Bild: Albert Hulm

HTTP-Einweiser

Eingehenden HTTP-Verkehr mit Traefik routen

Soll ein Webserver mehr als eine Webseite ausliefern und auf unterschiedliche Hostnamen hören, braucht es einen HTTP-Router. Die Open-Source-Lösung Traefik ist sehr flexibel. Vor allem im Zusammenspiel mit Containern. Noch dazu ist sie schnell eingerichtet.

Von Jan Mahn

Mit Container-Technik wie Docker ist es kein großer Aufwand, mehrere Webdienste auf einer Maschine zum Laufen zu bringen. Selbst ein kleiner virtueller Server oder ein heimischer Raspberry Pi können problemlos mehrere Webseiten ausliefern – zum Beispiel eine Dateiablage für die Familie und einen Blog. Damit hinter einer öffentlichen IP-Adresse verschiedene Seiten antworten, braucht es einen Router für eingehenden HTTP-Verkehr. Im besten Fall übernimmt dieser auch gleich die Beschaffung von TLS-Zertifikaten für die Auslieferung von https-Seiten, schreibt Logfiles und kümmert sich um Autorisierung der Zugriffe oder filtert zum Beispiel nach IP-Adressen.

Besonders populär sind im Container-Umfeld HTTP-Router auf Basis eines Nginx-Webservers – allen voran der Container „jwilder/nginx-proxy“. Er löst jedoch nur genau eine Aufgabe, nämlich das Routen von eingehendem Verkehr. Für die Zertifikatsbeschaffung braucht es einen weiteren Container, für Logging, Autorisierung und Filterung auch. Schnell hat man einen Container-Zoo aus kleinen Diensten von unterschiedlichen Entwicklern, bevor auch nur ein einziger Container mit der eigentlichen Anwendung läuft.

Eine vielseitige und dennoch handliche Lösung ist das Open-Source-Programm Traefik. In einem Container gibt es hier alles aus einer Hand. Traefik ist in der Programmiersprache Go speziell für diese Aufgabe geschrieben, kommt ohne zusätzlichen Webserver aus und fühlt sich in containerisierten Umgebungen zu Hause. Entwickelt wird die Software vom Unternehmen Containous, das sein Geld mit Beratung und Training im Container-Umfeld sowie mit einer Enterprise-Version von Traefik verdient. Die Community hinter dem Projekt ist groß und Kontributoren aus vielen Unternehmen tragen zur Entwicklung bei – Traefik

ist kein Spielzeug, sondern Software, die auch in großen Infrastrukturen genutzt wird.

Aktuell steht ein Umbruch bei Traefik bevor. Version 1.7 wird von einer neuen Major-Version 2.0 abgelöst, die aktuell in der Beta-Phase steckt. Für Ende 2019 ist die Veröffentlichung von 2.0 geplant. In diesem Artikel kommt bereits die neue Version zum Einsatz – für den produktiven Betrieb ist sie zwar noch nicht freigegeben, viele neue Funktionen sprechen aber dafür, sich jetzt schon damit vertraut zu machen. Der wesentliche Vorteil: Traefik 2.0 kann nicht nur HTTP, sondern beliebigen TCP-Verkehr verarbeiten und TLS terminieren. Das macht ihn zum Beispiel interessant, um einen Mailserver zu betreiben. Aber auch beim HTTP-Routing, das in diesem Artikel vorgestellt wird, gibt es wesentliche Neuerungen. Um diese zu verstehen, muss man zunächst die Funktion von Traefik und ein paar Begriffe kennen.

Begriffswelt

Traefik ist ein Edge-Router, lauscht also auf der Netzwerkkarte Richtung Internet, meist auf den Ports 80 und 443. In der Traefik-Terminologie ist diese Verbindung in die Außenwelt ein **EntryPoint**. Die Entscheidung, was mit einer Anfrage passieren soll, treffen dann sogenannte **Router**. Auf dieser Stufe definiert der Administrator Regeln, die darüber entscheiden, an welchen **Service** die Anfrage gehen soll. Jeder Service bringt die Funktionen eines Load-Balancers mit und verteilt die Anfragen auf alle Server, welche die geforderte Ressource bereitstellen. Das können Container sein, aber auch gewöhnlich installierte Webserver auf einer anderen Maschine, die Traefik über das Netzwerk erreichen kann.

