HTTPS using Nginx and Let's encrypt in Docker

6 Oct 2021 • 5 min read

This post is free for all to read thanks to the investment Mindsers Blog's subscribers have made in our independent publication. If this work is meaningful to you, I invite you to become a subscriber today.

As it is a really common task, this post will guide you through with a step-by-step process to protect your website (and your users) using HTTPS. The specific part here is that we will do this in a docker environment.

In this post, I will use Docker Compose to make the tutorial simpler and because I like the infrastructure as code movement.

Nginx as a server

To be able to use nginx as a server for any of our projects, we have to create a Docker Compose service for it. Docker will handle the download of the corresponding image and all the other tasks we used to do manually without Docker.

```
version: '3'

services:
  webserver:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
```

At anytime during the tutorial, you can run `docker compose up` to start the environment and see if everything goes well.

```
  webserver:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
    restart: always
```

As I want the server to be always up and running, I tell Docker that it should take care of restarting the "webserver" service when it unexpectedly shuts down.

```
version: '3'

services:
  webserver:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
    restart: always
    volumes:
      - ./nginx/conf/:/etc/nginx/conf.d/:ro
```

The ultimate goal of our installation isn't to serve the default welcome page of nginx. So, we need a way to update the nginx configuration and declare our website.

To accomplish that, we use the "volumes" feature of Docker. This means we map the folder located at `/etc/nginx/conf.d/` from the docker container to a folder located at `./nginx/conf/` on our local machine. Every file we add, remove or update into this folder locally will be updated into the container.

Note that I add a `:ro` at the end of the volume declaration. `ro` means "read-only". The container will never have the right to update a file into this folder. It is not a big deal, but consider it as a best practice. It can avoid wasting a few precious hours of debugging.

Now update the Docker Compose file as below:

```
version: '3'

services:
  webserver:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
    restart: always
    volumes:
      - ./nginx/conf/:/etc/nginx/conf.d/:ro
      - ./certbot/www:/var/www/certbot/:ro
```

And add the following configuration file into your `./nginx/conf/` local folder. Do not forget to update using your own data.

```
server {
    listen 80;
    listen [::]:80;

    server_name example.org www.example.org;
    server_tokens off;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://example.org$request_uri;
    }
}
```

`location /` block.

But the specificity here is the other `location` block. It serves the files Certbot need to authenticate our server and to create the HTTPS certificate for it.

Basically, we say "always redirect to HTTPS except for the `/.well-know/acme-challenge/` route".

We can now reload nginx by doing a rough `docker compose restart` or if you want to avoid service interruptions (even for a couple of seconds) reload it inside the container using `docker compose exec webserver nginx -s reload`.

Create the certificate using Certbot

For now, nothing will be shown because nginx keeps redirecting you to a `443` port that's not handled by nginx yet. But everything is fine. We only want Certbot to be able to authenticate our server.

To do so, we need to use the docker image for certbot and add it as a service to our Docker Compose project.

```
version: '3'

services:
  webserver:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
    restart: always
    volumes:
      - ./nginx/conf/:/etc/nginx/conf.d/:ro
      - ./certbot/www:/var/www/certbot/:ro
  certbot:
    image: certbot/certbot:latest
    volumes:
      - ./certbot/www/:/var/www/certbot/:rw
```

We now have two services, one for nginx and one for Certbot. You might have noticed they have declared the same volume. It is meant to make them communicate together.

Certbot will write its files into `./certbot/www/` and nginx will serve them on port `80` to every user asking for `/.well-know/acme-challenge/`. That's how Certbot can authenticate our server.

Note that for Certbot we used `:rw` which stands for "read and write" at the end of the volume declaration. If you don't, it won't be able to write into the folder and authentication will fail.

You can now test that everything is working by running `docker compose run --rm  certbot certonly --webroot --webroot-path /var/www/certbot/ --dry-run -d example.org`. You should get a success message like "The dry run was successful".

Now that we can create certificates for the server, we want to use them in nginx to handle secure connections with end users' browsers.

Certbot create the certificates in the `/etc/letsencrypt/` folder. Same principle as for the webroot, we'll use volumes to share the files between containers.

```
version: '3'

services:
  webserver:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
    restart: always
    volumes:
      - ./certbot/www/:/var/www/certbot/:ro
```

```
      - ./certbot/conf/:/etc/letsencrypt/:rw
```

Restart your container using `docker compose restart`. Nginx should now have access to the folder where Certbot creates certificates.

However, this folder is empty right now. Re-run Certbot without the `--dry-run` flag to fill the folder with certificates:

```
$ docker compose run --rm  certbot certonly --webroot --webroot-path /var/www/certbot/ -d example.org
```

Since we have those certificates, the piece left is the `443` configuration on nginx.

```
server {
    listen 80;
    listen [::]:80;

    server_name example.org www.example.org;
    server_tokens off;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://example.org$request_uri;
    }
}

server {
    listen 443 default_server ssl http2;
    listen [::]:443 ssl http2;

    server_name example.org;

    ssl_certificate /etc/nginx/ssl/live/example.org/fullchain.pem;
    ssl_certificate_key /etc/nginx/ssl/live/example.org/privkey.pem;

    location / {
       # ...
    }
}
```

Reloading the nginx server now will make it able to handle secure connection using HTTPS. Nginx is using the certificates and private keys from the Certbot volumes.

Renewing the certificates

One small issue you can have with Certbot and Let's Encrypt is that the certificates last only 3 months. You will regularly need to renew the certificates you use if you don't want people to get blocked by an ugly and scary message on their browser.

But since we have this Docker environment in place, it is easier than ever to renew the Let's Encrypt certificates!

```
$ docker compose run --rm certbot renew
```

This small "renew" command is enough to let your system work as expected. You just have to run it once every three months. You could even automate this process...

If you have any questions or advices, please create a comment below! I'll be really glad to read you. Also if you like this post, don't forget to share it with your friends. It helps a lot!

Written by Nathanaël Cherrier
Published in docker, ssl, nginx

Community discussion

Do not forget to read the Community Guidelines before commenting.

‹ Configurer HTTPS avec Nginx, Let's Encrypt et Docker

PHP8's Named arguments ›