simple-login / **app** Public

The SimpleLogin back-end

simplelogin.io

AGPL-3.0 License

1.7k stars **171** forks

Star Notifications

Code Issues 36 Pull requests 7 Discussions Actions ···

master Go to file

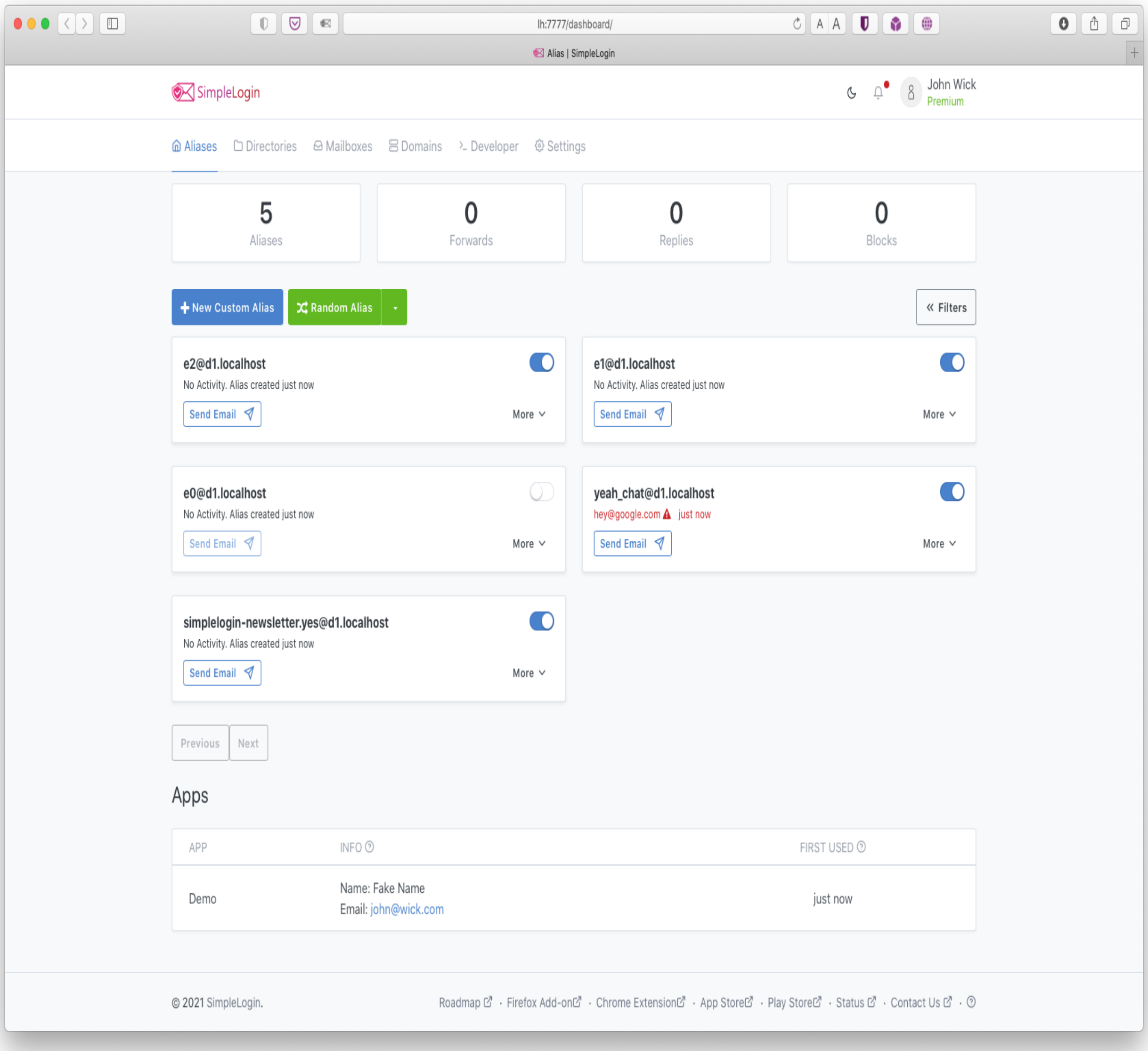
nguyenkims black format ··· 3 minutes ago **3,640**