Traefik bekommt also eine HTTP-Anfrage, entscheidet, wohin diese gehen soll, stellt dort die Anfrage und gibt das Ergebnis an den Client zurück. Zwischendurch – und das ist neu in Version 2.0 – kann man sogenannte **Middlewares** in die Kette einbauen. Diese bekommen alle Details zur Anfrage und können damit Aufgaben wie Logging übernehmen, aber zum Beispiel auch Pfade verändern, Zugriffe pro Minute limitieren oder Anfragen als unberechtigt ablehnen.

In einem Traefik-Setup kann die Definition von Routern, Services und Middlewares dynamisch sein. Man muss nicht per Hand an einer Konfigurationsdatei herumeditieren. Immer wenn man einen neuen Container startet oder einen vom Netz nimmt, bemerkt Traefik die Veränderung und liest die Konfiguration aus Informationen aus, die man dem Container mitgegeben hat. Zuständig für die automatische Konfiguration sind **Provider** – neben Docker kann Traefik auch auf Kubernetes, Marathon und Rancher reagieren. In den Beispielen in diesem Artikel kommt Docker als Provider zum Einsatz. Wer Traefik in anderen Umgebungen einsetzen möchte, findet in der Dokumentation (zu finden über ct.de/y1ds) passende Rezepte.

Erstkontakt

Für den ersten Traefik-Kontakt reicht eine Linux- oder macOS-Maschine mit aktuellem Docker und Docker-Compose aus. Docker für Windows verursacht leider teilweise Probleme beim Versuch, den Unix-Socket an den Container zu übergeben. Wenn Sie die Zertifikatsbeschaffung testen möchten, brauchen Sie einen Server mit öffentlicher IP und eine Domain. Über ct.de/y1ds finden Sie ein GitHub-Repository mit den vorgestellten Beispielen zum Kopieren.

Das Listing auf dieser Seite zeigt ein simples Beispiel. Der Traefik-Container bekommt mit dem Befehl `--providers.docker` die Aufgabe, auf dem Docker-Socket zu lauschen. Dieser wird ihm als Docker-Volume vom Host mitgegeben [1]. Nur der Traefik-Container bekommt eine Portweiterleitung auf die Netzwerkkarte des darunterliegenden Computers, er kann aber über das interne Docker-Netz mit den Webservern sprechen. Es folgen zwei weitere Container: „web1“ ist ein Nginx-Container, der mit einer Begrüßungsseite antwortet, „web2“ zeigt Details zur Anfrage und zum Container an.

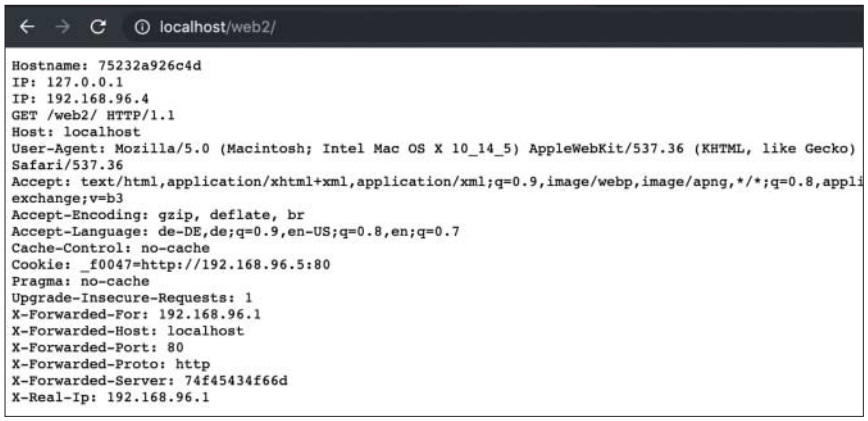
Die Routing-Regeln werden in Labels definiert, die dem Container angeheftet werden. Die Bezeichnungen der Einstellungen sind hierarchisch aufgebaut und mit Punkten getrennt. Definiert wird also eine `rule` für den `router` mit dem Namen `web1`. Hinter dem `=` folgt der Inhalt der Regel: `Host()` legt fest, dass der mit dem HTTP-Aufruf übergebene HTTP-Host-Header „web1.localhost“ lauten muss, damit Traefik den Verkehr hierhinleitet.

Der zweite Container bekommt eine komplexere Regel. Mit Version 2.0 ist es möglich, Regeln logisch mit „und“ (`&&`) und „oder“ (`||`) zu verknüpfen. Die Beispielregel bildet ein in der Praxis häufiges Beispiel ab: Sowohl unter der Subdomain „web2.localhost“, als auch unter „localhost/web2“ soll der Container antworten. Dafür wird als weiterer Baustein `PathPrefix()` verwendet, der den Beginn des Pfads prüft. Die Dokumentation (siehe ct.de/y1ds) listet alle möglichen Regeln auf – auch Regex-Freunde werden dort fündig.

