php

Docker Official Image • 1B+ • 7.8K

While designed for web development, the PHP scripting language also provides general-purpose use.

LANGUAGES & FRAMEWORKS

Quick reference

- **Maintained by:**
  the Docker Community
- **Where to get help:**
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective `Dockerfile` links

Note: the description for this image is longer than the Hub length limit of 25000, so the "Supported tags" list has been trimmed to compensate. See also docker/hub-feedback#238 and docker/roadmap#475.

- See "Supported tags and respective `Dockerfile` links" at https://github.com/docker-library/docs/tree/master/php/README.md

Quick reference (cont.)

- **Where to file issues:**
  https://github.com/docker-library/php/issues
- **Supported architectures:** (more info)
  `amd64`, `arm32v5`, `arm32v6`, `arm32v7`, `arm64v8`, `i386`, `mips64le`, `ppc64le`, `riscv64`, `s390x`
- **Published image artifact details:**
  repo-info repo's `repos/php/` directory (history)
  (image metadata, transfer size, etc)
- **Image updates:**
  official-images repo's `library/php` label

official-images repo's `library/php` file (history)

- **Source of this description:**
  docs repo's `php/` directory (history)

What is PHP?

PHP is a server-side scripting language designed for web development, but which can also be used as a general-purpose programming language. PHP can be added to straight HTML or it can be used with a variety of templating engines and web frameworks. PHP code is usually processed by an interpreter, which is either implemented as a native module on the web-server or as a common gateway interface (CGI).

> wikipedia.org/wiki/PHP

How to use this image

Create a `Dockerfile` in your PHP project

```
FROM php:8.2-cli
COPY . /usr/src/myapp
WORKDIR /usr/src/myapp
CMD [ "php", "./your-script.php" ]
```

Then, run the commands to build and run the Docker image:

```
$ docker build -t my-php-app .
$ docker run -it --rm --name my-running-app my-php-app
```

Run a single PHP script

For many simple, single file projects, you may find it inconvenient to write a complete `Dockerfile`. In such cases, you can run a PHP script by using the PHP Docker image directly:

```
$ docker run -it --rm --name my-running-script -v "$PWD":/usr/src/myapp -w /usr/src/myapp
php:8.2-cli php your-script.php
```

How to install more PHP extensions

Many extensions are already compiled into the image, so it's worth checking the output of `php -m` or `php -i` before going through the effort of compiling more.

We provide the helper scripts `docker-php-ext-configure`, `docker-php-ext-install`, and `docker-php-ext-enable` to more easily install PHP extensions.

In order to keep the images smaller, PHP's source is kept in a compressed tar file. To facilitate linking of PHP's source with any extension, we also provide the helper script `docker-php-source` to easily extract the tar or delete the extracted source. Note: if you do use `docker-php-source` to extract the source, be sure to delete it in the same layer of the docker image.

```
FROM php:8.2-cli
RUN docker-php-source extract \
        # do important things \
        && docker-php-source delete
```

PHP Core Extensions

For example, if you want to have a PHP-FPM image with the `gd` extension, you can inherit the base image that you like, and write your own `Dockerfile` like this:

```
FROM php:8.2-fpm
RUN apt-get update && apt-get install -y \
                libfreetype-dev \
                libjpeg62-turbo-dev \
                libpng-dev \
        && docker-php-ext-configure gd --with-freetype --with-jpeg \
        && docker-php-ext-install -j$(nproc) gd
```

Remember, you must install dependencies for your extensions manually. If an extension needs custom `configure` arguments, you can use the `docker-php-ext-configure` script like this example. There is no need to run `docker-php-source` manually in this case, since that is handled by the `configure` and `install` scripts.

If you are having difficulty figuring out which Debian or Alpine packages need to be installed before `docker-php-ext-install`, then have a look at the `install-php-extensions` project. This script builds upon the `docker-php-ext-*` scripts and simplifies the installation of PHP extensions by automatically adding and removing Debian (apt) and Alpine (apk) packages. For example, to install the GD extension you simply have to run `install-php-extensions gd`. This tool is contributed by community members and is not included in the images, please refer to their Git repository for installation, usage, and issues.

See also "Dockerizing Compiled Software" for a description of the technique Tianon uses for determining the necessary build-time dependencies for any bit of software (which applies directly to compiling PHP extensions).

Default extensions

Some extensions are compiled by default. This depends on the PHP version you are using. Run `php -m` in the container to get a list for your specific version.