View code

README.md

SimpleLogin | Protect your online identity with email alias

Chrome Extension 4.96/5 Firefox Add-On 5/5 license AGPL-3.0 Follow @simple_login 1.9k



APP	INFO	FIRST USED
Demo	Name: Fake Name Email: john@wick.com	just now

Your email address is your **online identity**. When you use the same email address everywhere, you can be easily tracked. More information on https://simplelogin.io

This README contains instructions on how to self host SimpleLogin.

Once you have your own SimpleLogin instance running, you can change the `API URL` in SimpleLogin's Chrome/Firefox extension, Android/iOS app to your server.

SimpleLogin roadmap is at https://github.com/simple-login/app/projects/1 and our forum at https://github.com/simple-login/app/discussions, feel free to submit new ideas or vote on features.

Prerequisites

- a Linux server (either a VM or dedicated server). This doc shows the setup for Ubuntu 18.04 LTS but the steps could be adapted for other popular Linux distributions. As most of components run as Docker container and Docker can be a bit heavy, having at least 2 GB of RAM is recommended. The server needs to have the port 25 (email), 80, 443 (for the webapp), 22 (so you can ssh into it) open.

- a domain that you can config the DNS. It could be a sub-domain. In the rest of the doc, let's say it's `mydomain.com` for the email and `app.mydomain.com` for SimpleLogin webapp. Please make sure to replace these values by your domain name whenever they appear in the doc. A trick we use is to download this README file on your computer and replace all `mydomain.com` occurrences by your domain.

Except for the DNS setup that is usually done on your domain registrar interface, all the below steps are to be done on your server. The commands are to run with `bash` (or any bash-compatible shell like `zsh`) being the shell. If you use other shells like `fish`, please make sure to adapt the commands.

Some utility packages

These packages are used to verify the setup. Install them by:

```
sudo apt update && sudo apt install -y dnsutils
```

Create a directory to store SimpleLogin data:

```
mkdir sl
mkdir sl/pgp # to store PGP key
mkdir sl/db # to store database
mkdir sl/upload # to store quarantine emails
```

DKIM

From Wikipedia https://en.wikipedia.org/wiki/DomainKeys_Identified_Mail

> DomainKeys Identified Mail (DKIM) is an email authentication method designed to detect forged sender addresses in emails (email spoofing), a technique often used in phishing and email spam.

Setting up DKIM is highly recommended to reduce the chance your emails ending up in the recipient's Spam folder.

First you need to generate a private and public key for DKIM:

```
openssl genrsa -out dkim.key 1024
openssl rsa -in dkim.key -pubout -out dkim.pub.key
```

You will need the files `dkim.key` and `dkim.pub.key` for the next steps.

For email gurus, we have chosen 1024 key length instead of 2048 for DNS simplicity as some registrars don't play well with long TXT record.

DNS

Please note that DNS changes could take up to 24 hours to propagate. In practice, it's a lot faster though (~1 minute or so in our test). In DNS setup, we usually use domain with a trailing dot (`.`) at the end to to force using absolute domain.

MX record

Create a **MX record** that points `mydomain.com.` to `app.mydomain.com.` with priority 10.

To verify if the DNS works, the following command

```
dig @1.1.1.1 mydomain.com mx
```

should return:

```
mydomain.com.   3600    IN      MX      10 app.mydomain.com.
```

A record

An **A record** that points `app.mydomain.com.` to your server IP. If you are using CloudFlare, we recommend to disable the "Proxy" option. To verify, the following command

```
dig @1.1.1.1 app.mydomain.com a
```

should return your server IP.

DKIM

Set up DKIM by adding a TXT record for `dkim._domainkey.mydomain.com.` with the following value:

```
v=DKIM1; k=rsa; p=PUBLIC_KEY
```

with `PUBLIC_KEY` being your `dkim.pub.key` but

- remove the `-----BEGIN PUBLIC KEY-----` and `-----END PUBLIC KEY-----`
- join all the lines on a single line.