Fahren Sie die Zusammenstellung mit `docker-compose up` hoch und versuchen Sie, aus dem Browser auf die verschiedenen Adressen zuzugreifen. Geben Sie die Loopback-IP 127.0.0.1 direkt ein, antwor-

```
version: "3.7"
services:
  traefik:
    image: traefik:v2.0
    command: --providers.docker
    ports:
     - "80:80"
    volumes:
     - /var/run/docker.sock:/var/run/docker.sock:rw
    web1:
      image: nginx:alpine
      labels:
      - traefik.http.routers.web1.rule=Host(`web1.localhost`)
    web2:
      image: containous/whoami
      labels:
       - traefik.http.routers.web2.rule=Host(`web2.localhost`) || ↵
       ↳( Host(`localhost`) && PathPrefix(`/web2`) )
```

Über Labels an den Containern bekommt Traefik Einstellungen übermittelt.



Der Container aus dem Image `containous/whoami` zeigt Details zur Anfrage und macht das Einrichten von Traefik einfacher.

tet Traefik mit einem Fehler 404, da keine passende Regel existiert.

Um Traefiks Fähigkeiten beim Load-Balancing zu testen, muss in Ihrer Docker-Umgebung der Swarm-Mode aktiv sein. Das funktioniert zum Testen auch mit nur einer Maschine. Wie Sie den Swarm-Mode aktivieren, haben wir bereits beschrieben [2]. Um mehrere Instanzen des Dienstes „web2“ im Schwarm zu erzeugen, starten Sie die Docker-Compose-Komposition erneut:

```
docker-compose up -d --scale web2=2
```

Wenn Sie jetzt „web2.localhost“ mehrmals neu laden, können Sie an der Angabe „Hostname“ in der ersten Zeile beobachten, wie die beiden Container immer abwechselnd antworten. Wer seine Besucher trotz Load-Balancing lieber auf einen Server festnageln möchte, kann per Label „Stickiness“ aktivieren. Der Client bekommt einen Cookie und Traefik leitet seine Anfragen immer gleich weiter. Bei der Gelegenheit wird auch das Prinzip der Labels verständlicher. Hängen Sie in der Docker-Compose-Datei einfach folgendes Label an den Service „web2“ an:

```
- traefik.http.services.web2.↵
↳loadBalancer.stickiness
```

Starten Sie dann erneut mit dem Parameter `--scale web2=2` und rufen Sie die Seite wiederholt auf. Das Pingpong hört auf und in den Entwicklertools des Browsers sollten Sie einen Cookie finden.

Zwischenschicht

Auch Middlewares können per Label aktiviert werden. Eine sehr simple Middleware heißt „ipwhitelist“. Sie kann einzelnen IPs oder IP-Bereichen erlauben, auf eine Ressource zuzugreifen. Alle anderen werden mit dem HTTP-Status 403 („Forbidden“) abgewiesen. Eine Middleware muss angelegt und mit einem Router verknüpft werden. Fügen Sie folgende Labels an den Dienst „web2“ an:

```
- traefik.http.middlewares.demo-↵
↳whitelist.sourcerange=127.0.0.1/32
- traefik.http.routers.web2.↵
↳middlewares=demo-whitelist@docker
```

Das erste Label legt die Middleware mit dem Namen „demo-whitelist“ an. Traefik registriert die Middleware im Namensraum des Providers, in diesem Fall heißt der „docker“ – sie kann also an jeden Router angehängt werden. Das passiert im zweiten Label. Der Router „web2“ leitet jetzt alle Anfragen durch die Middleware. Sie prüft, ob die anfragende IP aus dem 127er-Netz kommt. Hier können Sie, jeweils mit Komma getrennt, weitere Berechtigte anhängen. Ein Blick in die Doku im Abschnitt „Middlewares“ offenbart das volle Potenzial.

