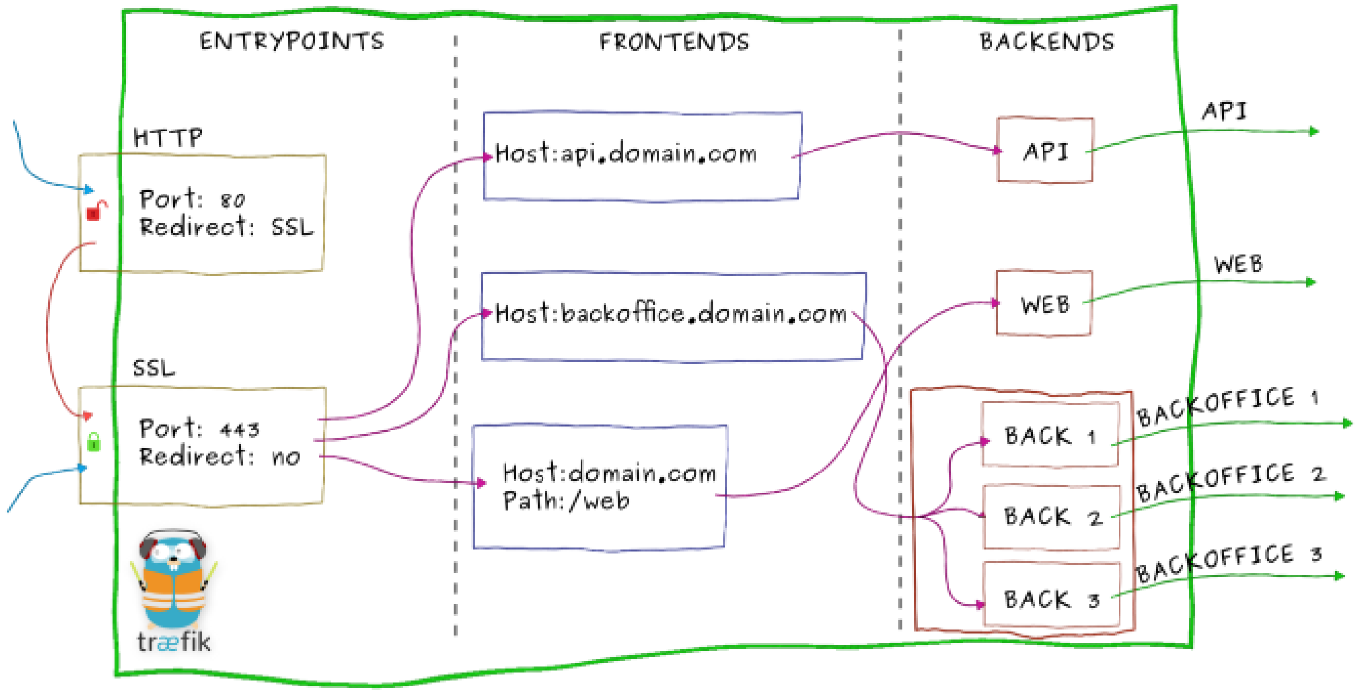
Getting started with Traefik and Consul

Andrei Dascalu Follow
Oct 30, 2018 · 4 min read



load balancing like a pro

A while ago I did a brief comparison piece on Traefik vs Nginx (vs Haproxy) as load balancers. Since then I've become closer and closer to containers (using them everywhere now) as well as orchestration systems while Traefik (due to its quick integration and painless SSL) has become the default tool for exposing services to the outside world.

This post is mean to offer Traefik newcomes a quick start that goes slightly beyond the official documentation. It's by no means a comprehensive 'best practices' list, but it can serve you well in getting up and running a web development environment in no time.

Let's start with some prerequisites.You will need a Unix machine (I use a lot VMs in Hetzner because they are cheap — 2 cores and 4Gb of RAM for 5 euros a month, whaaat?). *Docker* is next, *docker-compose* is third. You will also need the htpasswd tool (*apache2-utils* package in Ubuntu or via *brew* for Mac users). Once you have that, I suggest also doing a bit of kernel networking optimisations.

Now, the initial setup will have 3 elements: **Traefik** load balancer with automatic SSL from Letsencrypt, **Consul** for storing configuration and an

Nginx webserver just to show that our stuff works.

Start by creating a new folder for your project (or better yet, a Git repository somewhere like github or bitbucket).

