Nginx and Let’s Encrypt with Docker in Less Than 5 Minutes

pentacent.medium.com/nginx-and-lets-encrypt-with-docker-in-less-than-5-minutes-b4b8a60d3a71

19 December 2018

Philipp
Sep 28, 2018

·

4 min read

·

Getting Nginx to run with Let’s Encrypt in a docker-compose environment is trickier than you’d think …

The other day, I wanted to *quickly* launch an nginx server with Let’s Encrypt certificates. I expected the task to be easy and straightforward. Turns out: I was wrong, it took a significant amount of time and it’s quite a bit more complicated.

Of course, in the grand scheme of things, it *is* pretty straightforward. But there are a couple of details you need to be aware of. The goal of this guide is to help you build a docker-compose setup that runs nginx in one container and a service for obtaining and renewing HTTPS certificates in another. **Whether you're using nginx as a proxy for your web app or just for serving static files, this guide is for you.**

> TL;DR: The full code from this guide is available on GitHub.

Quick Reminder: What is docker-compose?

is a tool for defining containers and running them. It's a great choice when you have multiple interdependent containers but you don't need a full-blown container cluster like Kubernetes.

The official documentation puts it like this:

> With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration

This guide requires docker-compose. If you don't have it yet, take a look at the installation instructions and get it.
Hint: If you're installing docker-compose on CoreOS, it needs to go into `/opt/bin` *instead of* `/usr/local/bin`.

The Setup

Official images of nginx and an automated build of certbot, the EFF's tool for obtaining Let's Encrypt certificates, are available in the Docker library.

Let's begin with a basic `docker-compose.yml` configuration file that defines containers for both images:

```
version: '3'services:  nginx:    image: nginx:1.15-alpine    ports:      - "80:80"
- "443:443"    volumes:      - ./data/nginx:/etc/nginx/conf.d  certbot:    image:
certbot/certbot
```

Here is a simple nginx configuration that redirects all requests to HTTPS. The second `server` definition sets up a proxy to *example.org* for demonstration purposes. This is where you would add your own configuration for proxying requests to your app or serving local files.

Save this file as `data/nginx/app.conf` alongside `docker-compose.yml.` **Change `example.org` in both occurrences of `server_name` to your domain name.**

```
server {    listen 80;    server_name example.org;    location / {        return
301     }    }

server {
    listen 443 ssl;
    server_name example.org;

        location / {        proxy_pass  #for demo purposes    }}
```

If you would try to run `docker-compose up` now, nginx would fail to start because there is no certificate. We need to make some adjustments.

Linking up nginx and certbot

Let's Encrypt performs domain validation by requesting a well-known URL from a domain. If it receives a certain response (the "challenge"), the domain is considered validated. This is similar to how Google Search Console establishes ownership of a website. The response data is provided by certbot, so we need a way for the nginx container to serve files from certbot.

First of all, we need two shared Docker volumes. One for the validation challenges, the other for the actual certificates.

Add this to the `volumes` list of the `nginx` section in `docker-compose.yml`.

```
- ./data/certbot/conf:/etc/letsencrypt- ./data/certbot/www:/var/www/certbot
```

And this is the counterpart that needs to go in the `certbot` section:

```
volumes:  - ./data/certbot/conf:/etc/letsencrypt  -
./data/certbot/www:/var/www/certbot
```

Now we can make nginx serve the challenge files from certbot! Add this to the first (port 80) section of our nginx configuration (`data/nginx/app.conf`):

After that, we need to reference the HTTPS certificates. Add the soon-to-be-created certificate and its private key to the second `server` section (port 443). **Make sure to once again replace `example.org` with your domain name.**

```
ssl_certificate
/etc/letsencrypt/live/example.org/fullchain.pem;ssl_certificate_key
/etc/letsencrypt/live/example.org/privkey.pem;
```

And while we're at it: The folks at Let's Encrypt maintain best-practice HTTPS configurations for nginx. Let's also add them to our config file. This will score you a straight A in the SSL Labs test!

```
include /etc/letsencrypt/options-ssl-nginx.conf;ssl_dhparam /etc/letsencrypt/ssl-
dhparams.pem;
```

The Chicken or the Egg?

Now for the tricky part. We need nginx to perform the Let's Encrypt validation But nginx won't start if the certificates are missing.

So what do we do? Create a dummy certificate, start nginx, delete the dummy and request the real certificates.
Luckily, you don't have to do all this manually, I have created a convenient script for this.

Download the script to your working directory as `init-letsencrypt.sh:`

```
curl -L  > init-letsencrypt.sh
```

Edit the script to add in your domain(s) and your email address. If you've changed the directories of the shared Docker volumes, make sure you also adjust the `data_path` variable as well.

Then run `chmod +x init-letsencrypt.sh` and `sudo ./init-letsencrypt.sh`.

Automatic Certificate Renewal

Last but not least, we need to make sure our certificate is renewed when it's about to expire. The certbot image doesn't do that automatically but we can change that!

Add the following to the `certbot` section of `docker-compose.yml`:

```
entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h &
wait $${!}; done;'"
```

This will check if your certificate is up for renewal every 12 hours as recommended by Let's Encrypt.

In the `nginx` section, you need to make sure that nginx reloads the newly obtained certificates:

```
command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s reload; done &
nginx -g \"daemon off;\"'"
```

This makes nginx reload its configuration (and certificates) every six hours in the background and launches nginx in the foreground.

Docker-compose Me Up!

Everything is in place now. The initial certificates have been obtained and our containers are ready to launch. Simply run `docker-compose up` and enjoy your HTTPS-secured website or app.

Did you find this guide helpful?

Are you interested in best practices for application deployment? Make sure to subscribe to my newsletter!