Docker

- Install Docker & Docker Compose
- Portainer
- Pi-hole
- Bookstack
- DocuSeal
- Stirling-pdf

Install Docker & Docker Compose

docker

Uninstall old versions

Before you can install Docker Engine, you need to uninstall any conflicting packages.

Your Linux distribution may provide unofficial Docker packages, which may conflict with the official packages provided by Docker. You must uninstall these packages before you install the official version of Docker Engine.

The unofficial packages to uninstall are:

- `docker.io`
- `docker-compose`
- `docker-compose-v2`
- `docker-doc`
- `podman-docker`

Moreover, Docker Engine depends on `containerd` and `runc`. Docker Engine bundles these dependencies as one bundle: `containerd.io`. If you have installed the `containerd` or `runc` previously, uninstall them to avoid conflicts with the versions bundled with Docker Engine.

Run the following command to uninstall all conflicting packages:

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

`apt-get` might report that you have none of these packages installed.

Images, containers, volumes, and networks stored in /var/lib/docker/ aren't automatically removed when you uninstall Docker. If you want to start with a clean installation, and prefer to clean up any existing data, read the uninstall Docker Engine section.

Installation methods

You can install Docker Engine in different ways, depending on your needs:

- Docker Engine comes bundled with Docker Desktop for Linux. This is the easiest and quickest way to get started.
- Set up and install Docker Engine from Docker's apt repository.
- Install it manually and manage upgrades manually.
- Use a convenience script. Only recommended for testing and development environments.

Install using the apt repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker apt repository. Afterward, you can install and update Docker from the repository.

1. Set up Docker's `apt` repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

2. Install the Docker packages.

Latest

To install the latest version, run:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

Specific Version

To install a specific version of Docker Engine, start by listing the available versions in the repository:

```
# List the available versions:
apt-cache madison docker-ce | awk '{ print $3 }'

5:28.0.1-1~ubuntu.24.04~noble
5:28.0.0-1~ubuntu.24.04~noble
...
```

Select the desired version and install:

```
VERSION_STRING=5:28.0.1-1~ubuntu.24.04~noble
sudo apt-get install docker-ce=$VERSION_STRING docker-ce-cli=$VERSION_STRING containerd.io
docker-buildx-plugin docker-compose-plugin
```

3. Verify that the installation is successful by running the hello-world image:

```
sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints a confirmation message and exits.

You have now successfully installed and started Docker Engine.

> Receiving errors when trying to run without root? The docker user group exists but contains no users, which is why you’re required to use sudo to run Docker commands. >Continue to Linux postinstall to allow non-privileged users to run Docker commands and for other optional >configuration steps.

Upgrade Docker Engine

To upgrade Docker Engine, follow step 2 of the installation instructions, choosing the new version you want to install.

Portainer

portainer

Deployment

First, create the volume that Portainer Server will use to store its database:

```
docker volume create portainer_data
```

Then, download and install the Portainer Server container:

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:lts
```

> By default, Portainer generates and uses a self-signed SSL certificate to secure port 9443. Alternatively you can >provide your own SSL certificate during installation or via the Portainer UI after installation is complete.

> If you require HTTP port 9000 open for legacy reasons, add the following to your docker run command:
>
> `-p 9000:9000`

Portainer Server has now been installed. You can check to see whether the Portainer Server container has started by running `docker ps`:

```
root@server:~# docker ps
CONTAINER ID   IMAGE                           COMMAND                  CREATED
STATUS
PORTS
NAMES
de5b28eb2fa9   portainer/portainer-ce:lts      "/portainer"             2 weeks ago   Up 9
days   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp
portainer
```

Logging In

Now that the installation is complete, you can log into your Portainer Server instance by opening a web browser and going to:

```
https://localhost:9443
```

Replace `localhost` with the relevant IP address or FQDN if needed, and adjust the port if you changed it earlier.

You will be presented with the initial setup page for Portainer Server.