For example, if your `dkim.pub.key` is

```
-----BEGIN PUBLIC KEY-----
ab
cd
ef
gh
-----END PUBLIC KEY-----
```

then the `PUBLIC_KEY` would be `abcdefgh`.

You can get the `PUBLIC_KEY` by running this command:

```
sed "s/-----BEGIN PUBLIC KEY-----/v=DKIM1; k=rsa; p=/g" dkim.pub.key | sed 's
```

To verify, the following command

```
dig @1.1.1.1 dkim._domainkey.mydomain.com txt
```

should return the above value.

SPF

From Wikipedia https://en.wikipedia.org/wiki/Sender_Policy_Framework

> Sender Policy Framework (SPF) is an email authentication method designed to detect forging sender addresses during the delivery of the email

Similar to DKIM, setting up SPF is highly recommended. Add a TXT record for `mydomain.com.` with the value:

```
v=spf1 mx ~all
```

What it means is only your server can send email with `@mydomain.com` domain. To verify, the following command

```
dig @1.1.1.1 mydomain.com txt
```

should return the above value.

DMARC

From Wikipedia https://en.wikipedia.org/wiki/DMARC

> It (DMARC) is designed to give email domain owners the ability to protect their domain from unauthorized use, commonly known as email spoofing

Setting up DMARC is also recommended. Add a TXT record for `_dmarc.mydomain.com.` with the following value

```
v=DMARC1; p=quarantine; adkim=r; aspf=r
```

This is a `relaxed` DMARC policy. You can also use a more strict policy with `v=DMARC1; p=reject; adkim=s; aspf=s` value.

To verify, the following command

```
dig @1.1.1.1 _dmarc.mydomain.com txt
```

should return the set value.

For more information on DMARC, please consult https://tools.ietf.org/html/rfc7489

Docker

Now the boring DNS stuffs are done, let's do something more fun!

If you don't already have Docker installed on your server, please follow the steps on Docker CE for Ubuntu to install Docker.

You can also install Docker using the docker-install script which is

```
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

Prepare the Docker network

This Docker network will be used by the other Docker containers run in the next steps. Later, we will setup Postfix to authorize this network.

```
sudo docker network create -d bridge \
    --subnet=240.0.0.0/24 \
    --gateway=240.0.0.1 \
    sl-network
```

Postgres

This section creates a Postgres database using Docker.

If you already have a Postgres database in use, you can skip this section and just copy the database configuration (i.e. host, port, username, password, database name) to use in the next sections.

Run a Postgres Docker container as your Postgres database server. Make sure to replace `myuser` and `mypassword` with something more secret.

```
docker run -d \
    --name sl-db \
    -e POSTGRES_PASSWORD=mypassword \
    -e POSTGRES_USER=myuser \
    -e POSTGRES_DB=simplelogin \
    -p 127.0.0.1:5432:5432 \
    -v $(pwd)/sl/db:/var/lib/postgresql/data \
    --restart always \
    --network="sl-network" \
    postgres:12.1
```

To test whether the database operates correctly or not, run the following command:

