# Instalação Wordpress Docker Oficial

Link: https://hub.docker.com/_/wordpress

Atualização: 05/12/2025

# wordpress

Docker Official Image

The WordPress rich content management system can utilize plugins, widgets, and themes.

# wordpress Docker official image overview

## 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

- `6.9.0-php8.1-apache`, `6.9-php8.1-apache`, `6-php8.1-apache`, `php8.1-apache`, `6.9.0-php8.1`, `6.9-php8.1`, `6-php8.1`, `php8.1`
- `6.9.0-php8.1-fpm`, `6.9-php8.1-fpm`, `6-php8.1-fpm`, `php8.1-fpm`
- `6.9.0-php8.1-fpm-alpine`, `6.9-php8.1-fpm-alpine`, `6-php8.1-fpm-alpine`, `php8.1-fpm-alpine`

- `6.9.0-php8.2-apache`, `6.9-php8.2-apache`, `6-php8.2-apache`, `php8.2-apache`, `6.9.0-php8.2`, `6.9-php8.2`, `6-php8.2`, `php8.2`
- `6.9.0-php8.2-fpm`, `6.9-php8.2-fpm`, `6-php8.2-fpm`, `php8.2-fpm`
- `6.9.0-php8.2-fpm-alpine`, `6.9-php8.2-fpm-alpine`, `6-php8.2-fpm-alpine`, `php8.2-fpm-alpine`
- `6.9.0-apache`, `6.9-apache`, `6-apache`, `apache`, `6.9.0`, `6.9`, `6`, `latest`, `6.9.0-php8.3-apache`, `6.9-php8.3-apache`, `6-php8.3-apache`, `php8.3-apache`, `6.9.0-php8.3`, `6.9-php8.3`, `6-php8.3`, `php8.3`
- `6.9.0-fpm`, `6.9-fpm`, `6-fpm`, `fpm`, `6.9.0-php8.3-fpm`, `6.9-php8.3-fpm`, `6-php8.3-fpm`, `php8.3-fpm`
- `6.9.0-fpm-alpine`, `6.9-fpm-alpine`, `6-fpm-alpine`, `fpm-alpine`, `6.9.0-php8.3-fpm-alpine`, `6.9-php8.3-fpm-alpine`, `6-php8.3-fpm-alpine`, `php8.3-fpm-alpine`
- `6.9.0-php8.4-apache`, `6.9-php8.4-apache`, `6-php8.4-apache`, `php8.4-apache`, `6.9.0-php8.4`, `6.9-php8.4`, `6-php8.4`, `php8.4`
- `6.9.0-php8.4-fpm`, `6.9-php8.4-fpm`, `6-php8.4-fpm`, `php8.4-fpm`
- `6.9.0-php8.4-fpm-alpine`, `6.9-php8.4-fpm-alpine`, `6-php8.4-fpm-alpine`, `php8.4-fpm-alpine`
- `6.9.0-php8.5-apache`, `6.9-php8.5-apache`, `6-php8.5-apache`, `php8.5-apache`, `6.9.0-php8.5`, `6.9-php8.5`, `6-php8.5`, `php8.5`
- `6.9.0-php8.5-fpm`, `6.9-php8.5-fpm`, `6-php8.5-fpm`, `php8.5-fpm`
- `6.9.0-php8.5-fpm-alpine`, `6.9-php8.5-fpm-alpine`, `6-php8.5-fpm-alpine`, `php8.5-fpm-alpine`
- `cli-2.12.0-php8.1`, `cli-2.12-php8.1`, `cli-2-php8.1`, `cli-php8.1`
- `cli-2.12.0-php8.2`, `cli-2.12-php8.2`, `cli-2-php8.2`, `cli-php8.2`
- `cli-2.12.0`, `cli-2.12`, `cli-2`, `cli`, `cli-2.12.0-php8.3`, `cli-2.12-php8.3`, `cli-2-php8.3`, `cli-php8.3`
- `cli-2.12.0-php8.4`, `cli-2.12-php8.4`, `cli-2-php8.4`, `cli-php8.4`
- `cli-2.12.0-php8.5`, `cli-2.12-php8.5`, `cli-2-php8.5`, `cli-php8.5`

# Quick reference (cont.)

- **Where to file issues**: https://github.com/docker-library/wordpress/issues
- **Supported architectures**: (more info) `amd64`, `arm32v5`, `arm32v6`, `arm32v7`, `arm64v8`, `i386`, `ppc64le`, `riscv64`, `s390x`

- **Published image artifact details**:

  repo-info repo's `repos/wordpress/` directory (history)
  (image metadata, transfer size, etc)
- **Image updates**:

  official-images repo's `library/wordpress` label

  official-images repo's `library/wordpress` file (history)
- **Source of this description**:

  docs repo's `wordpress/` directory (history)

# What is WordPress?

WordPress is a free and open source blogging tool and a content management system (CMS) based on PHP and MySQL, which runs on a web hosting service. Features include a plugin architecture and a template system. WordPress is used by more than 22.0% of the top 10 million websites as of August 2013. WordPress is the most popular blogging system in use on the Web, at more than 60 million websites. The most popular languages used are English, Spanish and Bahasa Indonesia.