Statisch eingestellt

Die dynamische Konfiguration über Labels ist nur eine mögliche Option, die immer dann zum Einsatz kommen kann, wenn eine Einstellung direkt mit einem Container zusammenhängt. Für globale Einstellungen gibt es eine statische Konfigurationsdatei. Beim Start liest Traefik alle Einstellungen aus einer Datei mit dem Namen „traefik.toml“ oder „traefik.yml“ im Ordner „/etc/traefik“. Letzteres Format ist Autoren von Docker-Compose-Dateien eher vertraut. Damit die statische Konfiguration in Traefik ankommt, legen Sie die Datei „trafik.yml“ im gleichen Verzeichnis wie die Docker-Compose-Datei an und fügen Sie dem Traefik-Container ein zusätzliches Volume hinzu:

```
- ./traefik.yml:/etc/↵
↳traefik/traefik.yml
```

In dieser Datei kann man jetzt zum Beispiel die Übermittlung von Telemetrie an die Entwickler abdrehen:

```
global:
  sendAnonymousUsage: false
```

Zumindest während der Alpha- und Beta-Phase sollten Sie darauf aber freundlicherweise verzichten.

Zusätzlich zu dieser statischen Konfigurationsdatei, die nur bei einem Neustart des Traefik-Containers gelesen wird, gibt es einen dynamischen File-Provider, der jede Änderung sofort ausführt. Alles, was Sie per Label festgelegt haben, können Sie auch in einer Datei definieren. Sinnvoll ist es zum Beispiel, die Middlewares stattdessen in einer Datei anzulegen und den Routern dann per Label zuzuweisen. Damit eine dynamische Konfigurationsdatei ausgelesen wird, muss der File-Provider in der statischen Konfiguration erst einmal aktiviert werden. Fügen Sie dazu folgenden Schnipsel an die traefik.yml an:

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

Per Docker-Volume gelangt auch diese Datei in den Container:

```
- ./dynamic.yml:/etc/↵
 ↳traefik/dynamic/traefik.yml
```

In dieser Datei könnte man jetzt eine Middleware mit dem Namen „demo-whitelist“ anlegen:

```
http:
  middlewares:
    demo-whitelist:
      ipWhiteList:
        sourceRange:
        - 127.0.0.1
        - 192.168.0.10
```

Mit dem vollständigen Namen „demo-whitelist@file“ kann die Middleware jetzt an einen oder mehrere Router gebunden werden.

Verschlüsselt

Die letzte große Baustelle ist die Beschaffung von Zertifikaten bei Let’s Encrypt.

Um das nachzuvollziehen, müssen Sie auf einen Server mit öffentlicher IP wechseln. Traefik unterstützt alle Verfahren, die Let's Encrypt kennt. Das häufigste Verfahren, das auch in diesem Artikel beschrieben ist, ist noch immer die HTTP-Challenge, bei der der Server nach der Zertifikatsbestellung auf Port 80 einen Textschnipsel ausliefern muss. TLS-Challenge und DNS-Challenge, mit der auch Wildcard-Zertifikate ausgestellt werden können, werden in der Dokumentation beschrieben. Damit das HTTP-Verfahren klappt, brauchen Sie zunächst Entry-Points für Ports 80 und 443, und die Einstellungen für das ACME-Verfahren, die Sie in der statischen Konfigurationsdatei eintragen:

```
entryPoints:
  web:
    address: ":80"
  web-secure:
    address: ":443"
certificatesResolvers:
  sample:
    acme:
      email: admin@example.org
      storage: acme.json
      httpChallenge:
        # used during the cxallenge
        entryPoint: web
```

Passen Sie die E-Mail-Adresse an. Sie wird nicht im Zertifikat angezeigt und nur von Let's Encrypt verwendet, um Sie über ein ablaufendes Zertifikat zu informieren. Soweit lässt es Traefik aber für gewöhnlich nicht kommen und versucht, 30 Tage vor Ablauf ein neues Zertifikat zu bestellen. Auch in der Docker-Compose-Datei müssen die Ports 80 und 443 jetzt eingetragen sein.

Damit das Zertifikat einen Neustart des Containers übersteht, muss die Datei „acme.json" als Volume übergeben werden:

```
- ./acme.json:acme.json
```

Bei dieser Datei stellt sich Traefik aus gutem Grund etwas an: Aus Sicherheitsgründen müssen die Zugriffsrechte begrenzt sein. Legen Sie die Datei an und schränken Sie die Zugriffsrechte mit `chmod 600 acme.json` ein.