In the newly created folder, create a subfolder creatively called *traefik* (where we will store traefik-related configs). Inside *traefik*, run:

```
htpasswd -Bc security.htpasswd testuser
```

You will be prompted for a password and password confirmation. Once that's done, you will end up with a file called secure.htpasswd which contains a user and an encrypted password that will secure our private stuff (mainly Traefik dashboard and Consul UI).

Also inside traefik subfolder, create a file for Traefik's configuration called *traefik.toml* (yes, toml, something that yml/json users will learn to despise) with the following content:

Pause for explanation:

- *defaultEntryPoints* define those config elements that define places that our application can be called through publicly. We have 2, creatively called http and https which accept anything on ports 80 and 443 respectively.
- *docker* tells Traefik to use its Docker integration and the 2 items mean that Traefik will keep listening for new containers that start while **not** trying to expose every container that exposes a port.
- *acme* is the Letsencrypt integration. It defines the email to send with SSL certification requests, where to store certifications (*storage* can be a file, but in this case it is a Consul path), we are telling it to read the hosts dynamically and to log requests, as well as defining the points on which to listed for Letsencrypt's callback.
- *acme.domains* lists the domain for which to ask for separate certificates. You can list several such blocks, but each will have a main domain that will be certified as well as several 'sans'.
- *entryPoints* we define our entrypoints, you can define hosts per ports, etc

- *consul* enables Consul K/V store. The endpoint is the Consul api, (*kvstore* is the hostname we will use)

Once the file is saved, let’s exit the *traefik* subfolder and go back to your main project folder. Here we will need a *docker-compose.yml* file.

What’s important:

Traefik container:

- we mount 3 items for traefik: the config file, the Docker socket on your system and the traefik folder as a whole (we need this just for the htpasswd file in this form, but let’s be lazy)
- The labels tell our load balancer what container port to direct requests to, what domains to listen and route to this container, to disable health check on itself (for the monitor application this helps with performance), to enable load balancing for this container (since traefik will ignore this by default)

- We also override the traefik start command, here we ask for docker and explicitly ask to not expose containers by default (these are redundant here and set just for example, as we've specified the same in the config file)
- We also setup basic http authentication by pointing to the file we created above and which was mounted as a volume

Consul container:

- similarly, we specify roughly similar labels — enable Traefik, specify hosts and authentication
- we also mount a volume for data storage, if we upgrade or stop the container we may want to preserve data

Nginx:

- no basic authentication here, it's just an example of something expose to the outside world

We also need to manually create a volume and a network. It's good practice to manually create the network as everything that Traefik load balances needs to be on the same network. This also enables you to reuse the network with difference compose files but still allow the same Traefik to load balance additional services defined in a different file.

```
docker network create traefik
docker volume create kv-data
```

We're almost there!

Unfortunately Traefik has no way to populate the K/V store automatically, so now we need to start up the whole shabang.

```
docker-compose up -d
```

Then we will need to manually tell Traefik to store configuration in Consul

```
docker-compose exec loadbalancer /bin/sh

loadbalancer> traefik storeconfig
```

Now you're all set!

Of course, for the whole thing to work don't forget to use your own domains, for which you've setup proper DNS pointing to your server and your 80/443 ports are open! Easy stuff to miss.

Enjoy!

Docker Traefik Load Balancing Containers Consul

59 claps

WRITTEN BY

Andrei Dascalu

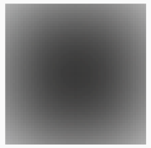
Follow

Write the first response

More From Medium

Gilded Rose Refactoring Kata

Tauqeer Ahmad



Make your business logic readable, and your configuration logic extendable

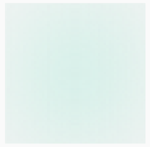
José 'Joshi' Ráez Rodríguez in The Startup

Data Structure & Algorithms—1. A C++ Primer

Brady Huang in CS Transfer Student

Using form_tag and params to create a search or render a sorted view in Rails

Tracie Masek



The Habits of Productive Developers

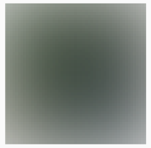
Dhananjay Trivedi in Better Programming

What Coding Bootcamp Students Do In A Typical Day

Sarah Daniels in The Startup

Two Way Analytics with R Shiny and Pokemon

Marshall Krassenstein in The Startup



Write simple and no-surprise code

Liu Tao in The Startup