The best Docker base image for your Python application (August 2021)

by Itamar Turner-Trauring
Last updated 01 Oct 2021, originally created 30 Aug 2021

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 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.

Why you shouldn't use Alpine Linux

A common suggestion for people who want small images is to use Alpine Linux, but that can lead to longer build times, obscure bugs, and performance issues.

You can see the linked article for details, but I recommend against using Alpine.

Option #1: Ubuntu LTS, RedHat Universal Base Image, Debian

There are three major operating systems that roughly meet the above criteria: Debian "Bullseye" 11, Ubuntu 20.04 LTS, and RedHat Enterprise Linux 8. Ubuntu and RHEL have backported newer versions of Python, so even though they are older than Debian 11 they still include Python 3.9.

Distribution	Released	End-of-life	Image	Python versions
Debian 11	Aug 2021	Aug 2025	`debian:11`	3.9
Ubuntu 20.04	Apr 2020	Apr 2025	`ubuntu:20.04`	3.9, 3.8
RHEL 8	May 2019	May 2024	RedHat registry	3.9, 3.8, 3.6

Additional notes:

- Previous versions of this article covered CentOS, but CentOS is no longer a long-term stable operating system.
- RHEL 8 will have security updates until 2029.

Option #2: The Python Docker image

Another alternative is Docker's own "official" `python` image, which comes pre-installed with multiple versions of Python (`3.7`, `3.8`, `3.9`, etc.), and has multiple variants:

- Alpine Linux, which as I explained above I don't recommend using.
- Debian "Bullseye" 11, 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 official Docker images will use, so overall disk usage will be low.
- Debian 11 `slim` variant. This lacks the common packages' layers, and so the image itself is much smaller, but if you use many other "official" Docker images the overall disk usage will be somewhat higher.

For comparison purposes, the download size of `python:3.9-slim-bullseye` is 44MB, and `python:3.9-alpine` is 16MB. Their uncompressed on-disk sizes are 122MB and 45MB respectively.

Comparing the options

Python versions

The "official" Docker image has the benefit of providing every version of Python you might want, and tends to be very up-to-date with bugfix releases. This is not necessarily the case with the long-term support distributions. The table above shows the versions they include (omitting Python 2, which you really shouldn't be using anymore).

Focusing on point releases, at the time of writing (August 2021), 3.9.6 has been out since late June. Here is what the different images provide:

Image	3.9 release
Debian 11	3.9.2
Ubuntu 20.04	3.9.5

Image	3.9 release
RHEL 8	3.9.2
Docker python	3.9.6

As you can see, Ubuntu is doing a better (but not perfect) job at staying up-to-date, but the “official” Docker image is the only one that actually includes the latest point release.

When Python 3.10 comes out, the “official” Docker image will likely be out within days; it’s not clear when or if Ubuntu or RHEL will backport it, and Debian has not in the past done this sort of backport.

Performance

I’ve written a separate article where I ran some benchmarks comparing multiple Python builds, but here is a quick update using the `pystone` benchmark: not a perfect benchmark, but correlated with overall performance enough that I think it’s meaningful. The performance measure is kilopystone/sec; higher is better, numbers are approximate.

Image	Performance
Debian 11	~255
Ubuntu 20.04	~255
RHEL 8	~245
`python:3.9-bullseye`	~222

Debian and Ubuntu are the fastest, RHEL 8 is slightly slower, and the Docker Python images are even slower. Python 3.10 includes some performance optimizations in the build by default, so that might improve the situation for the Docker `python` images.

System packages

Debian 11, is the most up-to-date in terms of system packages and libraries. The “official” Docker `python` image is based off Debian 11, so it has access to the same set of up-to-date packages. Next is Ubuntu (from 2020), and RHEL (from 2019).

Ease of use and annoyances

- ➢ Both the `python:3.9-slim-bullseye` and `registry.access.redhat.com/ubi8/python-39` ship preconfigured with Python.
- ➢ With Debian and Ubuntu images, you need to add a couple of lines to your `Dockerfile` to install Python 3.9—a minor inconvenience.
- ➢ Ubuntu will end up installing Python 3.8 as a side-effect of installing Python 3.9, which is annoying and leads to larger images.

So which should you use?

- ➢ If you're a RedHat shop, you'll want to use their image.
- ➢ If you want the absolute latest bugfix version of Python, or a wide variety of versions, the official Docker Python image is your best bet.
- ➢ If you care about performance, Debian 11 or Ubuntu 20.04 will give you one of the fastest builds of Python; Ubuntu does better on point releases, but will have slightly larger images (see above). The difference is at most 10% though, and many applications are not bottlenecked on Python performance.

None of these tradeoffs are perfect; perhaps the 3.10 official Docker image will reduce the performance gap. Lacking specific constraints, **I'd probably choose the official Docker Python image (`python:3.9-slim-bullseye`)** just to ensure the latest bugfixes are always available.

Learn how to build fast, production-ready Docker images—read the rest of the Docker packaging guide for Python.

---

best practices.

Your email address

Get your free mini-book

Next: Why you really need to upgrade pip
Previous: The worst so-called “best practice” for Docker