Pi-hole

Pi-hole website

Network-wide ad blocking via your own Linux hardware

Pi-hole website | Documentation | Discourse Forum | Donate

Upgrade Notes

> [!CAUTION]
>
> ## !!! THE LATEST VERSION CONTAINS BREAKING CHANGES
>
> **Pi-hole v6 has been entirely redesigned from the ground up and contains many breaking changes.**
>
> Environment variable names have changed, script locations may have changed.
>
> If you are using volumes to persist your configuration, be careful.
> Replacing any `v5` image *(`2024.07.0` and earlier)* with a `v6` image will result in updated configuration files. **These changes are irreversible**.
>
> Please read the README carefully before proceeding.
>
> https://docs.pi-hole.net/docker/

>

> [!NOTE] **Using Watchtower?**
>
> **See the Note on Watchtower at the bottom of this readme.**

> [!TIP] Some users have reported issues with using the `--privileged` flag on `2022.04` and above.
>
> TL;DR, don't use that mode, and be explicit with the permitted caps (if needed) instead.

Quick Start

Using Docker-compose:

1. Copy the below docker compose example and update as needed:

```
# More info at https://github.com/pi-hole/docker-pi-hole/ and https://docs.pi-hole.net/
services:
  pihole:
    container_name: pihole
    image: pihole/pihole:latest
    ports:
      # DNS Ports
      - "53:53/tcp"
      - "53:53/udp"
      # Default HTTP Port
      - "80:80/tcp"
      # Default HTTPs Port. FTL will generate a self-signed certificate
      - "443:443/tcp"
      # Uncomment the line below if you are using Pi-hole as your DHCP server
      #- "67:67/udp"
      # Uncomment the line below if you are using Pi-hole as your NTP server
      #- "123:123/udp"
    environment:
      # Set the appropriate timezone for your location
(https://en.wikipedia.org/wiki/List_of_tz_database_time_zones), e.g:
      TZ: 'Europe/London'
      # Set a password to access the web interface. Not setting one will result in a random
```

```
password being assigned
      FTLCONF_webserver_api_password: 'correct horse battery staple'
      # If using Docker's default `bridge` network setting the dns listening mode should be
set to 'all'
      FTLCONF_dns_listeningMode: 'all'
    # Volumes store your data between container upgrades
    volumes:
      # For persisting Pi-hole's databases and common configuration file
      - './etc-pihole:/etc/pihole'
      # Uncomment the below if you have custom dnsmasq config files that you want to persist.
Not needed for most starting fresh with Pi-hole v6. If you're upgrading from v5 you and have
used this directory before, you should keep it enabled for the first v6 container start to
allow for a complete migration. It can be removed afterwards. Needs environment variable
FTLCONF_misc_etc_dnsmasq_d: 'true'
      #- './etc-dnsmasq.d:/etc/dnsmasq.d'
    cap_add:
      # See https://github.com/pi-hole/docker-pi-hole#note-on-capabilities
      # Required if you are using Pi-hole as your DHCP server, else not needed
      - NET_ADMIN
      # Required if you are using Pi-hole as your NTP client to be able to set the host's
system time
      - SYS_TIME
      # Optional, if Pi-hole should get some more processing time
      - SYS_NICE
    restart: unless-stopped
```

2. Run `docker compose up -d` to build and start pi-hole (Syntax may be `docker-compose` on older systems).

> [!NOTE] Volumes are recommended for persisting data across container re-creations for updating images.

Automatic Ad List Updates

`cron` is baked into the container and will grab the newest versions of your lists and flush your logs. This happens once per week in the small hours of Sunday morning.

Running DHCP from Docker Pi-Hole

There are multiple different ways to run DHCP from within your Docker Pi-hole container, but it is slightly more advanced and one size does not fit all.

DHCP and Docker's multiple network modes are covered in detail on our docs site: Docker DHCP and Network Modes.

Configuration

It is recommended that you use environment variables to configure the Pi-hole docker container (more details below), however if you are persisting your `/etc/pihole` directory, you may choose instead to set them via the web interface or by directly editing `pihole.toml`.

> [!WARNING] Settings that are set via environment variables effectively become ***read-only***, meaning that you will not be able to change them in the web interface or CLI. This is to ensure a "single source of truth" on the config.
> If you later unset an environment variable, then FTL will revert to the default value for that setting.

Web interface password

To set a specific password for the web interface, use the environment variable `FTLCONF_webserver_api_password`.

If this variable is not detected and you have not already set one via `pihole setpassword` / `pihole-FTL --config webserver.api.password` inside the container, then a random password will be assigned on startup. This will be printed to the log. Run `docker logs pihole | grep random password` to find it.

To explicitly set no password, set `FTLCONF_webserver_api_password: ''`.

Recommended Environment Variables

Variable	Default	Value	Description
`TZ`	UTC	`<Timezone>`	Set your timezone to make sure logs rotate at local midnight instead of at UTC midnight.

