

Reverse Proxy with Traefik

- Installation
 - Getting started
 - Adding HTTPS support
 - Automatically redirect HTTP to HTTPS
 - Restricting access to services with ipwhitelist
 - Add custom error pages
 - Complete setup
- Adding a service
- Crowdsec setup
 - TODO

Installation

Getting started

First, let's create a small working setup, that exposes the dashboard to `traefik.<SITE>` where `site` is the hostname given as an environment variable (e.g. `traefik.example.com`)

`docker-compose.yml`

```
version: "3"

services:
  traefik:
    image: traefik
    container_name: traefik
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ./config/traefik.yml:/etc/traefik/traefik.yml
    labels:
      traefik.enable: true
      traefik.http.routers.traefik.service: api@internal
      traefik.http.routers.traefik.entrypoints: web
      traefik.http.routers.traefik.rule: Host(`traefik.${SITE}`)
```

`config/traefik.yml`

```
global:
  checkNewVersion: true
  sendAnonymousUsage: false
entryPoints:
  web:
    address: :80
api:
  dashboard: true
  insecure: true
providers:
```

docker:

endpoint: "unix:///var/run/docker.sock"

exposedByDefault: false

config/dynamic_config.yml is not yet needed

Traefik is now accepting http traffic on port 80. To route the traffic to an existing service, we can set labels inside the other apps `docker-compose` files (See [Adding a service](#)).

Adding HTTPS support

For now, traefik only accepts insecure http traffic. To allow secure access over https, we need to get a valid ssl certificate. For this we will use traefiks [automatic certificate generation](#). Traefik supports multiple acme challenges, but we will focus on using the DNS challenge.

Getting a letsencrypt certificate

First, we need to make some changes to our static configuration file `config/traefik.yml`.

1. Define an additional endpoint `websecure`:

```
entryPoints:
  websecure:
    address: :443
  http:
    tls:
      certResolver: netcup
```

2. Add a certificate resolver. We will use netcup for this example. The settings will need to set according to your provider (See [here](#)).

```
certificatesResolvers:
  netcup:
    acme:
      email: <email@example.com>
      storage: /etc/traefik/acme.json
      caServer: https://acme-staging-v02.api.letsencrypt.org/directory
    dnsChallenge:
      provider: netcup
      delayBeforeCheck: 900
    resolvers:
      - "root-dns.netcup.net:53"
      - "second-dns.netcup.net:53"
```

We will start to use the letsencrypt staging server to check if everything is working. If everythin works at the end, the `caServer` entry can simply be deleted.

3. Remove the `insecure: true` line from the api tag.

4. Add the dynamic config file provider

```
providers:
  file:
    filename: /etc/traefik/dynamic_config.yml
    watch: true
```

Next, we need to add the dynamic configuration by creating the file `config/dynamic_config.yml`.

```
tls:
  stores:
    default:
      defaultGeneratedCert:
      resolver: netcup
      domain:
        # main: "example.com"
      sans: "/*.example.com"
```

Here we our ssl certificate, that we will soon have, as our default one. This way we don't need to manually set it for every single service, that we want to access over https. We will simply acquire a wildcard certificate. If you also want to get a certificate for the base domain, you can uncomment the `main: "example.com"` line.

Now we only need to make some changes to our `docker-compose` file as follows:

1. Add `- "443:443"` to the ports of the container.
2. Bind the dynamic config and `acme.json` to the container

```
services:
  traefik:
    volumes:
      - ./config/dynamic_config.yml:/etc/traefik/dynamic_config.yml
      - ./config/acme.json:/etc/traefik/acme.json
```

3. Add the netcup customer information to the environment tag

```
services:
  traefik:
    environment:
      - NETCUP_CUSTOMER_NUMBER=${NETCUP_CUSTOMER_NUMBER}
      - NETCUP_API_KEY=${NETCUP_API_KEY}
      - NETCUP_API_PASSWORD=${NETCUP_API_PASSWORD}
```

4. Allow access to the dashboard over the websecure endpoint

```
services:
  traefik:
    labels:
      traefik.enable: true
      traefik.http.routers.traefik.service: api@internal
      traefik.http.routers.traefik.entrypoints: websecure
      traefik.http.routers.traefik.rule: Host(`traefik.${SITE}`)
```

When we start up the container now, traefik will try to get the ssl certificate from the staging server. After waiting some time, when you go to <https://traefik.example.com> you will hopefully see the letsencrypt certificate from the staging server. If yes, then everything works and you can switch to the non-staging caServer (see above). If not, make sure that everything is entered correctly.

