The worst so-called “best practice” for Docker

by Itamar Turner-Trauring
Last updated 01 Oct 2021, originally created 23 Mar 2021

Somebody is always wrong on the Internet, and bad Docker packaging advice is quite common. But one particular piece of advice keeps coming up, and it’s dangerous enough to merit its own article.

In a whole bunch of places you will be told not to install security updates when building your Docker image. I’ve been submitting PRs to fix this, so it’s up in fewer places now.

But previously this advice was given by the official Docker docs’ best practices page:

> Avoid `RUN apt-get upgrade`…, as many of the “essential” packages from the parent images cannot upgrade inside an unprivileged container.

To be clear: `RUN` commands happen during image build, not during container startup. The documentation there is saying “build your images without installing security updates”.

And you’ll see the same advice in the `hadolint Dockerfile` linter (it cites the above). This should also be fixed soon.

For the vast majority of people creating `Dockerfile`s this is absolutely awful advice. And since this bad advice is so common, let’s consider some of the justifications and why they are wrong.

> **Note:** An earlier version of this post also mentioned OWASP, but I opened a PR and they have since removed the bad advice.

Bad argument #1: You can’t upgrade inside an unprivileged container

As we saw above, Docker’s own documentation until very recently suggested that you not install security updates because you “cannot upgrade inside an unprivileged container.”

In order to install security updates, you need to be running as root or some other privileged user. And it’s true, you don’t want your Docker image to run as root.

But just because you’re installing security updates doesn’t mean your image needs to run as root. Behold the not-so-secret, quite obvious solution—first you install security updates, then you switch to another user:

```
FROM debian:buster
# Runs as root:
RUN apt-get update && apt-get -y upgrade

# Switch to non-root user:
RUN useradd --create-home appuser
WORKDIR /home/appuser
USER appuser

# Runs as non-root user:
ENTRYPOINT ["whoami"]
```

What if your base image already changes to a non-root user? That’s still not a problem, you can switch back and forth between different users throughout your `Dockerfile`. So you can switch to root, install security updates, and then switch back to the non-root user.

Just to demonstrate:

```
FROM debian:buster

# Switch to non-root user:
RUN useradd --create-home appuser
WORKDIR /home/appuser
USER appuser
RUN whoami

# Switch back to root.
USER root
RUN whoami
```

```
# Runs as non-root user:
USER appuser
RUN whoami
```

If we run this:

```
$ docker build .
Sending build context to Docker daemon  2.048kB
...
Step 4/9 : USER appuser
 ---> Running in bd9f962c3173
Removing intermediate container bd9f962c3173
 ---> 30c7b4932cfd
Step 5/9 : RUN whoami
 ---> Running in c763f389036f
appuser
Removing intermediate container c763f389036f
 ---> 305bf441eb99
Step 6/9 : USER root
 ---> Running in a7f1d6ae91b8
Removing intermediate container a7f1d6ae91b8
 ---> 5ac4d87a852f
Step 7/9 : RUN whoami
 ---> Running in 81f4bc596dad
root
Removing intermediate container 81f4bc596dad
 ---> 4bc187b4892a
Step 8/9 : USER appuser
 ---> Running in 08db9249418a
Removing intermediate container 08db9249418a
 ---> 651753d0a56e
Step 9/9 : RUN whoami
 ---> Running in c9fb60a9627d
appuser
...
```

Note: Outside any specific best practice being demonstrated, the Dockerfiles in this article are not examples of best practices, since the added complexity would obscure the main point of the article.

Bad argument #2: The maintainers of the base image should install security updates

The Docker best practices documentation also used to say that “if a package contained in the parent image is out-of-date, contact its maintainers.”

It would indeed be nice if base images always had security updates. But let’s see what real world base images are like.

