Home > DevOps

Deep dive in Docker: Tips and Tricks to build Docker images optimized for security and size.

3 January 2022 - 7 min. read

Amazon’s Elastic Container Registry *Containers* *Docker*

A brief introduction

Nowadays we have a lot of workloads that are built on top of Docker and it is crucial to keep our Docker images secure and optimized for the cloud. We already covered Docker in our blog, you can dig further on the origins of Docker or see a use case with Wordpress.

In the next paragraphs, we will see a few tips on how we can avoid the most common security vulnerabilities and how to optimize the size of our Docker images.

But first, we need to understand one of the core concepts of Docker: the layering system.

The Dockerfile is the main file that defines how a Docker image is built. It defines the base image from which we start, the application’s dependencies, how the application is packaged and how it is run.

When we start the build process from a Dockerfile, the Docker engine creates a series of layers, one for each command in the Dockerfile. Every time we execute a command in the Dockerfile, a new layer containing the changes is created on top of the previous one. The Docker engine executes the commands sequentially and eventually combines all the layers to create the final image.

How can I keep my Docker image small?

Start from a small base image. Alpine is a good choice

Keeping the size small generally means that it is faster to build, deploy and run. To achieve that, a good idea is to start from a base image that is itself very small. You can use `alpine:3.15` which is just about 5.6 MB instead of `ubuntu:20.04` which is (at the time of this article) 72 MB.

Alpine is a general-purpose Linux distribution built around musl libc and BusyBox that aims for simplicity, security, and resource efficiency. It contains only absolutely necessary packages because you know… *if it's not included it can't break*.

Keep in mind that it also has some drawbacks. For example, since Alpine is based on the musl C library, instead of the most widely used GNU C Library, you may encounter problems with some C-based dependencies.

Use multi-stage Dockerfiles

Aside from using Alpine as a base image, another method for reducing the size of your images is using multistage builds. A multistage build consists of a single Dockerfile in which we define multiple FROM instructions and each one defines a new stage.

In each FROM we can start from a previous stage in order to take advantage of cached build artifacts or we can start from an entirely new base image and just copy some artifacts from the previous stage.

Let's see a quick example:

```
# Base image with dependencies
FROM node:17.3.0-alpine3.12 AS base
WORKDIR /app
# Copy package.json and package-lock.json
COPY package*.json ./
# Install dependencies
RUN npm install



# Build Stage
```

```
FROM base AS build
WORKDIR /app
COPY . ./
# Build and bundle static files
RUN npm run build


# Release Stage
FROM node:17.3.0-alpine3.12 AS release
WORKDIR /app
COPY --from=base /app/package.json ./
# Install app dependencies
RUN npm install --only=production
COPY --from=build /app/dist/ ./
CMD [ "npm", "run", "start" ]
```

In this multi-stage Dockerfile we created a base image that has all the required dependencies, we then created a builder stage, and finally, for the actual runtime image, we started from a clean image, installed the production requirements, and selectively copied only the required artifacts.

By default, all stages are built but we can choose to build a specific stage with the flag `--target <stage-name>`

`docker build --target release --tag image .`

You can refer to the official docs for further details.

Make the most out of caching

To fully take advantage of layer caching, write your Dockerfiles in a way that commands that will not change often are executed before others. That way the layers that are created first will be cached and reused. Although this will not change the final image size, it will make the builds faster.

Use .dockerignore

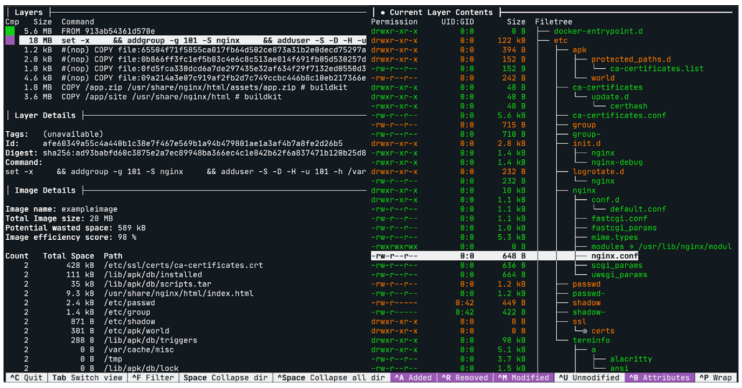
Add temporary files, intermediate build artifacts, local development files and sensitive info to .dockerignore exclude them. Next time you start a build it will be faster because the build context of the Docker engine will be smaller.