```
docker exec -it sl-db psql -U myuser simplelogin
```

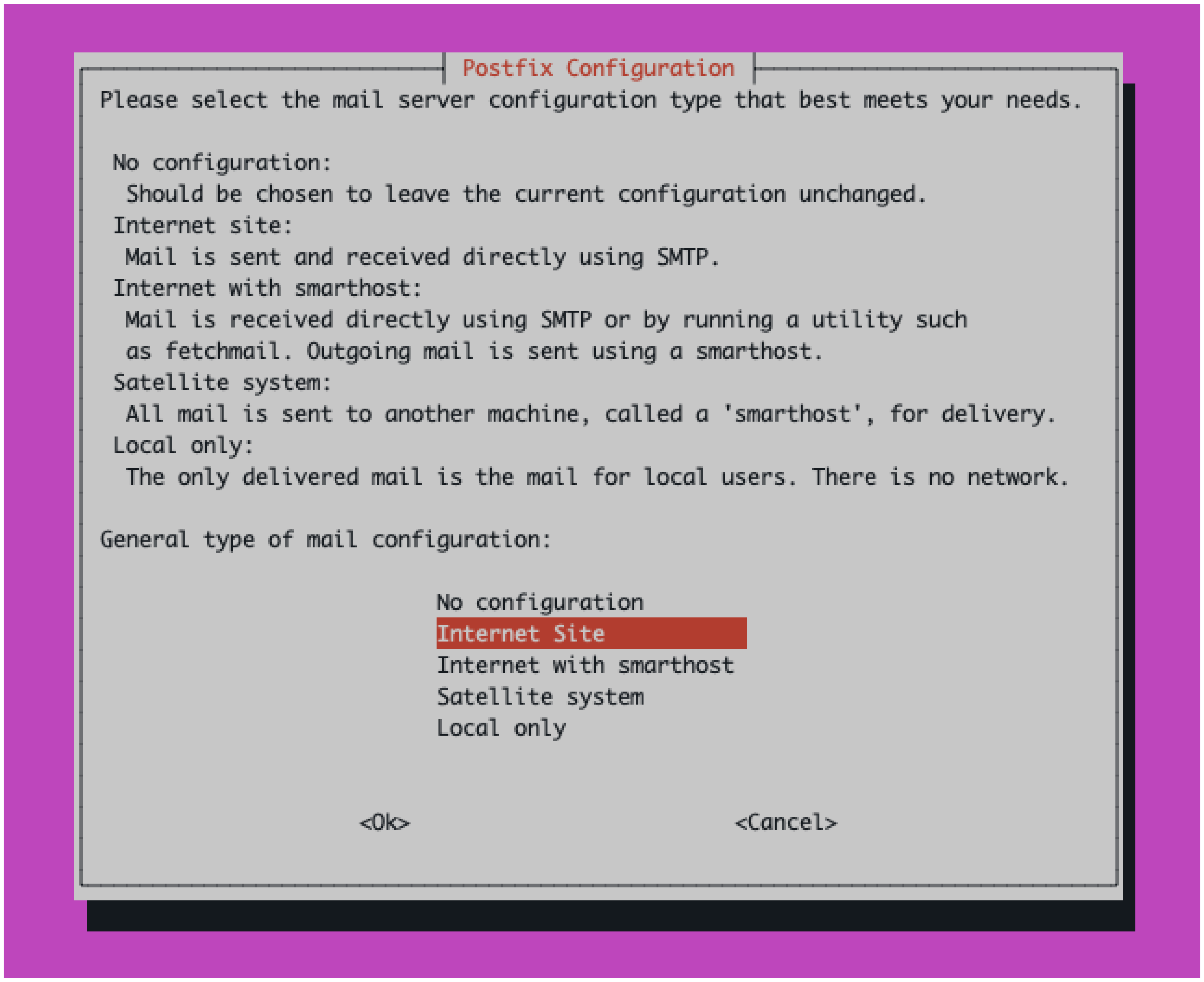
you should be logged in the postgres console. Type `exit` to exit postgres console.

Postfix

Install `postfix` and `postfix-pgsql`. The latter is used to connect Postfix and the Postgres database in the next steps.