PECL extensions

Some extensions are not provided with the PHP source, but are instead available through PECL. To install a PECL extension, use `pecl install` to download and compile it, then use `docker-php-ext-enable` to enable it:

```
FROM php:8.2-cli
RUN pecl install redis-5.3.7 \
        && pecl install xdebug-3.2.1 \
        && docker-php-ext-enable redis xdebug
```

```
FROM php:8.2-cli
RUN apt-get update && apt-get install -y libmemcached-dev libssl-dev zlib1g-dev \
        && pecl install memcached-3.2.0 \
        && docker-php-ext-enable memcached
```

It is *strongly* recommended that users use an explicit version number in their `pecl install` invocations to ensure proper PHP version compatibility (PECL does not check the PHP version compatibility when choosing a version of the extension to install, but does when trying to install it). Beyond the compatibility issue, it's also a good practice to ensure you know when your dependencies receive updates and can control those updates directly.

Unlike PHP core extensions, PECL extensions should be installed in series to fail properly if something went wrong. Otherwise errors are just skipped by PECL. For example, `pecl install memcached-3.2.0 && pecl install redis-5.3.7` instead of `pecl install memcached-3.2.0 redis-5.3.7`. However, `docker-php-ext-enable memcached redis` is fine to be all in one command.

Other extensions

Some extensions are not provided via either Core or PECL; these can be installed too, although the process is less automated:

```
FROM php:8.2-cli
RUN curl -fsSL '[url-to-custom-php-module]' -o module-name.tar.gz \
        && mkdir -p module-name \
        && sha256sum -c "[shasum-value]  module-name.tar.gz" \
        && tar -xf module-name.tar.gz -C module-name --strip-components=1 \
        && rm module-name.tar.gz \
        && ( \
                cd module-name \
                && phpize \
                && ./configure --enable-module-name \
                && make -j "$(nproc)" \
                && make install \
        ) \
        && rm -r module-name \
        && docker-php-ext-enable module-name
```

The `docker-php-ext-*` scripts *can* accept an arbitrary path, but it must be absolute (to disambiguate from built-in extension names), so the above example could also be written as the following:

```
FROM php:8.2-cli
RUN curl -fsSL '[url-to-custom-php-module]' -o module-name.tar.gz \
        && mkdir -p /tmp/module-name \
        && sha256sum -c "[shasum-value]  module-name.tar.gz" \
        && tar -xf module-name.tar.gz -C /tmp/module-name --strip-components=1 \
        && rm module-name.tar.gz \
        && docker-php-ext-configure /tmp/module-name --enable-module-name \
        && docker-php-ext-install /tmp/module-name \
        && rm -r /tmp/module-name
```

Running as an arbitrary user

For running the Apache variants as an arbitrary user, there are a couple choices:

- If your kernel is version 4.11 or newer, you can add `--sysctl net.ipv4.ip_unprivileged_port_start=0` (which will be the default in a future version of Docker) and then `--user` should work as it does for FPM.
- If you adjust the Apache configuration to use an "unprivileged" port (greater than 1024 by default), then `--user` should work as it does for FPM regardless of kernel version.

For running the FPM variants as an arbitrary user, the `--user` flag to `docker run` should be used (which can accept both a username/group in the container's `/etc/passwd` file like `--user daemon` or a specific UID/GID like `--user 1000:1000`).

"`E: Package 'php-XXX' has no installation candidate`"

As of docker-library/php#542, this image blocks the installation of Debian's PHP packages. There is some additional discussion of this change in docker-library/php#551 (comment), but the gist is that installing Debian's PHP packages in this image leads to two conflicting installations of PHP in a single image, which is almost certainly not the intended outcome.

For those broken by this change and looking for a workaround to apply in the meantime while a proper fix is developed, adding the following simple line to your `Dockerfile` should remove the block (**with the strong caveat that this will allow the installation of a second installation of PHP, which is definitely not what you're looking for unless you *really* know what you're doing**):

```
RUN rm /etc/apt/preferences.d/no-debian-php
```

The *proper* solution to this error is to either use `FROM debian:XXX` and install Debian's PHP packages directly, or to use `docker-php-ext-install`, `pecl`, and/or `phpize` to install the necessary additional extensions and utilities.

Configuration

This image ships with the default `php.ini-development` and `php.ini-production` configuration files.

It is *strongly* recommended to use the production config for images used in production environments!

The default config can be customized by copying configuration files into the `$PHP_INI_DIR/conf.d/` directory.