After a restart of the container, traefik should have gotten a valid letsencrypt certificate and you should be able to access the dashboard correctly.

See also

- <https://doc.traefik.io/traefik/https/overview/>

Automatically redirect HTTP to HTTPS

Up until now, we would need to specify if we want to access the secure or insecure version of the page. In addition we would need to specify two routers for each service, one for the web entrypoint and one for the websecure entrypoint. We can avoid both if we tell traefik to automatically redirect every request that is getting to the web entrypoint.

For this simply add the following under `entryPoints: web` inside the `config/traefik.yml` file.

```
http:
  redirections:
    entrypoint:
      to: websecure
      scheme: https
```

This will let traefik know to automatically redirect every request at the web entrypoint to the websecure entrypoint.

See also

- <https://doc.traefik.io/traefik/routing/entrypoints/#redirection>

Restricting access to services with ipwhitelist

Often it is best to only allow clients from the local network to access some services and restrict access to everyone else. We can do that by using the ipwhitelist middleware from traefik.

Add the following to `config/dynamic_config.yml`:

```
http:
  middlewares:
    localonly:
      ipWhiteList:
        sourceRange:
          - "192.168.0.0/24"
          - "127.0.0.1/32"
```

Everything in the range `192.168.0.0/24` aswell as the machine itself will be whitelisted with these settings. If you want to restrict access to a service, add the following to the labels tag in the `docker-compose` file:

```
services:
  traefik:
    labels:
      traefik.http.routers.traefik.middlewares: "localonly@file"
```

Now traefik for example will only be accessible from within the home network.

See also

- <https://doc.traefik.io/traefik/middlewares/http/ipwhitelist/>

Add custom error pages

We can change the default error pages by adding the errorpages middleware to specific services or the whole entrypoint. Independent of if we want error pages for specific services or for a whole entrypoint we first need to setup a few things.

Setup

If we want to service custom error pages, we will need to host a webserver like nginx, since traefik itself can only route traffic and not serve additional pages.

We will add an nginx container to our traefik `docker-compose.yml`, since we want to use both in combination with each other. Simply add the following under `services`:

```
services:
  error_pages:
    image: nginx:latest
    container_name: error_pages
    volumes:
      - ./config/error_handling/error_pages:/usr/share/nginx/error_pages
      - ./config/error_handling/default.conf:/etc/nginx/conf.d/default.conf
      - ./logs:/var/log/nginx
    labels:
      traefik.enable: true
      traefik.http.routers.error-router.rule: HostRegexp(`{host:.+}`)
      traefik.http.routers.error-router.priority: 1
      traefik.http.routers.error-router.entrypoints: websecure
      traefik.http.routers.error-router.middlewares: error-pages-middleware
      traefik.http.middlewares.error-pages-middleware.errors.status: 400-599
      traefik.http.middlewares.error-pages-middleware.errors.service: error-pages-service
      traefik.http.middlewares.error-pages-middleware.errors.query: /{status}.html
      traefik.http.services.error-pages-service.loadbalancer.server.port: 80
```

We then need to create the folder `config/error_handling` as well as `config/error_handling/error_pages`. In addition we need to create the file `config/error_handling/default.conf`.

We will use the nginx image to create a webserver and attach `error_pages` and `default_conf` to it.