```
sudo apt-get install -y postfix postfix-pgsql -y
```

Choose "Internet Site" in Postfix installation window then keep using the proposed value as *System mail name* in the next window.



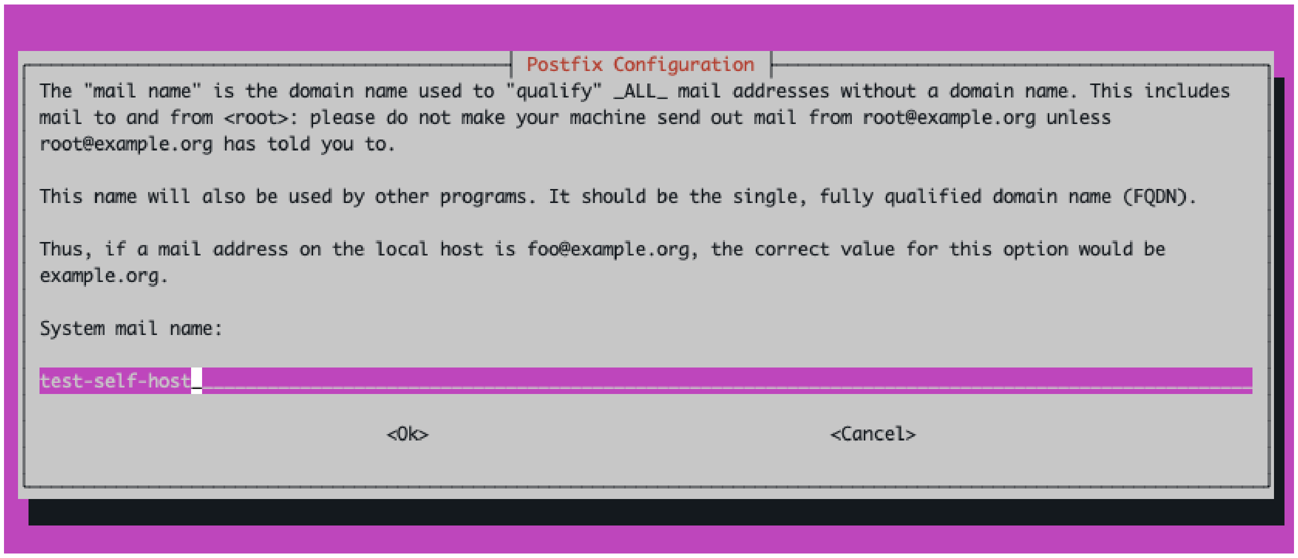
```
Postfix Configuration
Please select the mail server configuration type that best meets your needs.

 No configuration:
  Should be chosen to leave the current configuration unchanged.
 Internet site:
  Mail is sent and received directly using SMTP.
 Internet with smarthost:
  Mail is received directly using SMTP or by running a utility such
  as fetchmail. Outgoing mail is sent using a smarthost.
 Satellite system:
  All mail is sent to another machine, called a 'smarthost', for delivery.
 Local only:
  The only delivered mail is the mail for local users. There is no network.

General type of mail configuration:

                         No configuration
                         Internet Site
                         Internet with smarthost
                         Satellite system
                         Local only


                   <Ok>                         <Cancel>
```



```
Postfix Configuration
The "mail name" is the domain name used to "qualify" _ALL_ mail addresses without a domain name. This includes
mail to and from <root>: please do not make your machine send out mail from root@example.org unless
root@example.org has told you to.

This name will also be used by other programs. It should be the single, fully qualified domain name (FQDN).

Thus, if a mail address on the local host is foo@example.org, the correct value for this option would be
example.org.

System mail name:

test-self-host

              <Ok>                                  <Cancel>
```

Replace `/etc/postfix/main.cf` with the following content. Make sure to replace `mydomain.com` by your domain.

```
# POSTFIX config file, adapted for SimpleLogin
smtpd_banner = $myhostname ESMTP $mail_name (Ubuntu)
biff = no

# appending .domain is the MUA's job.
append_dot_mydomain = no

# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h

readme_directory = no

# See http://www.postfix.org/COMPATIBILITY_README.html -- default to 2 on
# fresh installs.
compatibility_level = 2

# TLS parameters
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache
smtp_tls_security_level = may
smtpd_tls_security_level = may

# See /usr/share/doc/postfix/TLS_README.gz in the postfix-doc package for
# information on enabling SSL in the smtp client.

alias_maps = hash:/etc/aliases
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128 240.0.0.0/24

# Set your domain here
mydestination =
myhostname = app.mydomain.com
mydomain = mydomain.com
myorigin = mydomain.com

relay_domains = pgsql:/etc/postfix/pgsql-relay-domains.cf
transport_maps = pgsql:/etc/postfix/pgsql-transport-maps.cf

