The best Docker base image for your Python application

by Itamar Turner-Trauring

When you're building a Docker image for your Python application, you're building on top of an existing image—and there are many possible choices. There are OS images like Ubuntu and CentOS, and there are the many different variants of the `python` base image.

Which one should you use? Which one is better? There are many choices, and it may not be obvious which is the best for your situation.

So to help you make a choice that fits your needs, in this article I'll go through some of the relevant criteria, and suggest some reasonable defaults that will work for most people.

What do you want from a base image?

There are a number of common criteria for choosing a base image, though your particular situation might emphasize, add, or remove some of these:

- **Stability:** You want a build today to give you the same basic set of libraries, directory structure, and infrastructure as a build tomorrow, otherwise your application will randomly break.
- **Security updates:** You want the base image to be well-maintained, so that you get security updates for the base operating system in a timely manner.
- **Up-to-date dependencies:** Unless you're building a very simple application, you will likely depend on operating system-installed libraries and applications (e.g. a compiler). You'd like them not to be too old.
- **Extensive dependencies:** For some applications less popular dependencies may be required—a base image with access to a large number of libraries makes this easier.
- **Up-to-date Python:** While this can be worked around by installing Python yourself, having an up-to-date Python available saves you some effort.
- **Small images:** All things being equal, it's better to have a smaller Docker image than a bigger Docker image.

The need for stability suggests not using operating systems with limited support lifetime, like Fedora or non-LTS Ubuntu releases.

Option #1: Ubuntu LTS, CentOS, Debian

There are three major operating systems that roughly meet the above criteria (dates and release versions are accurate at time of writing; the passage of time may require slightly different choices).

- Ubuntu 18.04 (the `ubuntu:18.04` image) was released in April 2018, and since it's a Long Term Support release it will get security updates until 2023.
- CentOS 7.6 (`centos:7.6.1810`) was released in October 2018, and will have full updates until Q4 2020 and maintenance updates until 2024.
- Debian 9 (aka "Stretch") was released in 2017, has updates until 2020, and LTS support until 2022. You can get newer packages via backports, but backports have no guarantee of security updates. Debian 10 ("Buster") should in theory be released in 2019, but Debian has been known to have delays in releases.

One problem with all of these images is that if you want the latest version of Python you'll have to install it yourself.

Option #2: The Python Docker image

Another alternative is the Docker `python` image, which comes pre-installed with specific versions of Python, and has multiple variants. For our purposes the interesting ones are:

- Alpine Linux, an operating system that was originally designed for small devices, and therefore tends to have small packages.
- Debian Stretch, with many common packages installed. The image itself is large, but the theory is that these packages are installed via common image layers that other Docker images will use, so overall disk usage will be low.
- Debian Stretch slim variant. This lacks the common packages' layers, and so the image itself is slow, but if you use many other Docker images based off Stretch the overall disk usage will be somewhat higher.

Why you shouldn't use Alpine Linux

A common suggestion for people who want small images is to use Alpine Linux, but using it has some costs. For one thing, Alpine has much fewer libraries than the other Linux distributions I mention above, so you might suffer from lack of a libraries.

There's also a major difference between Alpine and other Linux distributions: Alpine uses a different C library, musl, instead of the more common glibc. In theory musl and glibc are mostly compatible, but the differences can cause problems, and when problems do occur they are going to be strange and unexpected.

Some examples:

1. I once couldn't do DNS lookups in Alpine images running on minikube (Kubernetes in a VM) when using the WeWork coworking space's WiFi. The cause was a combination of a bad DNS setup by WeWork, the way Kubernetes and minikube do DNS, and musl's handling of this edge case vs. what glibc does. musl wasn't *wrong* (it matched the RFC), but I had to waste time figuring out the problem and then switching to a glibc-based image.
2. Another Alpine user discovered that their Python application was much slower because of the way musl allocates memory vs. glibc.
3. Alpine has a smaller default stack size for threads, which can lead to Python crashes.

These problems can usually be fixed or worked around, but it's just one more thing to worry about—and the corresponding benefit of a slightly smaller image isn't worth your time. **As a result, I recommend against using Alpine.**

So what should you use?

There's no point in using Debian Stretch directly, since there exist `python` base images with the latest Python installed on top of Stretch.

So as of May 2019:

1. `python:3.7-slim-stretch` or `python:3.7-stretch` (or whatever version of Python you use instead of `3.7`) are a reasonable starting point. I suspect for most people the `slim` variant will result in overall smaller disk usage.
2. If you need more recent libraries or compilers than those provided by Debian Stretch, you probably want to use `ubuntu:18.04`, which is more up-to-date than CentOS.
3. Once Debian Buster is released, the `python` images will likely have a `buster` variant which will be the clear winner: it will have both pre-installed new versions of Python and equivalent or more up-to-date packages than `ubuntu:18.04`.
4. And when April 2020 comes around, `ubuntu:20.04` will take the lead on having the most up-to-date packages.

You might also enjoy:

» Docker build caching can lead to insecure images

» Multi-stage Docker builds for Python: virtualenv, –user, and other methods

»» More articles on other topics