Example

```
FROM php:8.2-fpm-alpine

# Use the default production configuration
RUN mv "$PHP_INI_DIR/php.ini-production" "$PHP_INI_DIR/php.ini"
```

In many production environments, it is also recommended to (build and) enable the PHP core OPcache extension for performance. See the upstream OPcache documentation for more details.

Image Variants

The `php` images come in many flavors, each designed for a specific use case.

Some of these tags may have names like bookworm or trixie in them. These are the suite code names for releases of Debian and indicate which release the image is based on. If your image needs to install any additional packages beyond what comes with the image, you'll likely want to specify one of these explicitly to minimize breakage when there are new releases of Debian.

`php:<version>-cli`

This variant contains the PHP CLI tool with default mods. If you need a web server, this is probably not the image you are looking for. It is designed to be used both as a throw away container (mount your source code and start the container to start your app), as well as a base from which to build other images.

It also is the only variant which contains the (not recommended) `php-cgi` binary, which is likely necessary for some things like PPM.

Note that *all* variants of `php` contain the PHP CLI (`/usr/local/bin/php`).

`php:<version>-apache`

This image contains Debian's Apache httpd in conjunction with PHP (as `mod_php`) and uses `mpm_prefork` by default.

Apache with a `Dockerfile`

```
FROM php:7.2-apache
COPY src/ /var/www/html/
```

Where `src/` is the directory containing all your PHP code. Then, run the commands to build and run the Docker image:

```
$ docker build -t my-php-app .
$ docker run -d --name my-running-app my-php-app
```

We recommend that you add a `php.ini` configuration file; see the "Configuration" section for details.

Apache without a `Dockerfile`

```
$ docker run -d -p 80:80 --name my-apache-php-app -v "$PWD":/var/www/html php:7.2-apache
```

Changing `DocumentRoot` (or other Apache configuration)

Some applications may wish to change the default `DocumentRoot` in Apache (away from `/var/www/html`). The following demonstrates one way to do so using an environment variable (which can then be modified at container runtime as well):

```
FROM php:7.1-apache

ENV APACHE_DOCUMENT_ROOT /path/to/new/root

RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/sites-available/*.
conf
RUN sed -ri -e 's!/var/www/!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/apache2.conf /etc/apac
he2/conf-available/*.conf
```

A similar technique could be employed for other Apache configuration options.

`php:<version>-fpm`

This variant contains PHP's FastCGI Process Manager (FPM), which is the recommended FastCGI implementation for PHP.

In order to use this image variant, some kind of reverse proxy (such as NGINX, Apache, or other tool which speaks the FastCGI protocol) will be required.

Some potentially helpful resources:

- FPM's Official Configuration Reference
- Simplified example by @md5
- Very detailed article by Pascal Landau
- Stack Overflow discussion
- Apache httpd Wiki example

WARNING: the FastCGI protocol is inherently trusting, and thus *extremely* insecure to expose outside of a private container network -- unless you know *exactly* what you are doing (and are willing to accept the extreme risk), do not use Docker's `--publish` (`-p`) flag with this image variant.

`php:<version>-alpine`

This image is based on the popular Alpine Linux project, available in the `alpine` official image. Alpine Linux is much smaller than most distribution base images (~5MB), and thus leads to much slimmer images in general.

This variant is useful when final image size being as small as possible is your primary concern. The main caveat to note is that it does use musl libc instead of glibc and friends, so software will often run into issues depending on the depth of their libc requirements/assumptions. See this Hacker News comment thread for more discussion of the issues that might arise and some pro/con comparisons of using Alpine-based images.

To minimize image size, it's uncommon for additional related tools (such as `git` or `bash`) to be included in Alpine-based images. Using this image as a base, add the things you need in your own Dockerfile (see the `alpine` image description for examples of how to install packages if you are unfamiliar).

License

View license information for the software contained in this image.

As with all Docker images, these likely also contain other software which may be under other licenses (such as Bash, etc from the base distribution, along with any direct or indirect dependencies of the primary software being contained).

Some additional license information which was able to be auto-detected might be found in the `repo-info` repository's `php/` directory.

As for any pre-built image usage, it is the image user's responsibility to ensure that any use of this image complies with any relevant licenses for all software contained within.

Tag summary

Recent tags

8.5.1RC2-zts-alpine3....

Content type

Image

Digest

sha256:96867c62e...

Size

48.1 MB

Last updated

37 minutes ago

```
docker pull php:8.5.1RC2-zts-alpine3.23
```

This weeks pulls

Pulls: 1,326,409

Last week

Learn more

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.