Variable	Default	Value	Description
`FTLCONF_webserver_api_password`	random	`<Admin password>`	http://pi.hole/admin password. Run `docker logs pihole | grep random` to find your random password.
`FTLCONF_dns_upstreams`	`8.8.8.8;8.8.4.4`	IPs delimited by `;`	Upstream DNS server(s) for Pi-hole to forward queries to, separated by a semicolon. Supports non-standard ports with: `#[port number]`, e.g `127.0.0.1#5053;8.8.8.8;8.8.4.4`. Supports Docker service names and links instead of IPs, e.g `upstream0,upstream1` where `upstream0` and `upstream1` are the service names of or links to docker services. **Note:** The existence of this environment variable assumes this as the *sole* management of upstream DNS. Upstream DNS added via the web interface will be overwritten on container restart/recreation.

Optional Variables

Variable	Default	Value	Description
`TAIL_FTL_LOG`	`1`	`<0|1>`	Whether or not to output the FTL log when running the container. Can be disabled by setting the value to 0.

Variable	Default	Value	Description
`FTLCONF_[SETTING]`	unset	As per documentation	Customize pihole.toml with settings described in the API Documentation. Replace `.` with `_`, e.g for `dns.dnssec=true` use `FTLCONF_dns_dnssec: 'true'`. Array type configs should be delimited with `;`.
`PIHOLE_UID`	`1000`	Number	Overrides image's default pihole user id to match a host user id. **IMPORTANT**: id must not already be in use inside the container!
`PIHOLE_GID`	`1000`	Number	Overrides image's default pihole group id to match a host group id. **IMPORTANT**: id must not already be in use inside the container!
`WEBPASSWORD_FILE`	unset	`<Docker secret file>`	Set an Admin password using Docker secrets. If `FTLCONF_webserver_api_password` is set, `WEBPASSWORD_FILE` is ignored. If `FTLCONF_webserver_api_password` is empty, and `WEBPASSWORD_FILE` is set to a valid readable file, then `FTLCONF_webserver_api_password` will be set to the contents of `WEBPASSWORD_FILE`.

Advanced Variables

Variable	Default	Value	Description
`FTL_CMD`	`no-daemon`	`no-daemon -- <dnsmasq option>`	Customize dnsmasq startup options. e.g. `no-daemon --dns-forward-max 300` to increase max. number of concurrent dns queries on high load setups.

Variable	Default	Value	Description
`DNSMASQ_USER`	unset	`<pihole|root>`	Allows changing the user that FTLDNS runs as. Default: `pihole`, some systems such as Synology NAS may require you to change this to `root`. (See #963)
`ADDITIONAL_PACKAGES`	unset	Space separated list of APKs	HERE BE DRAGONS. Mostly for development purposes, this just makes it easier for those of us that always like to have whatever additional tools we need inside the container for debugging.
`FTLCONF_misc_etc_dnsmasq_d`	false	`true|false`	Load custom user configuration files from `/etc/dnsmasq.d/`

Here is a rundown of other arguments for your docker-compose / docker run.

Docker Arguments	Description
`-p <port>:<port>` **Recommended**	Ports to expose (53, 80, 443, 67), the bare minimum ports required for Pi-holes HTTP, HTTPS and DNS services.
`--restart=unless-stopped` **Recommended**	Automatically (re)start your Pi-hole on boot or in the event of a crash.
`-v $(pwd)/etc-pihole:/etc/pihole` **Recommended**	Volumes for your Pi-hole configs help persist changes across docker image updates.
`--net=host` *Optional*	Alternative to `-p <port>:<port>` arguments (Cannot be used at same time as `-p`) if you don't run any other web application. DHCP runs best with `--net=host`, otherwise your router must support dhcp-relay settings.
`--cap-add=NET_ADMIN` *Recommended*	Commonly added capability for DHCP, see Note on Capabilities below for other capabilities.
`--dns=n.n.n.n` *Optional*	Explicitly set container's DNS server. It is ***not recommended*** to set this to `localhost` / `127.0.0.1`.
`--env-file .env` *Optional*	File to store environment variables for docker replacing `-e key=value` settings. Here for convenience.