Btw you should not commit sensitive info in your repository :)

Keep an eye on the layers – Dive

Dive is a CLI tool that allows you to inspect the contents of each layer in a Docker Image. It can be very useful to see what has changed after each step in the dockerfile, because that way you can see if there are redundancies or unnecessary files in the image.

To see it in action just run `dive <image-name>`



```
│ Layers ├────────────────────────────────────────────────────────────────────── │ ● Current Layer Contents ├──────────────────────────────────
Cmp   Size  Command                                                             Permission     UID:GID       Size  Filetree
🟩  5.6 MB  FROM 913ab54361d570e                                                drwxr-xr-x         0:0        0 B  ├── docker-entrypoint.d
🟪   18 MB  set -x     && addgroup -g 101 -S nginx     && adduser -S -D -H -u   drwxr-xr-x         0:0     122 kB  ├── etc
    1.2 kB  #(nop) COPY file:65504f71f5855ca017fb64d502ce873a31b2e0decd75297a   drwxr-xr-x         0:0      394 B  │   ├── apk
    2.0 kB  #(nop) COPY file:0b866ff3fc1ef5b03c4e6c8c513ae014f691fb05d530257d   drwxr-xr-x         0:0      152 B  │   │   ├── protected_paths.d
    1.0 kB  #(nop) COPY file:0fd5fca330dcd6a7de297435e32af634f29f7132ed0550d3   -rw-r--r--         0:0      152 B  │   │   │   └── ca-certificates.list
    4.6 kB  #(nop) COPY file:09a214a3e07c919af2fb2d7c749ccbc446b8c10eb217366e   -rw-r--r--         0:0      242 B  │   │   └── world
    1.8 MB  COPY /app.zip /usr/share/nginx/html/assets/app.zip # buildkit       drwxr-xr-x         0:0       48 B  │   ├── ca-certificates
    3.6 MB  COPY /app/site /usr/share/nginx/html # buildkit                     drwxr-xr-x         0:0       48 B  │   │   └── update.d
                                                                                -rwxr-xr-x         0:0       48 B  │   │       └── certhash
│ Layer Details ├─────────────────────────────────────────────────────────────  -rw-r--r--         0:0     5.6 kB  │   ├── ca-certificates.conf
                                                                                -rw-r--r--         0:0      715 B  │   ├── group
Tags:   (unavailable)                                                           -rw-r--r--         0:0      710 B  │   ├── group-
Id:     afe60349a55c4a440b1c38e7f467e569b1a94b479801ae1a3af4b7a8fe2d26b5        drwxr-xr-x         0:0     2.8 kB  │   ├── init.d
Digest: sha256:ad93babfd60c3875e2a7ec89948ba366ec4c1e842b62f6a837471b120b25d8   -rwxr-xr-x         0:0     1.4 kB  │   │   ├── nginx
Command:                                                                        -rwxr-xr-x         0:0     1.4 kB  │   │   └── nginx-debug
set -x     && addgroup -g 101 -S nginx     && adduser -S -D -H -u 101 -h /var   drwxr-xr-x         0:0      232 B  │   ├── logrotate.d
                                                                                -rw-r--r--         0:0      232 B  │   │   └── nginx
│ Image Details ├─────────────────────────────────────────────────────────────  drwxr-xr-x         0:0      10 kB  │   ├── nginx
                                                                                drwxr-xr-x         0:0     1.1 kB  │   │   ├── conf.d
Image name: exampleimage                                                        -rw-r--r--         0:0     1.1 kB  │   │   │   └── default.conf
Total Image size: 28 MB                                                         -rw-r--r--         0:0     1.1 kB  │   │   ├── fastcgi.conf
Potential wasted space: 589 kB                                                  -rw-r--r--         0:0     1.0 kB  │   │   ├── fastcgi_params
Image efficiency score: 98 %                                                    -rw-r--r--         0:0     5.3 kB  │   │   ├── mime.types
                                                                                -rwxrwxrwx         0:0        0 B  │   │   ├── modules → /usr/lib/nginx/modul
Count   Total Space  Path                                                       -rw-r--r--         0:0      648 B  │   │   ├── nginx.conf
    2        428 kB  /etc/ssl/certs/ca-certificates.crt                         -rw-r--r--         0:0      636 B  │   │   ├── scgi_params
    2        111 kB  /lib/apk/db/installed                                      -rw-r--r--         0:0      664 B  │   │   └── uwsgi_params
    2         35 kB  /lib/apk/db/scripts.tar                                    -rw-r--r--         0:0     1.2 kB  │   ├── passwd
    2        9.3 kB  /usr/share/nginx/html/index.html                           -rw-r--r--         0:0     1.2 kB  │   ├── passwd-
    2        2.4 kB  /etc/passwd                                                -rw-r-----        0:42      449 B  │   ├── shadow
    2        1.4 kB  /etc/group                                                 -rw-r-----        0:42      422 B  │   ├── shadow-
    2         871 B  /etc/shadow                                                drwxr-xr-x         0:0        0 B  │   ├── ssl
    2         301 B  /etc/apk/world                                             drwxr-xr-x         0:0        0 B  │   │   └─⊕ certs
    2         288 B  /lib/apk/db/triggers                                       drwxr-xr-x         0:0      98 kB  │   └── terminfo
    2           0 B  /var/cache/misc                                            drwxr-xr-x         0:0     5.1 kB  │       ├── a
    2           0 B  /tmp                                                       -rw-r--r--         0:0     3.7 kB  │       │   ├── alacritty
    2           0 B  /lib/apk/db/lock                                           -rw-r--r--         0:0     1.5 kB  │       │   └── ansi
 ^C Quit │ Tab Switch view │ ^F Filter │ Space Collapse dir │ ^Space Collapse all dir │ ^A Added │ ^R Removed │ ^M Modified │ ^U Unmodified │ ^B Attributes │ ^P Wrap
```

