Published in Soulweb Academy

Marco Martino Follow
Mar 24, 2021 · 5 min read

Docker local dev stack with Traefik + Https + Dnsmasq + Locally-Trusted certificate (Mkcert) for Ubuntu 20.04

Manage your local development containers like a pro!

Photo by: Antoine Petitteville

Intro

In this *tutorial* I want to share the `Traefik` configuration we use in our company for local `Docker` development stacks.

> *"We have built this configuration to have a proper local development environment to rely on, with `https` enabled on local custom domains through* locally-trusted *certificates."*

It took us some time to actually have everything working together and in order, I hope this can make it easier for you! ;)

At the end of this tutorial you'll find a link to the repository containing the whole configuration.

What are we using

- Traefik v2.0
- Portainer
- *Whoami*
- Dnsmasq
- Mkcert

Requirements

- Docker
- Docker Compose

What's our mission

At the end of this tutorial, we'll hopefully have a running stack where to mount our `Docker` applications. The stack will run under the `*.docker.localdev` domain.

`Traefik` is going to be the `proxy` *server* for our applications, as well as our `middleware` to run those applications under `https`.

`Portainer` will help us to manage our `Docker` stacks.

`Whoami` prints out `Docker` *os* informations, in case we need them.

`Dnsmasq` is going to be in charge of routing all the request from `*.docker.localdev` to `Traefik` on `localhost`.

`Mkcert` will generate the *locally-trusted certificates* for our local *domain*.

Setup

In order to start our configuration, let's create a new folder under which we can put the required files:

```
$ mkdir docker
```

Traefik 2.0

We are going to start with `Traefik`!

Under the new created folder, let's create a new file called `traefik.yml` with the content:

```
version: '3'

services:
  traefik:
    image: traefik:v2.0
    container_name: "${DOCKER_NAME}_traefik"
    command:
      - --providers.docker=true
      # Enable the API handler in insecure mode,
      # which means that the Traefik API will be available directly
      # on the entry point named traefik.
      - --api.insecure=true
      # Defines the path to the configuration file with the certificates list.
      - --providers.file.filename=/root/.config/ssl.toml
      # Define Traefik entry points to port [80] for http and port [443] for https.
      - --entrypoints.web.address=:80
      - --entrypoints.websecure.address=:443
    networks:
      # Define the network on which traefik is going to operate.
      - web
    ports:
      # Open traefik http [80] and https [443] ports.
      - '80:80'
      - '443:443'
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      # Mount the configuration file with the certificates list.
      - ./traefik-ssl.toml:/root/.config/ssl.toml
      # Mount the folder containing the certificates for https.
      - ./certs/:/certs/
    labels:
      - "traefik.enable=true"
      # Enable Traefik API handler entrypoint on http.
      - "traefik.http.routers.traefik-http.entrypoints=web"
      # Define Traefik API handler http host.
      - "traefik.http.routers.traefik-http.rule=Host(`${DOCKER_BASE_URL}`)"
      # Define http middleware and redirection to https.
      - "traefik.http.routers.traefik-http.middlewares=traefik-https"
      - "traefik.http.middlewares.traefik-https.redirectscheme.scheme=https"
      # Enable Traefik API handler entrypoint on https.
      - "traefik.http.routers.traefik.entrypoints=websecure"
      # By default the Traefik API handler operates on the port [8080].
      # Define a load balancer to route the entry point to [8080].
      - "traefik.http.services.traefik.loadbalancer.server.port=8080"
      # Define Traefik API handler host.
      - "traefik.http.routers.traefik.rule=Host(`${DOCKER_BASE_URL}`)"
      # Instructs Traefik that the current router is dedicated to HTTPS requests only.
      - "traefik.http.routers.traefik.tls=true"

networks:
  web:
    external: true
```

You can see that we are using a couple of *environment variables*, `${DOCKER_NAME}` and `${DOCKER_BASE_URL}`. To define those *variables*, let's create a new file called `.env` with the content:

```
DOCKER_NAME=docker4localdev
DOCKER_BASE_URL=docker.localdev
```

We also have to create the network *[web]* which `Traefik` is going to operate on:

```
$ docker network create web
```

To conclude this section, let’s create the needed file and folder, mounted in our `Traefik` *volumes* above, to later setup our *locally-signed* certificates:

```
$ touch traefik-ssl.toml
$ mkdir certs
```

Portainer

Now, let's define the `portainer` configuration on the `traefik.yml` file, just after the `traefik` one:

```
portainer:
    image: portainer/portainer
    container_name: "${DOCKER_NAME}_portainer"
    command: --no-auth -H unix:///var/run/docker.sock
    networks:
      - web
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    labels:
      # Enable Portainer handler entrypoint on http.
      - "traefik.http.routers.${DOCKER_NAME}_portainer-http.entrypoints=web"
      # Define Portainer handler http host.
      - "traefik.http.routers.${DOCKER_NAME}_portainer-http.rule=Host(`portainer.${DOCKER_BASE_URL}`)"
      # Define http middleware and redirection to https.
      - "traefik.http.routers.${DOCKER_NAME}_portainer-http.middlewares=${DOCKER_NAME}_portainer-https"
      - "traefik.http.middlewares.${DOCKER_NAME}_portainer-https.redirectscheme.scheme=https"
      # Enable Portainer handler entrypoint on https.
      - "traefik.http.routers.${DOCKER_NAME}_portainer.entrypoints=websecure"
      # Define Portainer handler host.
      - "traefik.http.routers.${DOCKER_NAME}_portainer.rule=Host(`portainer.${DOCKER_BASE_URL}`)"
      # Instructs Traefik that the current router is dedicated to HTTPS requests only.
      - "traefik.http.routers.${DOCKER_NAME}_portainer.tls=true"
      # Define on which network Traefik is operating.
      - "traefik.docker.network=web"
```