Tips and Tricks

- A good way to test things are working right is by loading this page: http://pi.hole/admin/

- Port conflicts? Stop your server's existing DNS / Web services.
  - Don't forget to stop your services from auto-starting again after you reboot.
  - Ubuntu users see below for more detailed information.
- Docker's default network mode `bridge` isolates the container from the host's network. This is a more secure setting, but requires setting the Pi-hole DNS option for *Interface listening behavior* to "Listen on all interfaces, permit all origins".
- If you're using a Red Hat based distribution with an SELinux Enforcing policy, add `:z` to line with volumes.

Installing on Ubuntu or Fedora

Modern releases of Ubuntu (17.10+) and Fedora (33+) include `systemd-resolved` which is configured by default to implement a caching DNS stub resolver. This will prevent pi-hole from listening on port 53. The stub resolver should be disabled with: `sudo sed -r -i.orig 's/#?DNSStubListener=yes/DNSStubListener=no/g' /etc/systemd/resolved.conf`.

This will not change the nameserver settings, which point to the stub resolver thus preventing DNS resolution. Change the `/etc/resolv.conf` symlink to point to `/run/systemd/resolve/resolv.conf`, which is automatically updated to follow the system's `netplan`: `sudo sh -c 'rm /etc/resolv.conf && ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf'`. After making these changes, you should restart systemd-resolved using `systemctl restart systemd-resolved`.

Once pi-hole is installed, you'll want to configure your clients to use it (see here). If you used the symlink above, your docker host will either use whatever is served by DHCP, or whatever static setting you've configured. If you want to explicitly set your docker host's nameservers you can edit the netplan(s) found at `/etc/netplan`, then run `sudo netplan apply`.

Example netplan:

```
network:
    ethernets:
        ens160:
            dhcp4: true
            dhcp4-overrides:
                use-dns: false
            nameservers:
                addresses: [127.0.0.1]
    version: 2
```

Note that it is also possible to disable `systemd-resolved` entirely. However, this can cause problems with name resolution in vpns (see bug report).
It also disables the functionality of netplan since systemd-resolved is used as the default renderer (see `man netplan`).

If you choose to disable the service, you will need to manually set the nameservers, for example by creating a new `/etc/resolv.conf`.

Users of older Ubuntu releases (circa 17.04) will need to disable dnsmasq.

Installing on Dokku

@Rikj000 has produced a guide to assist users installing Pi-hole on Dokku.

Docker tags and versioning

The primary docker tags are explained in the following table. Click here to see the full list of tags.

See GitHub Release notes to see the specific version of Pi-hole Core, Web, and FTL included in the release.

The Date-based (including incremented "Patch" versions) do not relate to any kind of semantic version number, rather a date is used to differentiate between the new version and the old version, nothing more.

Release notes will always contain full details of changes in the container, including changes to core Pi-hole components.

tag	description
`latest`	Always latest release
`2022.04.0`	Date-based release
`2022.04.1`	Second release in a given month
`development`	Similar to `latest`, but for the development branch (pushed occasionally)
`*beta`	Early beta releases of upcoming versions - here be dragons
`nightly`	Like `development` but pushed every night and pulls from the latest `development` branches of the core Pi-hole components (Pi-hole, web, FTL)

Upgrading, Persistence, and Customizations

The standard Pi-hole customization abilities apply to this docker, but with docker twists such as using docker volume mounts to map host stored file configurations over the container defaults. However, mounting these configuration files as read-only should be avoided. Volumes are also important to persist the configuration in case you have removed the Pi-hole container which is a typical docker upgrade pattern.

Upgrading / Reconfiguring

Do not attempt to upgrade (`pihole -up`) or reconfigure (`pihole -r`).

New images will be released for upgrades, upgrading by replacing your old container with a fresh upgraded image is the 'docker way'. Long-living docker containers are not the docker way since they aim to be portable and reproducible, why not re-create them often! Just to prove you can.

0. Read the release notes for both this Docker release and the Pi-hole release
    - This will help you avoid common problems due to any known issues with upgrading or newly required arguments or variables
    - We will try to put common break/fixes at the top of this readme too
1. Download the latest version of the image: `docker pull pihole/pihole`
2. Throw away your container: `docker rm -f pihole`
    - **Warning:** When removing your pihole container you may be stuck without DNS until step 3; `docker pull` before `docker rm -f` to avoid DNS interruption.
    - If you care about your data (logs/customizations), make sure you have it volume-mapped or it will be deleted in this step.
3. Start your container with the newer base image: `docker run <args> pihole/pihole` (`<args>` being your preferred run volumes and env vars)

Why is this style of upgrading good?