The main features of Dive are:

Show Docker image contents in each layer: It provides you a nice tree view of all the contents of the image in a specific layer of the image.

Filter by new, modified, and removed files only: with the handy shortcuts you can enable/disable the view of specific files.

Estimate image efficiency: An experimental feature that shows you how much space is being wasted.

Bonus tip for automation

You can set the env variable `CI=true` to disable the interactive view and integrate it in a CI/CD pipeline.

You can create a .dive-ci file to customize three parameters for tests acceptance:

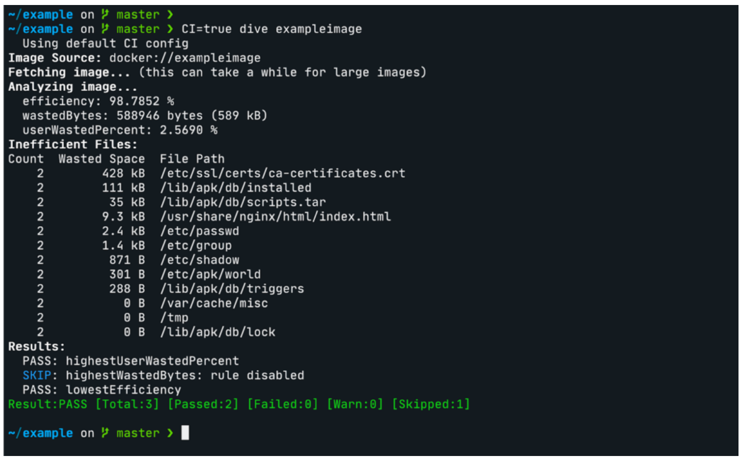
`lowestEfficiency`, `highestWastedBytes` and `highestUserWastedPercent`.

.dive-ci file:

```
rules:
 # Lowest allowed efficiency percentage (value range between 0-1).
 lowestEfficiency: 0.95

 # If the amount of wasted space is at least X or larger than X, mark
as failed.
 # Expressed in B, KB, MB, and GB.
 highestWastedBytes: 20MB

 # If the amount of wasted space makes up for X% or more of the imag
e, mark as failed.
 # Note: the base image layer is NOT included in the total image siz
e.
 # Expressed as a ratio between 0-1; fails if the threshold is met or
crossed.
 highestUserWastedPercent: 0.10
```



```
~/example on ᚠ master ❯
~/example on ᚠ master ❯ CI=true dive exampleimage
  Using default CI config
Image Source: docker://exampleimage
Fetching image... (this can take a while for large images)
Analyzing image...
  efficiency: 98.7852 %
  wastedBytes: 588946 bytes (589 kB)
  userWastedPercent: 2.5690 %
Inefficient Files:
Count  Wasted Space  File Path
    2        428 kB  /etc/ssl/certs/ca-certificates.crt
    2        111 kB  /lib/apk/db/installed
    2         35 kB  /lib/apk/db/scripts.tar
    2        9.3 kB  /usr/share/nginx/html/index.html
    2        2.4 kB  /etc/passwd
    2        1.4 kB  /etc/group
    2         871 B  /etc/shadow
    2         301 B  /etc/apk/world
    2         288 B  /lib/apk/db/triggers
    2           0 B  /var/cache/misc
    2           0 B  /tmp
    2           0 B  /lib/apk/db/lock
Results:
  PASS: highestUserWastedPercent
  SKIP: highestWastedBytes: rule disabled
  PASS: lowestEfficiency
Result:PASS [Total:3] [Passed:2] [Failed:0] [Warn:0] [Skipped:1]

~/example on ᚠ master ❯ █
```