# HELO restrictions
smtpd_delay_reject = yes
smtpd_helo_required = yes
smtpd_helo_restrictions =
    permit_mynetworks,
    reject_non_fqdn_helo_hostname,
    reject_invalid_helo_hostname,
    permit

# Sender restrictions:
smtpd_sender_restrictions =
    permit_mynetworks,
    reject_non_fqdn_sender,
    reject_unknown_sender_domain,
    permit

# Recipient restrictions:
smtpd_recipient_restrictions =
   reject_unauth_pipelining,
   reject_non_fqdn_recipient,
   reject_unknown_recipient_domain,
   permit_mynetworks,
   reject_unauth_destination,
   reject_rbl_client zen.spamhaus.org,
   reject_rbl_client bl.spamcop.net,
   permit
```

Create the `/etc/postfix/pgsql-relay-domains.cf` file with the following content. Make sure that the database config is correctly set, replace `mydomain.com` with your domain, update 'myuser' and 'mypassword' with your postgres credentials.

```
# postgres config
hosts = localhost
user = myuser
password = mypassword
dbname = simplelogin

query = SELECT domain FROM custom_domain WHERE domain='%s' AND
verified=true
    UNION SELECT '%s' WHERE '%s' = 'mydomain.com' LIMIT 1;
```

Create the `/etc/postfix/pgsql-transport-maps.cf` file with the following content. Again, make sure that the database config is correctly set, replace `mydomain.com` with your domain, update 'myuser' and 'mypassword' with your postgres credentials.

```
# postgres config
hosts = localhost
user = myuser
password = mypassword
dbname = simplelogin
```

```
# forward to smtp:127.0.0.1:20381 for custom domain AND email domain
query = SELECT 'smtp:127.0.0.1:20381' FROM custom_domain WHERE domain =
'%s' AND verified=true
    UNION SELECT 'smtp:127.0.0.1:20381' WHERE '%s' = 'mydomain.com' LIMIT
1;
```

Finally, restart Postfix

```
sudo systemctl restart postfix
```

Run SimpleLogin Docker containers

To run SimpleLogin, you need a config file at `~/simplelogin.env`. Below is an example that you can use right away, make sure to

- replace `mydomain.com` by your domain,
- set `FLASK_SECRET` to a secret string,
- update 'myuser' and 'mypassword' with your database credentials used in previous step.

All possible parameters can be found in config example. Some are optional and are commented out by default. Some have "dummy" values, fill them up if you want to enable these features (Paddle, AWS, etc).

```
# WebApp URL
URL=http://app.mydomain.com

# domain used to create alias
EMAIL_DOMAIN=mydomain.com

# transactional email is sent from this email address
SUPPORT_EMAIL=support@mydomain.com

# custom domain needs to point to these MX servers
EMAIL_SERVERS_WITH_PRIORITY=[(10, "app.mydomain.com.")]

# By default, new aliases must end with ".{random_word}". This is to avoid a
# this option doesn't make sense in self-hosted. Set this variable to disable
DISABLE_ALIAS_SUFFIX=1

# the DKIM private key used to compute DKIM-Signature
DKIM_PRIVATE_KEY_PATH=/dkim.key

# DB Connection
```

```
DB_URI=postgresql://myuser:mypassword@sl-db:5432/simplelogin

FLASK_SECRET=put_something_secret_here

GNUPGHOME=/sl/pgp

LOCAL_FILE_UPLOAD=1
```

Before running the webapp, you need to prepare the database by running the migration:

```
docker run --rm \
    --name sl-migration \
    -v $(pwd)/sl:/sl \
    -v $(pwd)/sl/upload:/code/static/upload \
    -v $(pwd)/dkim.key:/dkim.key \
    -v $(pwd)/dkim.pub.key:/dkim.pub.key \
    -v $(pwd)/simplelogin.env:/code/.env \
    --network="sl-network" \
    simplelogin/app:3.4.0 flask db upgrade
```

This command could take a while to download the `simplelogin/app` docker image.

Init data

```
docker run --rm \
    --name sl-init \
    -v $(pwd)/sl:/sl \
    -v $(pwd)/simplelogin.env:/code/.env \
    -v $(pwd)/dkim.key:/dkim.key \
    -v $(pwd)/dkim.pub.key:/dkim.pub.key \
    --network="sl-network" \
    simplelogin/app:3.4.0 python init_app.py
```

Now, it's time to run the `webapp` container!