Whoami

Last step, on our `traefik.yml` configuration, is to add the `whoami` configuration:

```
whoami:
    image: containous/whoami
    container_name: "${DOCKER_NAME}_whoami"
    networks:
      - web
    labels:
      # Enable Whoami handler entrypoint on http.
      - "traefik.http.routers.${DOCKER_NAME}_whoami-http.entrypoints=web"
      # Define Whoami handler http host.
      - "traefik.http.routers.${DOCKER_NAME}_whoami-http.rule=Host(`whoami.${DOCKER_BASE_URL}`)"
      # Define http middleware and redirection to https.
      - "traefik.http.routers.${DOCKER_NAME}_whoami-http.middlewares=${DOCKER_NAME}_whoami-https"
      - "traefik.http.middlewares.${DOCKER_NAME}_whoami-https.redirectscheme.scheme=https"
      # Enable Whoami handler entrypoint on https.
      - "traefik.http.routers.${DOCKER_NAME}_whoami.entrypoints=websecure"
      # Define Whoami handler host.
      - "traefik.http.routers.${DOCKER_NAME}_whoami.rule=Host(`whoami.${DOCKER_BASE_URL}`)"
      # Instructs Whoami that the current router is dedicated to HTTPS requests only.
      - "traefik.http.routers.${DOCKER_NAME}_whoami.tls=true"
      # Define on which network Traefik is operating.
      - "traefik.docker.network=web"
```

Dnsmasq

We can now install and setup `Dnsmasq`. You are probably used to setup your local `domains` in the `hosts` file, one by one. `Dnsmasq` is going to make this process easier.

Install

`Ubuntu` comes with `systemd-resolve`, which you need to disable since it binds to port *[53]*, which will conflict with `Dnsmasq` port.

Run the following commands to disable the service:

```
$ sudo systemctl disable systemd-resolved
$ sudo systemctl stop systemd-resolved
```

Remove the symlinked `resolv.conf` file:

```
$ ls -lh /etc/resolv.conf
$ sudo rm /etc/resolv.conf
```

Create new `resolv.conf` file:

```
$ sudo bash -c 'echo "nameserver 127.0.0.1" > /etc/resolv.conf'
$ sudo bash -c 'echo "nameserver 1.1.1.1" >> /etc/resolv.conf'
```

Install `Dnsmasq`:

```
$ sudo apt install dnsmasq
```

Setup

Let’s add `.localdev` to the `dnsmasq` config file:

```
$ sudo bash -c 'echo "address=/.localdev/127.0.0.1" >> /etc/dnsmasq.conf'
```

Create the `resolver` for the added address:

```
$ sudo mkdir -v /etc/resolver && sudo bash -c 'echo "nameserver 127.0.0.1" > /etc/resolver/localdev'
```

Restart `Dnsmasq` and network manager service:

```
$ sudo systemctl restart dnsmasq
```

Locally-trusted certificates (with Mkcert)

`Mkcert` is a simple tool for making locally-trusted development certificates.

Install

As a prerequisite, you are required to install `certutil`, a command-line utility that can create and modify certificate and key databases before you can install `mkcert` utility:

```
$ sudo apt install libnss3-tools -y
```

Then you can install `mkcert`:

```
$ wget https://github.com/FiloSottile/mkcert/releases/download/v1.4.3/mkcert-v1.4.3-linux-amd64
$ sudo mv mkcert-v1.4.3-linux-amd64 /usr/local/bin/mkcert && chmod +x /usr/local/bin/mkcert
```

Generate certificate

First, generate a local `Certificate-Authority`:

```
$ mkcert -install
```

Then, you are ready to generate your *locally-trusted* certificate:

```
$ mkcert -key-file ./certs/key.pem -cert-file ./certs/cert.pem localdev 'docker.localdev' '*.docker.localdev'
```

We can now add the generated certificate to the `traefik-ssl.toml` file:

```
[tls]

[tls.stores]

[tls.stores.default]

[tls.stores.default.defaultCertificate]
certFile = "/certs/cert.pem"
keyFile = "/certs/key.pem"
```

Time to Build!

Finally, we are ready to build and test our stack:

```
docker-compose -f traefik.yml up -d
```

Available domains

If everything is well done we should be able to access our stack:

- `Traefik` *Dashboard*: https://docker.localdev
- `Portainer`: https://portainer.docker.localdev
- `Whoami`: https://whoami.docker.localdev

Repository

You can find the whole configuration on this repository.

The repository contains also a small example of a `Python` *app* with `Flask`, under the folder `myapp`, where you can find a `docker-compose` configuration to see how to attach your applications to the `Traefik` stack we just built!

. . .

Among the many combinations available to manage local webapps with custom domains, we find this as one of the most flexible and straightforward. Once you set it up, you can just add other apps to the stack and start to work on it.

Now you can look at those times when you used to add that line manually to your `/etc/hosts` file or skip the complaints of your browser about your fake certificate, like if you're looking at an old picture of childhood 😝.