL - Dockerfile - main - Suspicious ENV key found: ["SECRET", "AKIGG23244GN2344GHG"]
FAIL - Dockerfile - main - Use COPY instead of ADD: app /app
FAIL - Dockerfile - main - Use COPY instead of ADD: code /tmp/code
FAIL - Dockerfile - main - Avoid using 'sudo' command: sudo apt-get update

6 tests, 1 passed, 1 warning, 4 failures, 0 exceptions
```

Why custom policies?

- Most organisations have common patterns across their workflows
- Some policies can be custom to their organisation, for example the below policy showcase that any base image not from exampletrustedregistry.com not allowed
 - Using these simple but powerful policies can easily prevent using untrusted images directly from the public registries and only allow developers and users to use internal private registries

```
deny[msg] {  
    input[i].Cmd == "from"  
    image := input[i].Value  
    not startswith(image, "exampletrustedregistry.com/")  
    msg := sprintf("Base image '%v' is used from untrusted registry", [image])  
}
```

Try it out yourself

The screenshot shows the Katacoda platform interface for a "docker-security-checker" scenario. On the left, there's a sidebar with navigation links like "Katacoda Overview & Solutions", "Search", "Your Profile", and "Log Out". The main content area has a title "What is docker-security-checker?" and a sub-section "What is Conftest?". It also includes a "CONTINUE" button at the bottom.

The right side of the screen displays a terminal window showing the execution of a Dockerfile and a security check script. The terminal output includes:

```
Terminal Authoring Information +  
Cloning into 'docker-security-checker'...  
cd docker-security-checker  
remote: Enumerating objects: 9, done.  
remote: Counting objects: 100% (9/9), done.  
remote: Compressing objects: 100% (8/8), done.  
remote: Total 9 (delta 1), reused 5 (delta 0), pack-reused 0  
Unpacking objects: 100% (9/9), done.  
Checking connectivity... done.  
$ cd docker-security-checker  
$ conftest test Dockerfile  
WARN - Dockerfile - Do not use latest tag with image: ["ubuntu:latest"]  
FAIL - Dockerfile - Suspicious ENV key found: ["SECRET", "AKIGG23244GN2344GHG"]  
FAIL - Dockerfile - Use COPY instead of ADD: app /app  
FAIL - Dockerfile - Use COPY instead of ADD: code /tmp/code  
  
-----  
PASS: 0/4  
WARN: 1/4  
FAIL: 3/4  
$
```

Below the terminal, a summary table provides a breakdown of the results:

Category	Count
PASS	0/4
WARN	1/4
FAIL	3/4

<https://katacoda.com/madhuakula/scenarios/docker-security-checker>

Next steps

- Try possible following the best practices when writing itself by using linters
- You can also use these check in your GitOps workflow, part of your git hooks
- You can create and standardise organisation wide custom policies as required for your workflow
- Adding these checks in CI/CD pipelines as part of your workflow will enable and validate security best practices
- Go beyond Dockerfiles and implement at each layer of your workflows

Are you interested to learn more tools around security?

The screenshot shows a dark-themed web page titled "Security Tools!" with a subtitle "Curated list of security tools for Hackers & Builders!". At the top right, there's an "Incognito" button. Below the title is a "Submit your awesome tools" button with a thumbs-up icon. To its right is a Twitter follow button for "@tldr.run" with 203 followers, and social sharing icons for Facebook, Twitter, LinkedIn, GitHub, YouTube, and Telegram. A search bar at the bottom contains the text "kubernetes".

Below the header, there are five filter buttons: Windows, Linux, Mac OS X, Open Source, Free, and Commercial.

The main content area displays a grid of tool cards:

- k-rail**: A workload policy enforcement tool for Kubernetes. It can help you secure a multi-tenant cluster with minimal disruption and maximum velocity.
- DAYTONA**: A vault client, but for servers and containers. This is intended to be a lighter, alternative, implementation of the Vault client primarily for services and containers. Its core features are the ability to automate authentication, fetching of secrets, and automated token renewal.
- Dockle**: Container Image Linter for Security, Helping build the Best-Practice Docker Image, Easy to start.
- kubectl**: A command line tool that implements kubelet's API. Part of kubectl's API is documented but most of it is not. This tool covers all the documented
- KubiScan**: Helps cluster administrators identify permissions that attackers could potentially exploit to compromise the clusters. This can be especially
- KUBESEC.IO**: Security risk analysis for Kubernetes resources

On the left side, there is a "Tags List" sidebar with a "Search Tags" input field and a list of tags categorized into sections like AWS, Cloud Security, Python, Provisioning, HashiCorp, Inventory, SQL, Vault, Kubernetes, and more.

<https://tools.tldr.run>

References & Resources

- <https://docs.docker.com/engine/reference/builder>
- [Dockerfile Best Practices talk at dockercon 19](#)
- https://docs.docker.com/develop/develop-images/dockerfile_best-practices
- <https://github.com/hexops/dockerfile>
- <https://engineering.bitnami.com/articles/best-practices-writing-a-dockerfile.html>
- <https://snyk.io/blog/10-docker-image-security-best-practices>
- <https://pythonspeed.com/docker/>
- <https://github.com/moby/buildkit>
- <https://github.com/goodwithtech/dockle>
- <https://github.com/hadolint/hadolint>
- <https://github.com/docker-slim/docker-slim>
- <https://www.openpolicyagent.org>
- <https://play.openpolicyagent.org>
- <https://www.conftest.dev>
- <https://github.com/madhuakula/docker-security-checker>

Thank You!

Madhu Akula

<https://madhuakula.com>