ProjectSend Docker Instalação

Link: https://docs.linuxserver.io/images/docker-projectsend/

Projectsend is a self-hosted application that lets you upload files and assign them to specific clients that you create yourself. Secure, private and easy. No more depending on external services or e-mail to send those files.

projectsend

Supported Architectures

We utilise the docker manifest for multi-platform awareness. More information is available from docker here and our announcement here.

Simply pulling `lscr.io/linuxserver/projectsend:latest` should retrieve the correct image for your arch, but you can also pull specific arch images via tags.

The architectures supported by this image are:

Architecture	Available	Tag
x86-64	✅	amd64-<version tag>
arm64	✅	arm64v8-<version tag>
armhf	✅	

Application Setup

Requires a user and database in either mysql or mariadb.

To set PHP options like max upload size please edit /config/php/projectsend.ini

To use translations, follow the instructions here. The necessary paths are symlinked under `/config/translations` (note that the "templates" paths don't need `lang` subdirectories).

More info at ProjectSend.

Usage

To help you get started creating a container from this image you can either use docker-compose or the docker cli.

Info

Unless a parameter is flaged as 'optional', it is *mandatory* and a value must be provided.

docker-compose (recommended, click here for more info)¶

```
---
services:
  projectsend:
    image: lscr.io/linuxserver/projectsend:latest
    container_name: projectsend
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Etc/UTC
    volumes:
      - /path/to/projectsend/config:/config
      - /path/to/data:/data
    ports:
      - 80:80
    restart: unless-stopped
```

docker cli (click here for more info)¶

```
docker run -d \
  --name=projectsend \
  -e PUID=1000 \
  -e PGID=1000 \
  -e TZ=Etc/UTC \
  -p 80:80 \
  -v /path/to/projectsend/config:/config \
  -v /path/to/data:/data \
  --restart unless-stopped \
  lscr.io/linuxserver/projectsend:latest
```

Parameters

Containers are configured using parameters passed at runtime (such as those above). These parameters are separated by a colon and indicate `<external>:<internal>` respectively. For example, `-p 8080:80` would expose port `80` from inside the container to be accessible from the host's IP on port `8080` outside the container.

Ports (`-p`)

Parameter	Function
`80:80`	WebUI

Environment Variables (`-e`)

Env	Function
`PUID=1000`	for UserID - see below for explanation
`PGID=1000`	for GroupID - see below for explanation
`TZ=Etc/UTC`	specify a timezone to use, see this list.

Volume Mappings (`-v`)

Volume	Function
`/config`	Persistent config files
`/data`	Where to store files to share.

Miscellaneous Options

Parameter	Function

Environment variables from files (Docker secrets)

You can set any environment variable from a file by using a special prepend `FILE__`.

As an example:

```
-e FILE__MYVAR=/run/secrets/mysecretvariable
```

Will set the environment variable `MYVAR` based on the contents of the `/run/secrets/mysecretvariable` file.

Umask for running applications

For all of our images we provide the ability to override the default umask settings for services started within the containers using the optional `-e UMASK=022` setting. Keep in mind umask is not chmod it subtracts from permissions based on it's value it does not add. Please read up here before asking for support.

User / Group Identifiers

When using volumes (`-v` flags), permissions issues can arise between the host OS and the container, we avoid this issue by allowing you to specify the user `PUID` and group `PGID`.

Ensure any volume directories on the host are owned by the same user you specify and any permissions issues will vanish like magic.

In this instance `PUID=1000` and `PGID=1000`, to find yours use `id your_user` as below:

```
id your_user
```

Example output:

```
uid=1000(your_user) gid=1000(your_user) groups=1000(your_user)
```

Docker Mods

Docker Mods Docker Universal Mods

We publish various Docker Mods to enable additional functionality within the containers. The list of Mods available for this image (if any) as well as universal mods that can be applied to any one of our images can be accessed via the dynamic badges above.

Support Info

- Shell access whilst the container is running:

```
docker exec -it projectsend /bin/bash
```

- To monitor the logs of the container in realtime:

```
docker logs -f projectsend
```

- Container version number:

```
docker inspect -f '{{ index .Config.Labels "build_version" }}' projectsend
```

- Image version number:

```
docker inspect -f '{{ index .Config.Labels "build_version" }}' lscr.io/linuxserver/projectsend:latest
```

Updating Info

Most of our images are static, versioned, and require an image update and container recreation to update the app inside. With some exceptions (noted in the relevant readme.md), we do not recommend or support updating apps inside the container. Please consult the Application Setup section above to see if it is recommended for the image.

Below are the instructions for updating containers:

Via Docker Compose

- Update images:
  - All images:

```
docker-compose pull
```

  - Single image:

```
docker-compose pull projectsend
```

- Update containers:
  - All containers:

```
docker-compose up -d
```

  - Single container:

```
docker-compose up -d projectsend
```

- You can also remove the old dangling images:

```
docker image prune
```

Via Docker Run

- Update the image:

```
docker pull lscr.io/linuxserver/projectsend:latest
```

- Stop the running container:

```
docker stop projectsend
```

- Delete the container:

```
docker rm projectsend
```

- Recreate a new container with the same docker run parameters as instructed above (if mapped correctly to a host folder, your `/config` folder and settings will be preserved)
- You can also remove the old dangling images:

```
docker image prune
```

Image Update Notifications - Diun (Docker Image Update Notifier)

Tip

We recommend Diun for update notifications. Other tools that automatically update containers unattended are not recommended or supported.

Building locally

If you want to make local modifications to these images for development purposes or just to customize the logic:

```
git clone https://github.com/linuxserver/docker-projectsend.git
cd docker-projectsend
docker build \
  --no-cache \
  --pull \
  -t lscr.io/linuxserver/projectsend:latest .
```

The ARM variants can be built on x86_64 hardware and vice versa using `lscr.io/linuxserver/qemu-static`

```
docker run --rm --privileged lscr.io/linuxserver/qemu-static --reset
```

Once registered you can define the dockerfile to use with `-f Dockerfile.aarch64`.

To help with development, we generate this dependency graph.

Init dependency graph

Versions

- **21.12.24:** - Rebase to Alpine 3.21, move php .ini file to /config/php.
- **06.06.24:** - Rebase to Alpine 3.20.
- **23.12.23:** - Rebase to Alpine 3.19 with php 8.3.
- **25.05.23:** - Rebase to Alpine 3.18, deprecate armhf.
- **08.03.23:** - Rebasing to alpine 3.17 and upgrading to s6v3.
- **23.08.22:** - Add translation support
- **20.08.22:** - Rebasing to alpine 3.15 with php8. Restructure nginx configs (see changes announcement).
- **24.06.21:** - Rebasing to alpine 3.14, switch to nginx
- **23.01.21:** - Rebasing to alpine 3.13.
- **01.06.20:** - Rebasing to alpine 3.12.
- **31.12.19:** - Rebase to Alpine 3.11 and upgrade to PHP7.
- **23.03.19:** - Switching to new Base images, shift to arm32v7 tag.
- **11.02.19:** - Add pipeline logic and multi arch.
- **11.06.17:** - Fetch version from github.
- **09.12.17:** - Rebase to alpine 3.7.
- **13.06.17:** - Initial Release.

February 3, 2025 February 5, 2019

Revision #2
Created 17 April 2025 00:06:42 by Administrador
Updated 17 April 2025 00:16:30 by Administrador