Here’s our Docker file, using the official `ubuntu` image as its base image. `ubuntu` is currently the third-most popular image on Docker Hub, so this is not an obscure example.

```
FROM ubuntu:20.04
RUN apt-get update && apt-get -y upgrade
```

I’m going to pull the latest base image, then build the image:

```
$ docker pull ubuntu 20.04
...
$ docker build .
$ docker build -t updated-debian .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM ubuntu:20.04
 ---> 4dd97cefde62
Step 2/2 : RUN apt-get update && apt-get -y upgrade
 ---> Running in 6a8d1c8c73ac
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
...
Fetched 17.3 MB in 5s (3644 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
```

```
Calculating upgrade...
The following packages will be upgraded:
  libsystemd0 libudev1 libzstd1
3 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 592 kB of archives.
After this operation, 2048 B of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 libsystemd0 ar
Get:2 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 libudev1 amd64
Get:3 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 libzstd1 amd64
debconf: delaying package configuration, since apt-utils is not installed
Fetched 592 kB in 1s (741 kB/s)
...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Removing intermediate container 6a8d1c8c73ac
 ---> 268228101da0
Successfully built 268228101da0
```

As you can see, three packages were installed—and at least one of them was a security update.

At the time of writing, March 23rd 2021, the `ubuntu:20.04` image was last rebuilt on March 3rd. On March 8th Ubuntu put out a security update for libzstd1, which is why we had to download and install a new version.

In short, relying on the base image just isn’t good enough. And remember, this is the *official Ubuntu Docker image*, blessed by Docker the company, maintained by a company with plenty of resources. Who knows what some small open source project, maintained in someone’s spare time, will do.

Bad argument #3: If you install security updates you will get the latest version of packages

This argument is made in a Cloudberry Engineering blog post: “This might be a bit of a stretch but the reasoning is the following: you want to pin the version of your software dependencies, if you do `apt-get upgrade` you will effectively upgrade them all to the latest version.”

This is, in fact, a stretch.

It’s true, there is an implicit process suggested here, one which they don’t make explicit and therefore most people are going to miss, but a security process nonetheless:

1. You keep a pinned list of every single system package.
2. You keep track of all security updates for your upstream packages.
3. Whenever a security update happens you update your pinned list manually.

If you’re doing that, great, that does actually ensure security updates. But realistically this is a lot of work, and most people are not going to do this.

Nor is it clear that manual pinning of system packages is actually useful for most people. Many users of Docker base images like `python:3.9` never install any new system packages. So all they’re installing is the base system, which in a stable operating system is mostly updated for—significant bug fixes and security updates.

You know what’s a good, easy way to make sure you have both security updates and fixes for major bugs? `apt-get upgrade`.

And yes, you don’t want everything changing out from under you, but that’s why you’re using a stable base image that only changes every couple years, like Debian Stable or Ubuntu LTS.

Bad argument #4: Upgrades don’t work

Hadolint is a `Dockerfile` linter, and it complains if you `apt-get upgrade` (though this will get fixed soon). The relevant wiki page cites some of the arguments we’ve already seen, as well as this argument: “[updating packages] can often fail if something is trying to modify init or make device changes inside a container.”

Now in all fairness to the author of the article being quoted, that original article was written in 2014, a very different time (in fact the article link 404s, I had to use the Wayback Machine to see the original). Docker’s initial release was in 2013, so yes, at the time Linux distributions did not necessarily work out of the box with Docker.

It’s not 2014 anymore, though, and Linux distributions are very much aware that they need package upgrades to work on Docker.

Please install security updates

To conclude, the idea you shouldn’t install security updates is based on either:

- Obsolete problems (“upgrades don’t work”).
- Theoretical ideal worlds we don’t live in (“the base image will install updates”).
- Non-sequiturs (“you can’t upgrade inside an unprivileged container”).
- Requiring a heavyweight process most people don’t need, namely pinning all system packages. You should be pinning your Python dependencies, though, since they’ll change meaningfully far more frequently than packages in a long-term-support Linux distribution.

Please, run `dnf/apk/apt-get upgrade` in your `Dockerfile`, you really do want to install security updates in your Docker image. And after that, make sure Docker caching doesn’t break your updating.

Free ebook: "Introduction to Dockerizing for Production"

Learn a step-by-step iterative DevOps packaging process in this **free mini-ebook**. You'll learn what to prioritize, the decisions you need to make, and the ongoing organizational processes you need to start.

Plus, you'll join over 8000 people getting weekly emails covering practical tools and techniques, from Docker packaging to Python best practices.

Your email address

Get your free mini-book

- Home
- About
- RSS Feed
- Mastodon
- Privacy policy
- Terms & Conditions