> “ wikipedia.org/wiki/WordPress

logo

# How to use this image

```
$ docker run --name some-wordpress --network some-network -d wordpress
```

The following environment variables are also honored for configuring your WordPress instance (by a custom `wp-config.php` implementation):

- `-e WORDPRESS_DB_HOST=...`
- `-e WORDPRESS_DB_USER=...`
- `-e WORDPRESS_DB_PASSWORD=...`
- `-e WORDPRESS_DB_NAME=...`
- `-e WORDPRESS_TABLE_PREFIX=...`
- `-e WORDPRESS_AUTH_KEY=...`, `-e WORDPRESS_SECURE_AUTH_KEY=...`, `-e WORDPRESS_LOGGED_IN_KEY=...`, `-e WORDPRESS_NONCE_KEY=...`, `-e WORDPRESS_AUTH_SALT=...`, `-e WORDPRESS_SECURE_AUTH_SALT=...`, `-e WORDPRESS_LOGGED_IN_SALT=...`, `-e WORDPRESS_NONCE_SALT=...` (default to unique random SHA1s, but only if other environment variable configuration is provided)

- `-e WORDPRESS_DEBUG=1` (defaults to disabled, non-empty value will enable `WP_DEBUG` in `wp-config.php`)
- `-e WORDPRESS_CONFIG_EXTRA=...` (defaults to nothing, the value will be evaluated by the `eval()` function in `wp-config.php`. This variable is especially useful for applying extra configuration values this image does not provide by default such as `WP_ALLOW_MULTISITE`; see docker-library/wordpress#142 for more details)

The `WORDPRESS_DB_NAME` needs to already exist on the given MySQL server; it will not be created by the `wordpress` container.

If you'd like to be able to access the instance from the host without the container's IP, standard port mappings can be used:

```
$ docker run --name some-wordpress -p 8080:80 -d wordpress
```

Then, access it via `http://localhost:8080` or `http://host-ip:8080` in a browser.

When running WordPress with TLS behind a reverse proxy such as NGINX which is responsible for doing TLS termination, be sure to set `X-Forwarded-Proto` appropriately (see "Using a Reverse Proxy" in "Administration Over SSL" in upstream's documentation). No additional environment variables or configuration should be necessary (this image automatically adds the noted `HTTP_X_FORWARDED_PROTO` code to `wp-config.php` if *any* of the above-noted environment variables are specified).

If your database requires SSL, WordPress ticket #28625 has the relevant details regarding support for that with WordPress upstream. As a workaround, the "Secure DB Connection" plugin can be extracted into the WordPress directory and the appropriate values described in the configuration of that plugin added in `wp-config.php`.

## Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files. For example:

```
$ docker run --name some-wordpress -e WORDPRESS_DB_PASSWORD_FILE=/run/secrets/mysql-root ... -
d wordpress:tag
```

Currently, this is supported for `WORDPRESS_DB_HOST`, `WORDPRESS_DB_USER`, `WORDPRESS_DB_PASSWORD`, `WORDPRESS_DB_NAME`, `WORDPRESS_AUTH_KEY`, `WORDPRESS_SECURE_AUTH_KEY`, `WORDPRESS_LOGGED_IN_KEY`, `WORDPRESS_NONCE_KEY`, `WORDPRESS_AUTH_SALT`, `WORDPRESS_SECURE_AUTH_SALT`, `WORDPRESS_LOGGED_IN_SALT`, `WORDPRESS_NONCE_SALT`, `WORDPRESS_TABLE_PREFIX`, and `WORDPRESS_DEBUG`.

## ... via `docker compose`?

Example `compose.yaml` for `wordpress`:

```
services:

  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html

  db:
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - db:/var/lib/mysql

volumes:
  wordpress:
  db:
```

Run `docker compose up`, wait for it to initialize completely, and visit `http://localhost:8080` or `http://host-ip:8080` (as appropriate).

## Adding additional libraries / extensions

This image does not provide any additional PHP extensions or other libraries, even if they are required by popular plugins (e.g. it cannot send e-mails). There are an infinite number of possible

plugins, and they potentially require any extension PHP supports. Including every PHP extension that exists would dramatically increase the image size.

If you need additional PHP extensions, you'll need to create your own image `FROM` this one. The documentation of the `php` image explains how to compile additional extensions. Additionally, an older `Dockerfile` for `wordpress` has a simplified example of doing this and a newer version of that same `Dockerfile` has a much more thorough example.

