Traefik 中文文档手册

书栈(BookStack.CN)

目　录

致谢

当前文档 《Traefik 中文文档手册》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2019-04-16。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

内容来源：Traefik中国 https://docs.traefik.cn/

文档地址：http://www.bookstack.cn/books/traefik

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

Træfik 是一个为了让部署微服务更加便捷而诞生的现代HTTP反向代理、负载均衡工具。 它支持多种后台 (Docker, Swarm, Kubernetes, Marathon, Mesos, Consul, Etcd, Zookeeper, BoltDB, Rest API, file…) 来自动化、动态的应用它的配置文件设置。

概览

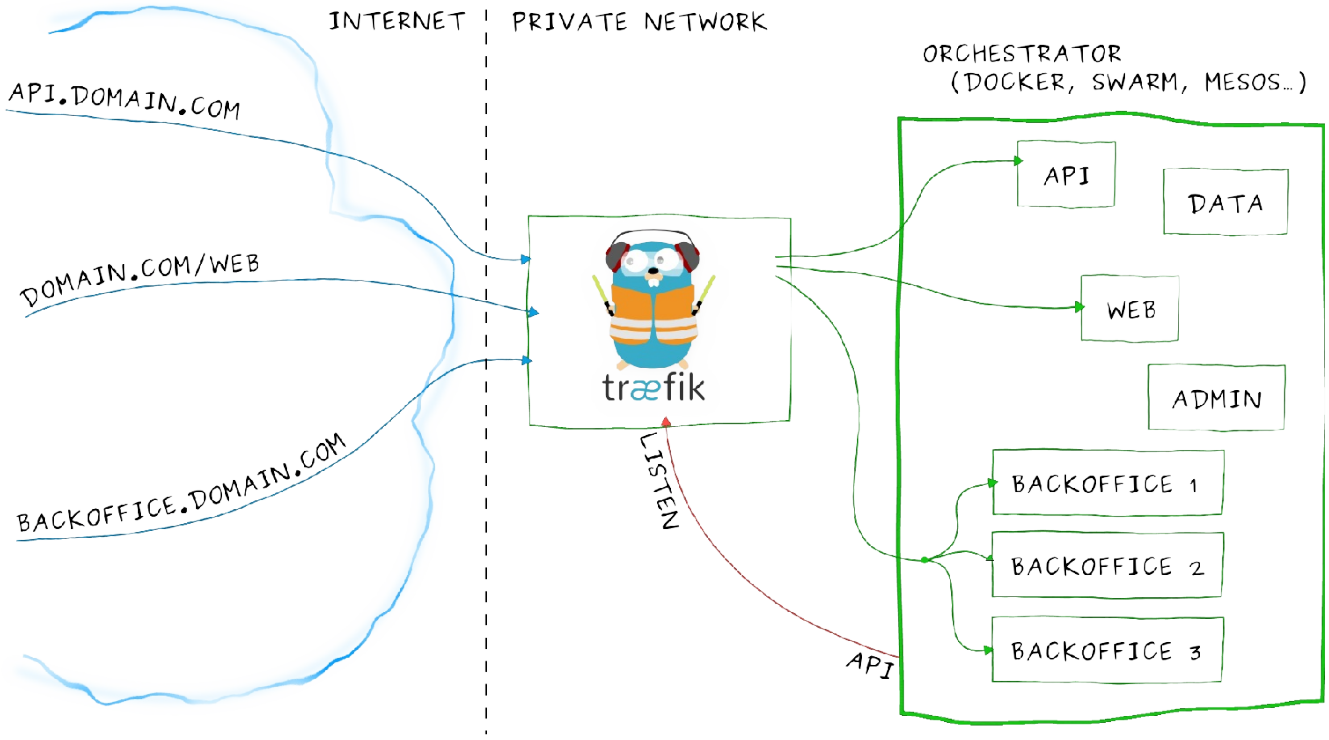
假设你已经在你的基础设施上部署了一堆微服务。你可能使用了一个服务发现系统（例如 etcd 或 consul）或者一个资源管理框架（swarm，Mesos/Marathon）来管理所有这些服务。 如果你想让你的用户去从互联网访问你的某些微服务， 你就必需使用虚拟hosts或前缀路径来配置一个反向代理:

- 域名 `api.domain.com` 将指向你的私有网络中的微服务 `api`
- 路径 `domain.com/web` 将指向你的私有网络中的微服务 `web`
- 域名 `backoffice.domain.com` 将指向你的私有网络中的微服务 `backoffice` ，在你的多

台实例之间负载均衡
但一个微服务的结构时动态的。。。 服务在会经常被添加、移除、杀死或更新，可能一天之内就会发生许多次。

传统的反向代理原生不支持动态配置。你不可能轻易的通过热更新更改它们的配置。

这时，Træfik就诞生了。



Træfik 可以监听你的服务发现、管理API，并且每当你的微服务被添加、移除、杀死或更新都会被感知，并且可以自动生成它们的配置文件。 指向到你服务的路由将会被直接创建出来。

运行它并忘记它吧！

快速上手

你可以通过这个 Katacoda 教程 快速感受Traefik是如何在多个Docker容器间负载均衡的。

这里有一个 Ed Robinson 在 ContainerCamp UK 会议中的演讲。 你将从中发现 Træfik 的基本特性并看到更多Træfik与Kubernetes的示例。