Security basics

A basic rule of thumb is to **start from the latest stable version** of the base image. That way you will have the latest security patches for the native tools and libraries. The same rule also applies to the application-level dependencies.

Another basic rule is to **reduce the attack surface**: you should have only the minimum set of tools and libraries required to run the application. To achieve this you can start from a very small base image (something like Alpine) and then install the application-level dependencies on top of that.

One other good practice to follow is leveraging **vulnerability scanning tools** to check for known vulnerabilities in your Docker Images. Amazon's Elastic Container Registry allows us to scan our images at no additional cost!

AWS ECR Scan

Amazon's Elastic Container Registry (ECR) is a fully managed container registry that, among other features, offers a way to scan your images for vulnerabilities.

Two scanning options are available

Basic scanning

Allows manual and automatic scanning of images and is a free service. It uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source Clair project.

Enhanced scanning

Amazon ECR integrates with Amazon Inspector to provide automated scanning of the repositories. Your container images are continuously scanned and when new vulnerabilities appear Inspector notifies you through EventBridge.

You can scan an image manually through the AWS Console or you can use the following AWS CLI command:

```
aws ecr start-image-scan --repository-name nginx-test --image-id imag
eTag=base
{
        "registryId": "424242424242",
        "repositoryName": "nginx-test",
        "imageId": {
        "imageDigest": "sha256:61191087790c31e43eb37caa10de1135b002f1
0c09fdda7fa8a5989db74033aa",
        "imageTag": "base"
        },
        "imageScanStatus": {
        "status": "IN_PROGRESS"
        }
}
```

and then use the following command to check the status of the scan:

```
aws ecr describe-image-scan-findings --repository-name nginx-test --i
mage-id imageTag=base
{
        "imageScanFindings": {
        "findings": [...],
        "imageScanCompletedAt": "2021-12-22T11:44:55+01:00",
        "vulnerabilitySourceUpdatedAt": "2021-12-22T01:26:43+01:00",
        "findingSeverityCounts": {
                "HIGH": 8,
                "MEDIUM": 41,
                "LOW": 23,
                "UNDEFINED": 1,
                "INFORMATIONAL": 67
        }
        },
        "registryId": "4242424242424242",
        "repositoryName": "nginx-test",
        "imageId": {
        "imageDigest": "sha256:61191087790c31e43eb37caa10de1135b002f1
0c09fdda7fa8a5989db74033aa",
        "imageTag": "base"
        },
        "imageScanStatus": {
        "status": "COMPLETE",
        "description": "The scan was completed successfully."
        }
}
```

Scan on push

You can enable the ECR `ScanOnPush` feature on a specific repository or enable it on all your repositories by setting the Basic scanning configuration in the Console `Amazon ECR > Private registry > Scanning configuration`

By setting this feature you can also benefit from the AWS EventBridge integration. Since every scan emits an event, you can intercept them with Event Bridge and build for example an event-driven notification system to alert you in case of critical security issues in your images.

To Conclude

In this article, we covered some practices that you can adopt to optimize your Docker build process in multiple aspects.

In regard to image size and build duration we saw some Dockerfile tips & tricks, we learned how to use Dive to inspect the layers and how to integrate it in a pipeline. And finally we saw how you can leverage ECR scan to automate scanning of your Images.

If you're still curious about the topic, here are some useful links:

Alpine Linux

AWS ECR

Docker Multi Stage Build

Clair Project

Dive

Contact us or write in comments to share your Docker experience! And see you in a few days for a new article on Proud2beCloud!

Mehmed Dourmouch

DevOps Engineer. Very Dev, not so Ops. I like to break things and see what happens, I also automate everything. I often participate in cybersecurity CTFs and in my free time I produce cacophony with my guitar.