With the help of labels we create a new traefik router `error-router` with a low priority, that uses regex to catch all requests not otherwise routed. Next we create a new middleware `error-pages-middleware`, that takes all requests that couldn't be routed (status 400 to 599) and query the `error-pages-service` (nginx itself at port 80) at `/[status].html`. Imagine we get a request for `this-does-not-exist.example.com`. Instead of traefik directly responding with a 404 status, we instead catch the request and query nginx for `/404.html`

Now add the following to `config/error_handling/default.conf`:

```
server {  
    listen      80;  
    server_name localhost;  
  
    error_page 404 /404.html;  
    error_page 403 /403.html;  
    # error_page <status> /<status>.html  
  
    location /{  
        internal;  
        root /usr/share/nginx/error_pages;  
    }  
}
```

Whenever nginx gets a request for `/404.html` for example, it will serve the file `config/error_handling/error_pages/404.html`. Simply add your custom error pages to that folder and they will be served when needed.

Using error pages for specific services

To use custom pages for a specific service, add the following to the labels of your service inside your `docker-compose.yml`:

```
services:  
  traefik:  
    labels:  
      traefik.http.routers.traefik.middlewares: "error-pages-middleware@docker"
```

Using error pages for every service

To avoid having to manually assign the new error pages middleware to every service, we can instead use it for every service. Add the following to the websecure entrypoint inside your

config/traefik.yml :

```
entryPoints:
  websecure:
    http:
      middlewares:
        - error-pages-middleware@docker
```

Combine ipwhitelist with custom error pages

To combine both middlewares, we can use the `chain` middleware. Add the following to the http tag inside your `config/dynamic_config.yml` :

```
http:
  middlewares:
    localonly:
      ipWhiteList:
        sourceRange:
          - "192.168.0.0/24"
          - "127.0.0.1/32"
    secured:
      chain:
        middlewares:
          - error-pages-middleware@docker
          - localonly
```

This will combine the middlewares in **reverse order**. Every request goes thorough the localonly middleware first and if the ip is not whitelisted, nginx will be queried for the appropriate error page.

NOTE: This should theoretically return the 403.html file. Yet somehow, although nginx gets queried for `/403.html`, nginx returns the `/404.html` page. I can't figure out why....

Now, if a service should only be accessible from inside the home network, we can instead add the following label:

```
traefik.http.routers.traefik.middlewares: "secured@file"
```

When we want to allow public access to a service, we simply omit this line and traefik will automatically only use the custom error pages middleware.

See also

- <https://doc.traefik.io/traefik/middlewares/http/errorpages/>
- <https://www.imandrea.me/blog/traefik-custom-404/>

Complete setup

The following shows a complete setup for traefik with automatic ssl certificates, automatic http redirects to https, ipwhitelisting and custom error messages

File structure

```
├─ config
|   ├── acme.json
|   ├── dynamic_config.yml
|   ├── error_handling
|   |   ├── default.conf
|   |   └─ error_pages
|   |       ├── 403.html
|   |       ├── 404.html
|   |       └─ ...
|   └─ traefik.yml
├─ docker-compose.yml
└─ logs
    ├── access.log
    └─ error.log
```

docker-compose.yml

```
version: '3'

services:
  traefik:
    image: traefik
    container_name: traefik
    depends_on:
      - error_pages
    ports:
      - "80:80"
      - "443:443"
      - "8080:8080"
    volumes:
```

- /var/run/docker.sock:/var/run/docker.sock
- ./config/traefik.yml:/etc/traefik/traefik.yml
- ./config/dynamic_config.yml:/etc/traefik/dynamic_config.yml
- ./config/acme.json:/etc/traefik/acme.json

environment:

- NETCUP_CUSTOMER_NUMBER=\${NETCUP_CUSTOMER_NUMBER}
- NETCUP_API_KEY=\${NETCUP_API_KEY}
- NETCUP_API_PASSWORD=\${NETCUP_API_PASSWORD}

labels:

```
traefik.enable: "true"
traefik.http.routers.traefik.service: 'api@internal'
traefik.http.routers.traefik.middlewares: "secured@file"
traefik.http.routers.traefik.entrypoints: "websecure"
traefik.http.routers.traefik.rule: "Host(`traefik.${SITE}`)"
```

error_pages:

image: nginx:latest

container_name: error_pages

volumes:

- ./config/error_handling/error_pages:/usr/share/nginx/error_pages
- ./config/error_handling/default.conf:/etc/nginx/conf.d/default.conf
- ./logs:/var/log/nginx

labels:

```
traefik.enable: true
```

```
traefik.http.routers.error-router.rule: HostRegexp(`{host:.+}`)
traefik.http.routers.error-router.priority: 1
traefik.http.routers.error-router.entrypoints: websecure
traefik.http.routers.error-router.middlewares: error-pages-middleware
traefik.http.middlewares.error-pages-middleware.errors.status: 400-599
traefik.http.middlewares.error-pages-middleware.errors.service: error-pages-service
traefik.http.middlewares.error-pages-middleware.errors.query: /{status}.html
traefik.http.services.error-pages-service.loadbalancer.server.port: 80
```

config/traefic.yml

global:

checkNewVersion: true

sendAnonymousUsage: false

entryPoints:

web:

address: :80

http:

redirections:

entrypoint:

to: websecure

scheme: https

websecure:

address: :443

http:

middlewares:

- error-pages-middleware@docker

tls:

certResolver: netcup

certificatesResolvers:

netcup:

acme:

email: <email@example.com>

storage: /etc/traefik/acme.json

caServer: https://acme-staging-v02.api.letsencrypt.org/directory

dnsChallenge:

provider: netcup

delayBeforeCheck: 900

resolvers:

- "root-dns.netcup.net:53"

- "second-dns.netcup.net:53"

api:

dashboard: true

providers:

docker:

endpoint: "unix:///var/run/docker.sock"

exposedByDefault: false

file:

filename: /etc/traefik/dynamic_config.yml

watch: true

config/dynamic_config.yml

```
http:
  middlewares:
    localonly:
      ipWhiteList:
        sourceRange:
          - "192.168.0.0/24"
          - "127.0.0.1/32"
    secured:
      chain:
        middlewares:
          - error-pages-middleware@docker
          - localonly
  tls:
    stores:
      default:
        defaultGeneratedCert:
          resolver: netcup
        domain:
          # main: "example.com"
          sans: "/*.example.com"
```

config/error_handling/default.conf

```
server {
    listen    80;
    server_name localhost;

    error_page 403 /403.html;
    error_page 404 /404.html;
    # error_page <code> /<code>.html;
    location /{
        internal;
        root /usr/share/nginx/error_pages;
    }
}
```

config/error_handling/error_pages/xxx.html

You can create your own custom error pages or insert predefined ones.

Some examples: <https://github.com/tarampampam/error-pages>

Things to keep in mind

- Set your email and domain name(s) accordingly.
- Change the provider settings for the DNS challenge according to your provider [here](#).
- Make sure you use the staging caServer of letsencrypt first and then switch back to the normal one, when everything works as intended.

Adding a service

If you followed the [Installation](#) correctly, adding services to traefik gets really easy.

Adding docker services with docker compose

To add a docker compose service, simply add the following labels:

```
services:
  bookstack:
    labels:
      traefik.enable: true
      traefik.http.routers.bookstack.rule: Host(`bookstack.${SITE}`)
      traefik.http.routers.bookstack.entrypoints: websecure
```

This will dynamically tell traefik to route the service, no restart of traefik is required. The service will be routed via the `websecure` entrypoint (Port 443 with default wildcard certificate). The service will then be available at `https://bookstack.<SITE>`, e.g. `https://bookstack.example.com`. By default, the [custom error pages](#) will be used.

If you want your service to only be accessible from your internal network, add the following label aswell:

```
traefik.http.routers.bookstack.middlewares: secured@file
```

If your container uses multiple ports, you can specify the correct one for traefik to use by adding the label:

```
traefik.http.services.<servicename>.loadbalancer.server.port: <PORT>
```

Crowdsec setup

Crowdsec setup

TODO

- <https://goneuland.de/traefik-v2-3-reverse-proxy-mit-crowdsec-im-stack-einrichten/>