A couple reasons:

- Everyone is starting from the same base image which has been tested to known it works.
- No worrying about upgrading from A to B, B to C, or A to C is required when rolling out updates, it reduces complexity, and simply allows a 'fresh start' every time while preserving customizations with volumes.
- Basically I'm encouraging phoenix server principles for your containers.

To reconfigure Pi-hole you'll either need to use an existing container environment variables or, if there is no a variable for what you need, use the web UI or CLI commands.

Building the image locally

Occasionally you may need to try an alternative branch of one of the components (`core`, `web`, `ftl`). On bare metal you would run, for example, `pihole checkout core custombranchname`, however in Docker world we have disabled this command as it can cause unpredictable results.

The preferred method is to clone this repository and build the image locally with `./build.sh`.

Usage:

```
./build.sh [-l] [-f <ftl_branch>] [-c <core_branch>] [-w <web_branch>] [-p <padd_branch>] [-t <tag>] [use_cache]
```

Options:

- `-f <branch>` / `--ftlbranch <branch>`: Specify FTL branch (cannot be used in conjunction with `-l`)
- `-c <branch>` / `--corebranch <branch>`: Specify Core branch
- `-w <branch>` / `--webbranch <branch>`: Specify Web branch
- `-p <branch>` / `--paddbranch <branch>`: Specify PADD branch
- `-t <tag>` / `--tag <tag>`: Specify Docker image tag (default: `pihole:local`)
- `-l` / `--local`: Use locally built FTL binary (requires `src/pihole-FTL` file)
- `use_cache`: Enable caching (by default `--no-cache` is used)

If no options are specified, the following command will be executed:

```
docker buildx build src/. --tag pihole:local --no-cache
```

Pi-hole features

Here are some relevant wiki pages from Pi-hole's documentation.

We install all pihole utilities so the the built in pihole commands will work via `docker exec <container> <command>` like so:

- `docker exec pihole_container_name pihole updateGravity`
- `docker exec pihole_container_name pihole -w spclient.wg.spotify.com`
- `docker exec pihole_container_name pihole -wild example.com`

Customizations

The webserver and DNS service inside the container can be customized if necessary. Any configuration files you volume mount into `/etc/dnsmasq.d/` will be loaded by pihole-FTL when the container starts or restarts.

Note on Capabilities

Pi-hole's DNS core (FTL) expects to have the following capabilities available:

- `CAP_NET_BIND_SERVICE`: Allows FTLDNS binding to TCP/UDP sockets below 1024 (specifically DNS service on port 53)
- `CAP_NET_RAW`: use raw and packet sockets (needed for handling DHCPv6 requests, and verifying that an IP is not in use before leasing it)
- `CAP_NET_ADMIN`: modify routing tables and other network-related operations (in particular inserting an entry in the neighbor table to answer DHCP requests using unicast packets)
- `CAP_SYS_NICE`: FTL sets itself as an important process to get some more processing time if the latter is running low
- `CAP_CHOWN`: we need to be able to change ownership of log files and databases in case FTL is started as a different user than `pihole`
- `CAP_SYS_TIME`: FTL needs to be able to set the system time to update it using the Network Time Protocol (NTP) in the background

This image automatically grants those capabilities, if available, to the FTLDNS process, even when run as non-root.
By default, docker does not include the `NET_ADMIN` capability for non-privileged containers, and it is recommended to explicitly add it to the container using `--cap-add=NET_ADMIN`.
However, if DHCP and IPv6 Router Advertisements are not in use, it should be safe to skip it. For the most paranoid, it should even be possible to explicitly drop the `NET_RAW` capability to prevent FTLDNS from automatically gaining it.

Note on Watchtower

We have noticed that a lot of people use Watchtower to keep their Pi-hole containers up to date. For the same reason we don't provide an auto-update feature on a bare metal install, you *should not* have a system automatically update your Pi-hole container. Especially unattended. As much as we try to ensure nothing will go wrong, sometimes things do go wrong - and you need to set aside time to *manually* pull and update to the version of the container you wish to run. The upgrade process should be along the lines of:

- **Important**: Read the release notes. Sometimes you will need to make changes other than just updating the image.
- Pull the new image.
- Stop and *remove* the running Pi-hole container
  - If you care about your data (logs/customizations), make sure you have it volume-mapped or it will be deleted in this step.
- Recreate the container using the new image.

