Ewww — a Web server for static sites

Lars Wirzenius

2022-06-18 09:29

Contents

1 Introduction

Ewww is a web server for static sites. It aims to be simple code, simple to configure, simple to keep running, and fast.

1.1 Use cases

- I have files in a directory, and a domain name pointing at the host. I want to serve the files using HTTPS. I want the TLS certificate to come from Let's Encrypt, but the web server doesn't need to be involved in its creation or renewal.
- Same, but I have multiple domain names and each should serve from different directories and potentially have their own certificates.
- Same, but some of the domain names are aliases for each other, and web clients should be redirected to the main one.

2 Requirements

These are main, highlevel requirements. Detailed requirements are expressed as *scenarios* in the acceptance criteria chapter.

- Fast, at least 100 requests per second over localhost, using HTTPS, on my Thinkpad T480 laptop. A self-signed certificate is OK.
- Fast, time from starting server to having served first HTTPS request should be at most 100 ms.
- Serves only HTTPS, except what needs to be served over plain HTTP, e.g., for Let's Encrypt certificate validation. Any plain HTTP access must be explicitly allowed.

I don't need flexibility, and I don't want to configure anything that's not essential for this. Hardcoded assumptions are A-OK, if my life as someone running the program is easier.

At this point, I don't need support for `If-Modified-Since` or `ETag`. or generating directory listings. I don't even care about MIME types for now. Those will probably become important once I start using this software for real, but for now I am trying to keep requirements minimal.

3 Architecture

This is a thin layer on top of the Rust warp crate[1]. It does minimal processing for each request, and does not cache anything.

At startup, the server is provided with a single configuration file, which looks like this:

```
webroot: /srv/http/example.com
hosts:
  - example.com
  - www.example.com
ports:
  http: 80, 8080, 8888
  https: 443, 4433
tls-cert: /etc/letsencrypt/live/certname/fullchain.pem
tls-key: /etc/letsencrypt/live/certname/privkey.pem
```

The hosts are aliases; the first host on the list is the main one, the others automatically redirect to it.

The server is started via systemd or other mechanism that binds to privileged ports and handles process management: daemonization, restarting, etc. The configuration specifies for each port if plain HTTP or HTTPS is expected.

[1] `https://crates.io/crates/warp`

The server automatically listens on both port 80 (http) and 443 (https) so that it can serve the Let's Encrypt files. It only serves the `/.well-known/` path prefix in the webroot on port 80. Everything else gets redirected to 443. I don't think I need to serve other ports, but it's a handy feature to have for testing, so it shall be supported at least for testing.

There is no "reload configuration". The server needs to be restarted. This is good enough for me, but may not be good enough for more serious use on sites with much traffic. Restarting should be fast.

Only the GET and HEAD methods are supported for HTTP: this is a server for static content only. Every other method returns an error.

4 Acceptance criteria

4.1 Smoke test

given a self-signed certificate as **snakeoil.pem**, using key **snakeoil.key**
and directory **webroot**
and file **webroot/foo.html** from **webpage.html**
and a running server using config file **smoke.yaml**
when I request **GET https://example.com/foo.html**
then I get status code **200**
and header **content-type** is "**text/html**"
and body is "**this is your web page**"

The following config file does not specify port numbers. The test scaffolding adds randomly chosen port numbers so that the test can run without being root.

File: **smoke.yaml**

```
webroot: webroot
tls_cert: snakeoil.pem
tls_key: snakeoil.key
```

File: **webpage.html**

```
this is your web page
```

4.2 Performance test

```
given a self-signed certificate as snakeoil.pem, using key snakeoil.key
and a running server using config file smoke.yaml
and 1000 files in webroot
when I request files under https://example.com in random order 100000 times
then I can do at least 100 requests per second
```

4.3 Using POST, PUT, or DELETE fails

```
given a self-signed certificate as snakeoil.pem, using key snakeoil.key
and a running server using config file smoke.yaml

when I request POST https://example.com/
then I get status code 405
and allow is "GET HEAD"

when I request PUT https://example.com/
then I get status code 405
and allow is "GET HEAD"

when I request DELETE https://example.com/
then I get status code 405
and allow is "GET HEAD"
```

4.4 Request asking file from parent of webroot fails

The HTTP client must not be able to escape the webroot by using `/../` in the request path.

given a self-signed certificate as **snakeoil.pem**, using key **snakeoil.key**
and directory **somedir/webroot**
and file **somedir/secret.txt** from **secret.txt**
and file **somedir/webroot/foo.html** from **webpage.html**
and a running server using config file **somedir.yaml**
when I request **GET https://example.com/foo.html**
then I get status code **200**
and body is "**this is your web page**"
when I request **GET https://example.com/../secret.txt**
then I get status code **404**

File: **somedir.yaml**

```
webroot: somedir/webroot
tls_cert: snakeoil.pem
tls_key: snakeoil.key
```

File: **secret.txt**

```
secret
```