```
docker run -d \
    --name sl-app \
    -v $(pwd)/sl:/sl \
    -v $(pwd)/sl/upload:/code/static/upload \
    -v $(pwd)/simplelogin.env:/code/.env \
    -v $(pwd)/dkim.key:/dkim.key \
    -v $(pwd)/dkim.pub.key:/dkim.pub.key \
    -p 127.0.0.1:7777:7777 \
    --restart always \
    --network="sl-network" \
    simplelogin/app:3.4.0
```

Next run the `email handler`

```
docker run -d \
    --name sl-email \
    -v $(pwd)/sl:/sl \
    -v $(pwd)/sl/upload:/code/static/upload \
    -v $(pwd)/simplelogin.env:/code/.env \
    -v $(pwd)/dkim.key:/dkim.key \
    -v $(pwd)/dkim.pub.key:/dkim.pub.key \
    -p 127.0.0.1:20381:20381 \
    --restart always \
    --network="sl-network" \
    simplelogin/app:3.4.0 python email_handler.py
```

And finally the `job runner`

```
docker run -d \
    --name sl-job-runner \
    -v $(pwd)/sl:/sl \
    -v $(pwd)/sl/upload:/code/static/upload \
    -v $(pwd)/simplelogin.env:/code/.env \
    -v $(pwd)/dkim.key:/dkim.key \
    -v $(pwd)/dkim.pub.key:/dkim.pub.key \
    --restart always \
    --network="sl-network" \
    simplelogin/app:3.4.0 python job_runner.py
```

Nginx

Install Nginx and make sure to replace `mydomain.com` by your domain

```
sudo apt-get install -y nginx
```

Then, create `/etc/nginx/sites-enabled/simplelogin` with the following lines:

```
server {
    server_name  app.mydomain.com;

    location / {
        proxy_pass http://localhost:7777;
    }
}
```

Reload Nginx with the command below

```
sudo systemctl reload nginx
```

At this step, you should also setup the SSL for Nginx. Certbot can be a good option if you want a free SSL certificate.

Enjoy!

If all the above steps are successful, open http://app.mydomain.com/ and create your first account!

By default, new accounts are not premium so don't have unlimited alias. To make your account premium, please go to the database, table "users" and set "lifetime" column to "1" or "TRUE".

You don't have to pay anything to SimpleLogin to use all its features. If you like the project, you can make a donation on our Patreon page at https://www.patreon.com/simplelogin

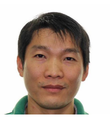
Misc

The above self-hosting instructions correspond to a freshly Ubuntu server and doesn't cover all possible server configuration. Below are pointers to different topics:

- Troubleshooting
- Enable SSL
- UFW - uncomplicated firewall
- SES - Amazon Simple Email Service
- Upgrade existing SimpleLogin installation
- Enforce SPF
- Postfix TLS

❤️ Contributors

Thanks go to these wonderful people:

Dung Nguyen Van	Giuseppe Federico	Ninh Dinh	 Tung Nguyen V. N.	Son Nguyen Kim	Raymond Nook	Sibren Vasse	Sylvia van Os

Releases

15 tags

Sponsor this project

patreon.com/**simplelogin**

opencollective.com/**simplelogin**

https://www.paypal.me/RealSimpleLogin

Packages

No packages published

Contributors 25

+ 14 contributors

Languages

● **CSS** 36.3% ● **Python** 36.2% ● **JavaScript** 16.6% ● **HTML** 10.4% ● **Jinja** 0.3% ● **Shell** 0.2%