Caddy Lets Encrypt

Installation

Beschreibung:

Caddy ein letsencrypt Docker Proxy/Agent der Automatisch auch Zertifikate verlängert.
Caddy besteht aus einem Containerund einer Config Datei für die Domänen

Installation

Docker installieren

```
apt install curl docker.io docker-compose
```

Beispiel Konfig für einen apache2 der auf http port 8080 lauscht und den Caddy der auf Port 80 und 443 lauscht.
Port 80 wird auf 443 umgeleitet.
Der Port 443 wieder rum an 8080.

In der Composer Datei haben wir den port bei Apache rausgenommen, weil er auf 8080 im internet Docker Netz lauscht

Verzeichnisse erstellen

```
mkdir caddy_data
mkdir caddy_config
```

Die Compose Datei

```
version: '3.8'

services:
  apache:
    image: httpd:latest
    container_name: apache-webserver
    #ports:
    #  - "8080:8080"
```

```
    volumes:
      - ./apache_html:/usr/local/apache2/htdocs/
    restart: unless-stopped

  caddy:
    image: caddy:latest
    container_name: caddy-reverse-proxy
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
      - ./caddy_data:/data
      - ./caddy_config:/config
    restart: unless-stopped
```

Nun noch die Caddfile anlegen

Caddyfile für die Nutzung von Lets Encrypt

```
nano Caddyfile
```

Inhalt, wenn man es erst testen möchte mit Staging einfach den Kommentar entfernen.

```
example.com {
    reverse_proxy http://<servicename_docker_name>:8080
    tls <deine_emailadresse> {
        #ca https://acme-staging-v02.api.letsencrypt.org/directory
    }
}
```

Lets Encrypt Global Einstellungen wenn gewünscht:

Man kann aber auch die Lets Encrypt defination global setzen.
Dann braucht man das nicht mehr für jede Domain.

Wenn man es erst testen möchte mit Staging einfach den Kommentar entfernen.

```
{
    #acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
    email deine_emailadresse@example.com
}

example.com {
    reverse_proxy http://<servicename_docker_name>:8080
}
```

Über den Paramter tls können dann noch die Versionen angepasst werden.
Lässt man tls weg, wird automatisch das aktuellste verwendet, wie im oberen Beispiel, da haben wir kein tls mit drin.
Hier wird TLS Global definiert, sprich gillt für alle

```
{
    #acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
    email admin@example.com
    tls {
        protocols tls1.2 tls1.3
    }
}

example.com {
    reverse_proxy http://localhost:8080
}

example2.com {
    reverse_proxy http://localhost:8081
}
```

Man kann aber auch in den einzelnen Domain weiterhin die Versionen angeben, macht Sinn wenn diese sich unterscheiden.
Aus irgendeinem Grund, was nicht zu empfehlen ist, braucht der Server noch TLS 1.1

```
{
    #acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
    email admin@example.com
    tls {
        protocols tls1.2 tls1.3
    }
}
```

```
example.com {
	tls {
		protocols tls1.2 tls1.3
	}
	reverse_proxy http://localhost:8080
}

example2.com {
	tls {
		protocols tls1.1
	}
	reverse_proxy http://localhost:8081
}
```

Vorhandene Zertifikate nutzen:

Hier werden TLS 1.2 und TLS 1.3 erzwungen, und zusätzlich kannst du Client-Zertifikate verwenden, wenn du möchtest.

```
example.com {
    reverse_proxy http://<servicename_docker_name>:8080
    tls {
        protocols tls1.2 tls1.3
        client_auth {
            mode require_and_verify
            trusted_ca_cert_file /path/to/ca.pem
        }
    }
}
```

Caddyfile was selbst signiertes Zertifikat nutzen soll

Wir können einen Domain Namen angeben wenn wir einen eigenen DNS Server Betreiben, ansonsten Hostname oder ip Adressee angeben.

Hinweis: Ein selbst signiertes Zertifikat ist nur 12 Stunden gültig.
Es wird im Arbeitsspeicher generiert. Allerdings muss dann erneut das Zertifikat zur Ausnahme hinzugefügt werden.

Es muss unbedingt ein Domänen Namen verwendet werden!!!

Wenn kein eigener DNS Server vorhanden dann in die hosts Datei eintragen. IP Adresse geht nicht!!!!

```
example.local.lan {
    reverse_proxy http://<servicename_docker_name>:8080
    tls internal
}
```

Kurzversion in compose Datei ohne config Datei:

Man kann auch ohne config Dateien einen Caddy bauen, dann wird immer der übergebenen Domainname genommen und im parameter das redirect

Hier ist wordpress der interne name des Docker containers wordpress und der zielport auf dem Wordpress lauscht

in der .env datei kommt die PUBLIC url rein

Inhalt

```
PUBLIC_URL = "https://example.com"
```

Docker compose container

```
...
  caddy:
    image: caddy:latest
    restart: always
    ports:
      - "80:80"
      - "443:443"
    command: caddy reverse-proxy --from ${PUBLIC_URL} --to wordpress:80
    volumes:
      - ./data/caddy/data:/data
      - ./data/caddy/config:/config
```

...

