nginx ★

Docker Official Images

Official build of Nginx.

10M+

Container Linux IBM Z PowerPC 64 LE 386 ARM 64 ARM x86-64 Application Infrastructure Official Image

Linux - ARM 64 (latest)

Copy and paste to pull this image

```
docker pull nginx
```

View Available Tags

DESCRIPTION REVIEWS TAGS

Supported tags and respective `Dockerfile` links

- `1.17.1` , `mainline` , `1` , `1.17` , `latest`
- `1.17.1-perl` , `mainline-perl` , `1-perl` , `1.17-perl` , `perl`
- `1.17.1-alpine` , `mainline-alpine` , `1-alpine` , `1.17-alpine` , `alpine`
- `1.17.1-alpine-perl` , `mainline-alpine-perl` , `1-alpine-perl` , `1.17-alpine-perl` , `alpine-perl`
- `1.16.0` , `stable` , `1.16`
- `1.16.0-perl` , `stable-perl` , `1.16-perl`
- `1.16.0-alpine` , `stable-alpine` , `1.16-alpine`
- `1.16.0-alpine-perl` , `stable-alpine-perl` , `1.16-alpine-perl`

Quick reference

- **Where to get help**:
  the Docker Community Forums, the Docker Community Slack, or Stack Overflow
- **Where to file issues**:
  https://github.com/nginxinc/docker-nginx/issues
- **Maintained by**:
  the NGINX Docker Maintainers
- **Supported architectures**: (more info)
  `amd64` , `arm32v6` , `arm32v7` , `arm64v8` , `i386` , `ppc64le` , `s390x`
- **Published image artifact details**:
  repo-info repo's `repos/nginx/` directory (history)
  (image metadata, transfer size, etc)

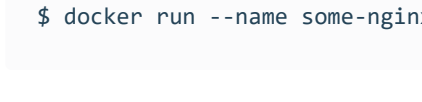
- **Image updates**:
  official-images PRs with label `library/nginx`
  official-images repo's `library/nginx` file (history)

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

What is nginx?

Nginx (pronounced "engine-x") is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server (origin server). The nginx project started with a strong focus on high concurrency, high performance and low memory usage. It is licensed under the 2-clause BSD-like license and it runs on Linux, BSD variants, Mac OS X, Solaris, AIX, HP-UX, as well as on other *nix flavors. It also has a proof of concept port for Microsoft Windows.

wikipedia.org/wiki/Nginx



How to use this image

Hosting some simple static content

```
$ docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Alternatively, a simple `Dockerfile` can be used to generate a new image that includes the necessary content (which is a much cleaner solution than the bind mount above):

```
FROM nginx
COPY static-html-directory /usr/share/nginx/html
```

Place this file in the same directory as your directory of content ("static-html-directory"), run `docker build -t some-content-nginx .`, then start your container:

```
$ docker run --name some-nginx -d some-content-nginx
```

Exposing external port

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

Then you can hit `http://localhost:8080` or `http://host-ip:8080` in your browser.

Complex configuration

```
$ docker run --name my-custom-nginx-container -v /host/path/nginx.conf:/etc/nginx/nginx.conf:ro -d nginx
```

For information on the syntax of the nginx configuration files, see the official documentation (specifically the Beginner's Guide).

If you wish to adapt the default configuration, use something like the following to copy it from a running nginx container:

```
$ docker run --name tmp-nginx-container -d nginx
$ docker cp tmp-nginx-container:/etc/nginx/nginx.conf /host/path/nginx.conf
$ docker rm -f tmp-nginx-container
```

This can also be accomplished more cleanly using a simple `Dockerfile` (in `/host/path/`):

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

If you add a custom `CMD` in the Dockerfile, be sure to include `-g daemon off;` in the `CMD` in order for nginx to stay in the foreground, so that Docker can track the process properly (otherwise your container will stop immediately after starting)!

Then build the image with `docker build -t custom-nginx .` and run it as follows:

```
$ docker run --name my-custom-nginx-container -d custom-nginx
```

Using environment variables in nginx configuration

Out-of-the-box, nginx doesn't support environment variables inside most configuration blocks. But `envsubst` may be used as a workaround if you need to generate your nginx configuration dynamically before nginx starts.

Here is an example using docker-compose.yml:

```
web:
  image: nginx
  volumes:
   - ./mysite.template:/etc/nginx/conf.d/mysite.template
  ports:
   - "8080:80"
  environment:
   - NGINX_HOST=foobar.com
   - NGINX_PORT=80
  command: /bin/bash -c "envsubst < /etc/nginx/conf.d/mysite.template > /etc/nginx/conf.d/default.conf && exec nginx -g 'daemon off;'"
```

The `mysite.template` file may then contain variable references like this:

```
listen ${NGINX_PORT};
```

Running nginx in read-only mode

To run nginx in read-only mode, you will need to mount a Docker volume to every location where nginx writes information. The default nginx configuration requires write access to `/var/cache` and `/var/run`. This can be easily accomplished by running nginx as follows:

```
$ docker run -d -p 80:80 --read-only -v $(pwd)/nginx-cache:/var/cache/nginx -v $(pwd)/nginx-pid:/var/run nginx
```

If you have a more advanced configuration that requires nginx to write to other locations, simply add more volume mounts to those locations.

Running nginx in debug mode

Images since version 1.9.8 come with `nginx-debug` binary that produces verbose output when using higher log levels. It can be used with simple CMD substitution:

```
$ docker run --name my-nginx -v /host/path/nginx.conf:/etc/nginx/nginx.conf:ro -d nginx nginx-debug -g 'daemon off;'
```

Similar configuration in docker-compose.yml may look like this:

```
web:
  image: nginx
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
  command: [nginx-debug, '-g', 'daemon off;']
```

User and group id

Since 1.17.0, both alpine- and debian-based images variants use the same user and group ids to drop the privileges for worker processes:

```
$ id
uid=101(nginx) gid=101(nginx) groups=101(nginx)
```

Running nginx as a non-root user

It is possible to run the image as a less privileged arbitrary UID/GID. This, however, requires modification of nginx configuration to use directories writeable by that specific UID/GID pair:

```
$ docker run -d -v $PWD/nginx.conf:/etc/nginx/nginx.conf nginx
```

where nginx.conf in the current directory should have the following directives re-defined:

```
pid        /tmp/nginx.pid;
```

And in the http context:

```
http {
    client_body_temp_path /tmp/client_temp;
    proxy_temp_path       /tmp/proxy_temp_path;
    fastcgi_temp_path     /tmp/fastcgi_temp;
    uwsgi_temp_path       /tmp/uwsgi_temp;
    scgi_temp_path        /tmp/scgi_temp;
...
}
```

Monitoring nginx with Amplify

Amplify is a free monitoring tool that can be used to monitor microservice architectures based on nginx. Amplify is developed and maintained by the company behind the nginx software.

With Amplify it is possible to collect and aggregate metrics across containers, and present a coherent set of visualizations of the key performance data, such as active connections or requests per second. It is also easy to quickly check for any performance degradations, traffic anomalies, and get a deeper insight into the nginx configuration in general.

In order to use Amplify, a small Python-based agent software (Amplify Agent) should be installed inside the container.

For more information about Amplify, please check the official documentation here.

Image Variants

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

`nginx:<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.

`nginx:<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 highly recommended when final image size being as small as possible is desired. The main caveat to note is that it does use musl libc instead of glibc and friends, so certain software might run into issues depending on the depth of their libc requirements. However, most software doesn't have an issue with this, so this variant is usually a very safe choice. 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 `nginx/` 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.

Be the first to give insight into your experience by rating and reviewing the product.

Add Product Review

Select a product tier

Start Your Review