Pi-hole is an integral part of your network, don't let it fall over because of an unattended update in the middle of the night.

Bookstack

Pi-hole website_

Docker Compose

```
services:
  mysql:
    image: mysql:8.3
    environment:
    - MYSQL_ROOT_PASSWORD=secret
    - MYSQL_DATABASE=bookstack
    - MYSQL_USER=bookstack
    - MYSQL_PASSWORD=secret
    volumes:
    - mysql-data:/var/lib/mysql

  bookstack:
    image: solidnerd/bookstack:25.2.0
    depends_on:
    - mysql
    environment:
    - DB_HOST=mysql:3306
    - DB_DATABASE=bookstack
    - DB_USERNAME=bookstack
    - DB_PASSWORD=secret
    #set the APP_ to the URL of bookstack without without a trailing slash
APP_URL=https://example.com
    - APP_URL=http://example.com
    # APP_KEY is used for encryption where needed, so needs to be persisted to
    # preserve decryption abilities.
    # Can run `php artisan key:generate` to generate a key
    - APP_KEY=SomeRandomStringWith32Characters
    volumes:
    - uploads:/var/www/bookstack/public/uploads
    - storage-uploads:/var/www/bookstack/storage/uploads
    ports:
    - "8080:8080"
```

```
volumes:
 mysql-data:
 uploads:
 storage-uploads:
```

DocuSeal

Pi-hole website_

Let's create a folder for our Docuseal install. I like to create a parent level folder for 'docker', then put each docker application within that folder.

```
mkdir -p docker/docuseal
```

Now we'll move into that folder, and create a new compose.yaml file where we'll define how we want Docuseal to be configured.

```
cd docker/docuseal
```

```
nano compose.yaml
```

Inside the new compose.yaml file, we need to add the block of yaml code below.

```
services:
  docuseal:
    depends_on:
      postgres:
        condition: service_healthy
    image: docuseal/docuseal:latest
    ports:
      - 3000:3000
    volumes:
      - ./data:/data
    environment:
      #- FORCE_SSL=${HOST}
      - DATABASE_URL=postgresql://postgres:this-needs-to-be-a-long-strong-
password@postgres:5432/docuseal


  postgres:
    image: postgres:15
    volumes:
      - './pg_data:/var/lib/postgresql/data'
    environment:
      POSTGRES_USER: postgres
```

```
      POSTGRES_PASSWORD: this-needs-to-be-a-long-strong-password   # <-- should match the
password in the DATABASE_URL above
      POSTGRES_DB: docuseal
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5
```

In the file, you may need to make a change to the port mapping. `3000` is a very common port for many applications. If you already have an application using port `3000` on your host machine, you can change the port number on the left side of the port mapping. In my case I changed it to `8022`, so that line looked like

`- 8022:3000`

Next, in two places, there is a password being set. First in the DATABASE_URL environment variable in the docuseal section. I have it defaulted to "this-needs-to-be-a-long-strong-password", but you should change this to an actual long, strong password. Once set, you need to copy it, and paste it in as the value for POSTGRES_PASSWORD in the postgres section. Those two passwords must match for the application to function.

Finally, I have commented out the environment variable line for - FORCE_SSL=${HOST}, but if you are setting this up without a reverse proxy, then in the original file found here, you will see a 'caddy' section as a built in reverse proxy. You will need to uncomment that line by removing the hash tag, and then make sure the 'caddy' section is added back.

Once you've made those changes, you can save your compose.yaml file with CTRL + O, then press Enter to confrim, and use CTRL + X to exit.

We are ready to start up the application. We can do that with the command:

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

Stirling-pdf

Pi-hole website_

Docker Compose

```
services:
  stirling-pdf:
    image: docker.stirlingpdf.com/stirlingtools/stirling-pdf
    container_name: "Sterling-pdf"
    ports:
      - '8981:8080'   # change the left side port if 8080 is already in use on your host
machine
    volumes:
      - ./trainingData:/usr/share/tessdata #Required for extra OCR languages
      - ./extraConfigs:/configs
      - /location/of/customFiles:/customFiles/
      - ./logs:/logs/
    environment:
      - DOCKER_ENABLE_SECURITY=false
      - INSTALL_BOOK_AND_ADVANCED_HTML_OPS=false
      - LANGS=en_US    # change this to your preferred language code
```