## Include pre-installed themes / plugins

Mount the volume containing your themes or plugins to the proper directory; and then apply them through the "wp-admin" UI. Ensure read/write/execute permissions are in place for the user:

- Themes go in a subdirectory in `/var/www/html/wp-content/themes/`
- Plugins go in a subdirectory in `/var/www/html/wp-content/plugins/`

If you wish to provide additional content in an image for deploying in multiple installations, place it in the same directories under `/usr/src/wordpress/` instead (which gets copied to `/var/www/html/` on the container's initial startup).

## Static image / updates-via-redeploy

The default configuration for this image matches the official WordPress defaults in which automatic updates are enabled (so the initial install comes from the image, but after that it becomes self-managing within the `/var/www/html/` data volume).

If you wish to have a more static deployment (similar to other containerized applications) and deploy new containers to update WordPress + themes/plugins, then you'll want to use something like the following (and run the resulting image read-only):

```
FROM wordpress:apache
WORKDIR /usr/src/wordpress
RUN set -eux; \
	find /etc/apache2 -name '*.conf' -type f -exec sed -ri -e "s!/var/www/html!$PWD!g" -e "s!Directory /var/www/!Directory $PWD!g" '{}' +; \
	cp -s wp-config-docker.php wp-config.php
COPY custom-theme/ ./wp-content/themes/custom-theme/
COPY custom-plugin/ ./wp-content/plugins/custom-plugin/
```

For FPM-based images, remove the `find` instruction and adjust the `SCRIPT_FILENAME` paths in your reverse proxy from `/var/www/html` to `/usr/src/wordpress`.

Run the result read-only, providing writeable storage for `/tmp`, `/run`, and (optionally) `wp-content/uploads`:

```
$ docker run ... \
  --read-only \
  --tmpfs /tmp \
  --tmpfs /run \
  --mount type=...,src=...,dst=/usr/src/wordpress/wp-content/uploads \
  ... \
  --env WORDPRESS_DB_HOST=... \
  --env WORDPRESS_AUTH_KEY=... \
  --env ... \
  custom-wordpress:tag
```

**Note:** be sure to rebuild and redeploy regularly to ensure you get all the latest WordPress security updates.

## Running as an arbitrary user

See the "Running as an arbitrary user" section of the `php` image documentation.

When running WP-CLI via the `cli` variants of this image, it is important to note that they're based on Alpine, and have a default `USER` of Alpine's `www-data`, whose UID is `82` (compared to the Debian-based WordPress variants whose default effective UID is `33`), so when running `wordpress:cli` against an existing Debian-based WordPress install, something like `--user 33:33` is likely going to be necessary (possibly also something like `-e HOME=/tmp` depending on the `wp` command invoked and whether it tries to use `~/.wp-cli`). See docker-library/wordpress#256 for more discussion around this.

## Configuring PHP directives

See the "Configuration" section of the `php` image documentation.

For example, to adjust common `php.ini` flags like `upload_max_filesize`, you could create a `custom.ini` with the desired parameters and place it in the `$PHP_INI_DIR/conf.d/` directory:

```
FROM wordpress:tag
COPY custom.ini $PHP_INI_DIR/conf.d/
```

# Image Variants

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

### `wordpress:<version>`

This is the defacto image. If you are unsure about what your needs are, you probably want to use this one. 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 the base to build other images off of.

## `wordpress:<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.

## `wordpress:cli`

This image variant does not contain WordPress itself, but instead contains WP-CLI.

The simplest way to use it with an existing WordPress container would be something similar to the following:

```
$ docker run -it --rm \
--volumes-from some-wordpress \
--network container:some-wordpress \
-e WORDPRESS_DB_USER=... \
-e WORDPRESS_DB_PASSWORD=... \
# [and other used environment variables]
wordpress:cli user list
```

Generally speaking, for WP-CLI to interact with a WordPress install, it needs access to the on-disk files of the WordPress install, and access to the database (and the easiest way to accomplish that such that `wp-config.php` does not require changes is to simply join the networking context of the

existing and presumably working WordPress container, but there are many other ways to accomplish that which will be left as an exercise for the reader).

**NOTE:** Since March 2021, WordPress images use a customized `wp-config.php` that pulls the values directly from the environment variables defined above (see `wp-config-docker.php` in docker-library/wordpress#572 and docker-library/wordpress#577). As a result of reading environment variables directly, the cli container also needs the same set of environment variables to properly evaluate `wp-config.php`.

# 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 `wordpress/` 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.

---

Revision #1
Created 5 December 2025 12:15:39 by Administrador
Updated 5 December 2025 12:18:14 by Administrador