Redirect Typen

Beschreibung:

Es gibt situationen da möchte man mehrere Domains auf eine Domain umleiten.
Transparent nicht transparent usw.

Varianten:

Szenario	Beispiel	Sichtbare Domain	Anmerkung
Redirect (Ziel-Domain sichtbar)	`redir`	Ziel-Domain	Klassische Umleitung (SEO-freundlich).
Reverse Proxy (Original-Domain bleibt)	`reverse_proxy`	Ursprungs-Domain	Domain-Masking.
Alias (Inhalt auf beiden Domains)	`reverse_proxy` für beide Domains	Beide Domains	Kein Unterschied, keine Umleitung.
Path-Spezifisch	`redir /path https://domain/path`	Ziel-Domain (bei Redirect)	Nur ein Pfad wird umgeleitet.
Rewrite (intern)	`rewrite`	Ursprungs-Domain	URL-Struktur bleibt für Benutzer gleich.
Header-Up bei Proxy	`reverse_proxy` mit Header	Ursprungs-Domain	Transparent für APIs oder Anwendungen.

Beispiel Caddyfiles

Modus: redir

```
example.de {
    reverse_proxy http://localhost:8080  # Dein Documento-Server
}

example.com,
example.org,
```

```
example.io {
    redir https://example.de{uri} permanent
}
```

Erklärung:

1. **Haupt-Domain konfigurieren**:
   - Im Block für `example.de` wird der Reverse-Proxy für deinen Documento-Server eingerichtet, sodass Anfragen an diese Domain an den Server weitergeleitet werden.
2. **Redirect für alternative Domains**:
   - Die weiteren Domains (`example.com`, `example.org`, `example.io`) werden in einem Block zusammengefasst.
   - Mit der `redir`-Anweisung leitest du alle Anfragen von diesen Domains auf `https://example.de` um.
   - `{uri}` sorgt dafür, dass der ursprüngliche Pfad erhalten bleibt, z. B. `/dokumente/signieren` wird auf `https://example.de/dokumente/signieren` weitergeleitet.
   - `permanent` gibt einen HTTP-Statuscode 301 aus, der dem Browser mitteilt, dass diese Umleitung dauerhaft ist.

Vorteile:

- **SEO-freundlich**: Suchmaschinen erkennen die 301-Weiterleitung als dauerhafte Umleitung, was Duplicate Content vermeidet.
- **Benutzerfreundlich**: Der Benutzer sieht nur die Ziel-Domain (`example.de`) im Browser.
- **SSL**: Caddy erstellt automatisch Zertifikate für alle Domains in der Caddyfile. Es empfielt. Sich abe reine Email zu hinterlegen, oder erstmal staging zu verwenden, wegen dem Rate Limit

Schritte zur Umsetzung:

1. Stelle sicher, dass die alternativen Domains (z. B. `example.com`) ebenfalls auf deinen Server zeigen (über DNS A/AAAA-Einträge).
2. Aktualisiere die Caddyfile entsprechend.
3. Starte oder lade Caddy neu mit `caddy reload`.

Modus: reverse proxy

```
example.com {
    reverse_proxy https://example.de
}
```

Erklärung:

- Alle Anfragen an `fexample.com` werden an `fexample.de` weitergeleitet, **ohne dass die Domain im Browser geändert wird**.
- Die Verbindung ist transparent, der Benutzer sieht weiterhin `example.com`.

Das ist meisten der Standard Fall wenn wir einfach nur einen Server ein SSL Zertifikat mittels Caddy verpassen wollen.
Dann redirecten wir auf die IP oder auf den docker container namen, mit dessen Port.
Aber man kann es natürlich auch auf andere öffentliche Domänen linken.

Modus: reverse proxy für mehrere Domains (Alias)

Weitere Varianten (Umleitungen oder Maskierungen)

1. **Einfaches Redirect mit URI beibehalten (wie im ersten Beispiel)**:
   - Benutzer wird umgeleitet und sieht die Ziel-Domain (e.g., `example.de`).

```
example.com {
    redir https://example.de{uri} permanent
}
```

2. **Alias (gleicher Inhalt unter beiden Domains)**:
   - Beide Domains zeigen auf denselben Inhalt, ohne dass eine Umleitung erfolgt.

```
examplede, example.com {
    reverse_proxy http://localhost:8080
}
```

Modus: Path-Spezifische Umleitung:

```
example.com {
    redir /docs https://example.de/docs permanent
}
```

Nur ein bestimmter Pfad wird auf eine andere Domain umgeleitet.
So kann man verschiedene Ordner zum Beispiel docs auf die subdomain docu.example.com umelieten die ein komplett anderer
Server ist.

Modus: URL-Rewrite (benutzerdefiniert):

```
example.com {
    rewrite /alte-seite /neue-seite
}
```

Ändert die URL-Struktur intern, ohne dass es für den Benutzer sichtbar ist.

Modus: Transparentes Proxying mit Header-Anpassung:

```
example.com {
    reverse_proxy https://example.de {
        header_up Host {host}
    }
}
```

Für APIs oder Anwendungen, bei denen die Ziel-Domain geändert wird, können spezielle Header gesetzt werden.