accesto

Docker Deep Dive

Optimization techniques
for better Docker containers

0

Why you should read this book?

There are many reasons why you should. Just to name a few: cutting build times by half to save lots of time every day, saving hundreds of megabytes of disk space, improving project security, mastering Docker, or just paying attention to the quality of Docker images and runtimes you create.

Still not interested? By using the described rules, I was able to optimize an image from over 3GB to 800 MB in the development version, and cut the build time by 4x.
I hope you are interested now :)

Table of contents

Dockerfile instructions order

Often overlooked, yet very powerful - optimizing the order of instructions in your Dockerfile can have a tremendous impact on its build performance. The reason for this is how Docker layers work

Because each step of the build process is a new layer, nd each layer has a checksum that depends on the files in that layer, ordering the instructions properly will introduce huge benefits.

Let's take a look at the following Dockerfile:

```
FROM node:15
RUN  npm install -g http-server
WORKDIR /app
COPY . /app/
RUN  npm install
RUN  npm run build
EXPOSE 8080
CMD [ “http-server”, “dist” ]
```

Quite straightforward - just a simple Dockerfile for a small Vue application. If we modify some files, we can easily re-build it to publish a new version:

```
docker build -t instruction_order:pre -f Dockerfile.pre
```

In case of my Macbook Pro, it took 51s in total to rebuild

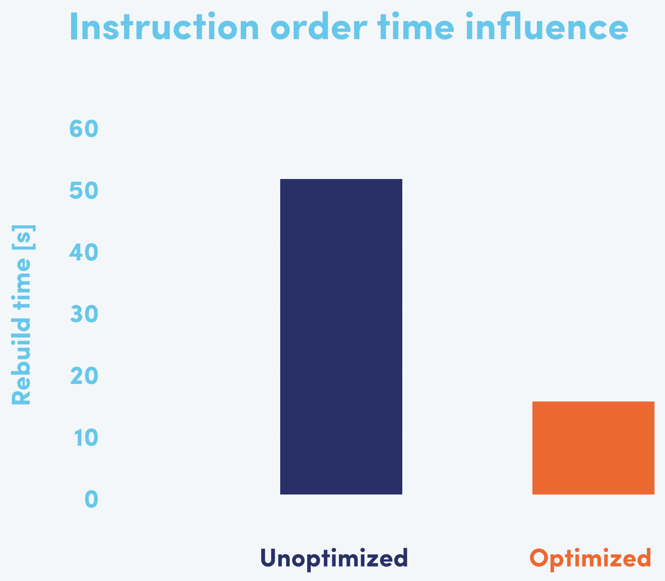
Such a Dockerfile might look familiar, in fact many of the projects I worked on had similar Dockerfiles. Yet, this is utterly suboptimal. Why? Because it won't make use of cache for the installation step, and as you might know – npm install takes a while.

The result of npm install depends on two files: package.json and package-lock.json. Only changes in these two files will require running a npm install. If you fix a typo in an html file, why would you run the npm install again? You wouldn't, but in the case of the above Dockerfile, you do. That's because the small html file change also changes the layer checksum at `line 6`, thus making the build process longer than it should be.

A fixed Dockerfile would look like this:

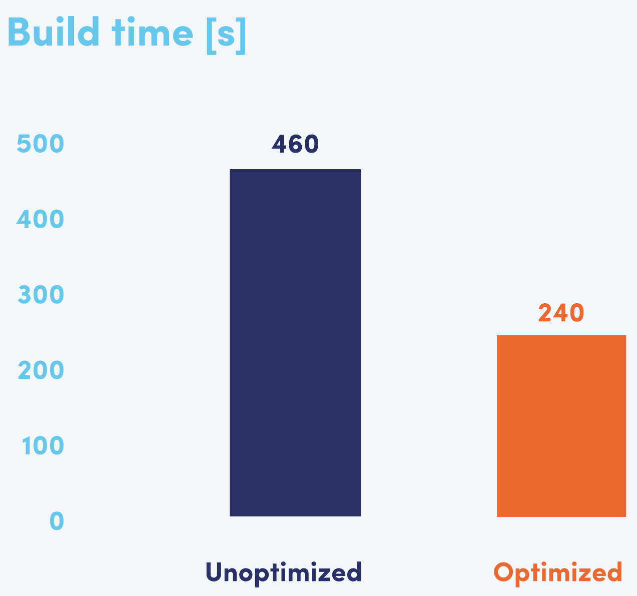
```
FROM node:15
RUN  npm install -g http-server
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
EXPOSE 8080
CMD [ “http-server”, “dist” ]
```

See the difference? It’s very subtle, and easy to implement. But what's the difference in build time? Have a look:



So, in introducing this simple two-line change, we reduced **from 51s to only 15s**. That's almost **3,5 times faster**!

Now in regards to the projects I work on, there are also backend dependencies to be installed. It's accomplished by running Composer. The build time difference for both npm and composer is as follows:



So using this simple trick, I was able to optimize the build time by almost 50%, and save more than 3 minutes on each build!