Der Container, der die Webseite enthält, braucht jetzt nur noch zwei Labels:

```
- traefik.http.routers.web2.rule=↴
↳Host(`web2.example.org`)
- traefik.http.routers.web2.tls=true
```

Anhand der Route, die auf einen Host festgelegt ist, erkennt Traefik den Bedarf an einem passenden Zertifikat und beginnt mit der Beschaffung. Da kaum ein Benutzer auf die Idee kommt, eine Adresse mit „https://" in seinen Browser einzutippen, braucht es noch eine kleine Umleitung. Anders als in Traefik 1 kommt auch dafür eine Middleware zum Einsatz. Mit folgendem YAML-Schnipsel wird alles, was auf Port 80 ankommt, auf Port 443 verwiesen:

```
http:
  routers:
    https-redirect:
      rule: 'HostRegexp(`{any:.*}`)'
      middlewares:
        - https-redirect
      service: redirect-all
  middlewares:
    https-redirect:
      redirectscheme:
        scheme: https
  services:
    redirect-all:
      LoadBalancer:
        servers:
          - url: ''
```

Beobachten

Mit den vorgestellten Handgriffen ist nur ein kleiner Teil des Funktionsumfangs von Traefik 2.0 beschrieben, vor allem das TCP-Routing eröffnet neue Möglichkeiten: Auch Mail-, MQTT-, oder FTP-Server finden damit ein sicheres Zuhause mit zwangsweise aktiviertem TLS, das an zentraler Stelle terminiert wird. Mit Traefik 1 muss man sich noch damit behelfen, die Zertifikate von Traefik abzuholen, aufwendig zu extrahieren und in andere Container zu kopieren.

Wenn das Basis-Setup läuft, lohnt auf alle Fälle ein Blick auf die Logging- und Metrik-Funktionen von Traefik. Von Haus aus kann die Software alle Zugriffe mit bekannten Tools wie InfluxDB [3] oder Prometheus protokollieren. Letzte große Baustelle in der Beta-Version ist das Web-Frontend. Um dieses zu aktivieren, reicht es aus, in der Docker-Compose-Datei den `command:` um `--api` zu erweitern. Außerdem muss Port 8080 per Docker-Portzuweisung mit der Netzwerkkarte verbunden werden. Aktuell kann man allerdings noch keine Middleware einschalten, um das Frontend abzusichern. Daher sollten Sie die Portweiterleitung noch nicht auf einem Server aktivieren, der öffentlich im Internet hängt.

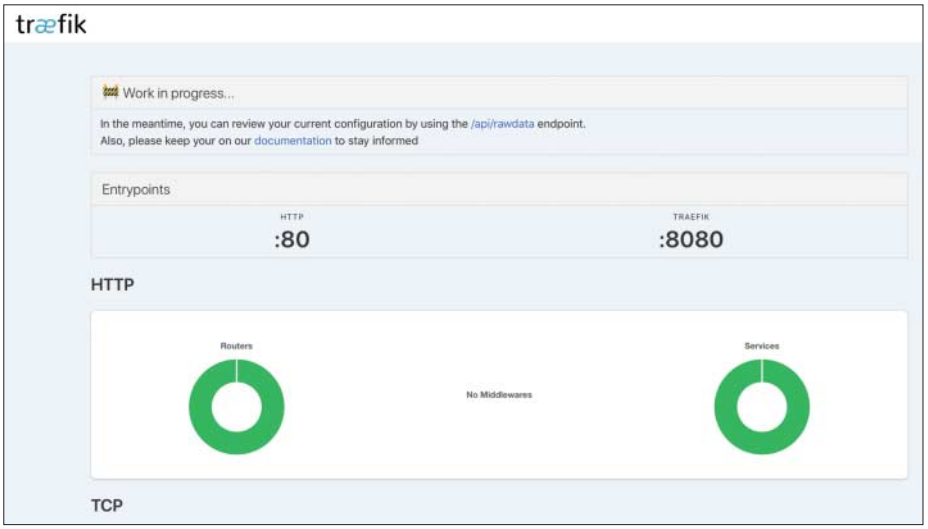
Zusammen mit dem Frontend aktiviert man ein API, über das man den Zustand der Traefik-Installation überwachen kann. Wenn Traefik das zentrale Eingangstor für viele Dienste ist, kann das für den Überblick über alle Routen und Regeln ein nützliches Feature werden.

(jam@ct.de) c't

Literatur

[1] Jan Mahn, Container-Dolmetscher, Hinter den Kulissen der Docker-Kommandozeile, c't 6/2019, S. 156
[2] Peter Siering, Schwärmchen, Docker Swarm: Container verteilen und verwalten, c't 5/2019, S. 142
[3] Jan Mahn, Geschichtsschreiber, InfluxDB: Spezialisierte Datenbank für Messwerte und Logging, c't 5/2019, S. 154

Dokumentation und Beispiele: **ct.de/y1ds**



Die grafische Oberfläche ist in der Beta-Phase noch eine Baustelle.