这里有一个 Emile Vauge 在法国 Devoxx France 2016 会议中的演讲。 你将从中发现 Træfik 的基本特性并看到更多Træfik与Docker， Mesos/Marathon 和 Let’s Encrypt的示例。

获取

二进制文件

从 版本下载 页面下载最新的可执行文件并以这个 示例配置文件 运行:

```
./traefik -c traefik.toml
```

Docker

通过Docker镜像:

```
docker run -d -p 8080:8080 -p 80:80 -v
$PWD/traefik.toml:/etc/traefik/traefik.toml traefik
```

测试

你可以简单的通过Docker compose测试 Træfik, 将这个 `docker-compose.yml` 文件放在名称叫做 `traefik` 的目录下:

```
version: '2'

services:
  proxy:
    image: traefik
    command: --web --docker --docker.domain=docker.localhost --logLevel=DEBUG
    networks:
      - webgateway
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /dev/null:/traefik.toml

networks:
  webgateway:
```

```
    driver: bridge
```

在名称叫做 `traefik` 的目录下运行：

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

在浏览器中你可以打开 `http://localhost:8080` 来访问 Træfık 的控制后台来发现下面的魔法。

现在，创建一个名称为 `test` 的目录，并在目录中使用以下内容创建一个 `docker-compose.yml` 文件：

```
version: '2'

services:
  whoami:
    image: emilevauge/whoami
    networks:
      - web
    labels:
      - "traefik.backend=whoami"
      - "traefik.frontend.rule=Host:whoami.docker.localhost"

networks:
  web:
    external:
      name: traefik_webgateway
```

然后，在 `test` 目录下按顺序执行以下命令：

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

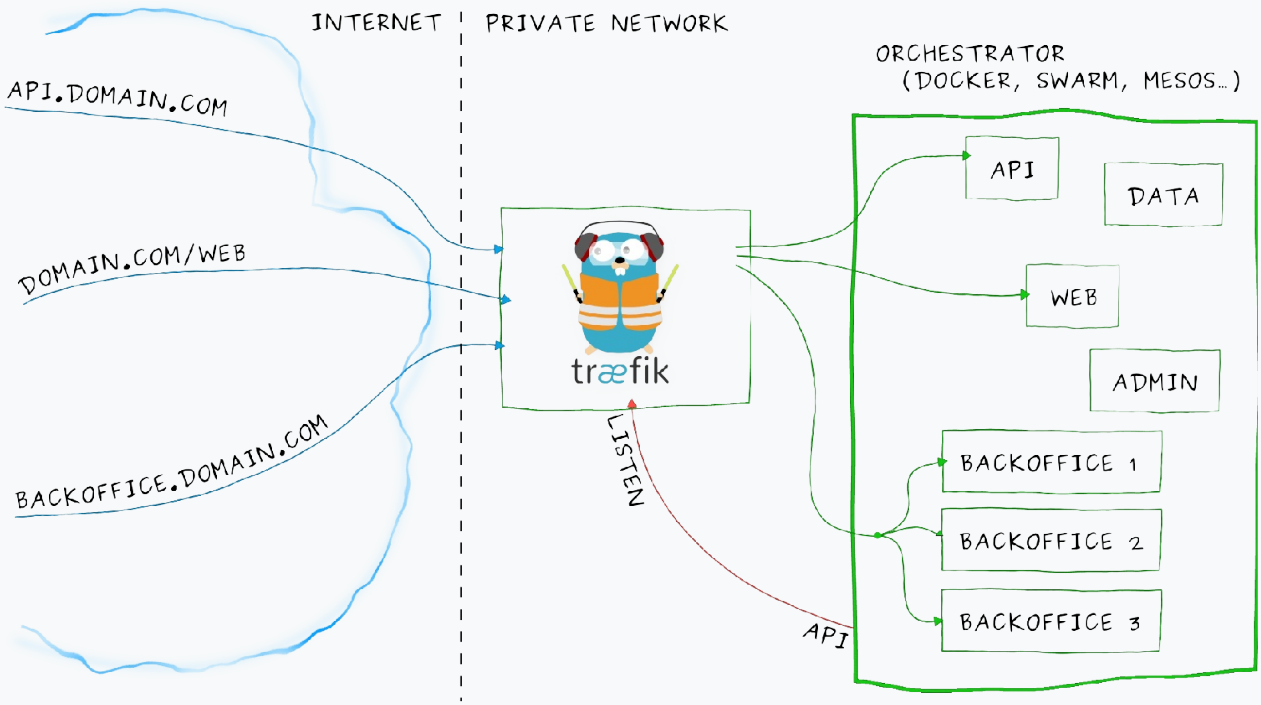
最后，测试 `test_whoami_1` 和 `test_whoami_2` 这两个服务之间的负载均衡：

```
$ curl -H Host:whoami.docker.localhost http://127.0.0.1
Hostname: ef194d07634a
IP: 127.0.0.1
IP: ::1
IP: 172.17.0.4
IP: fe80::42:acff:fe11:4
GET / HTTP/1.1
```

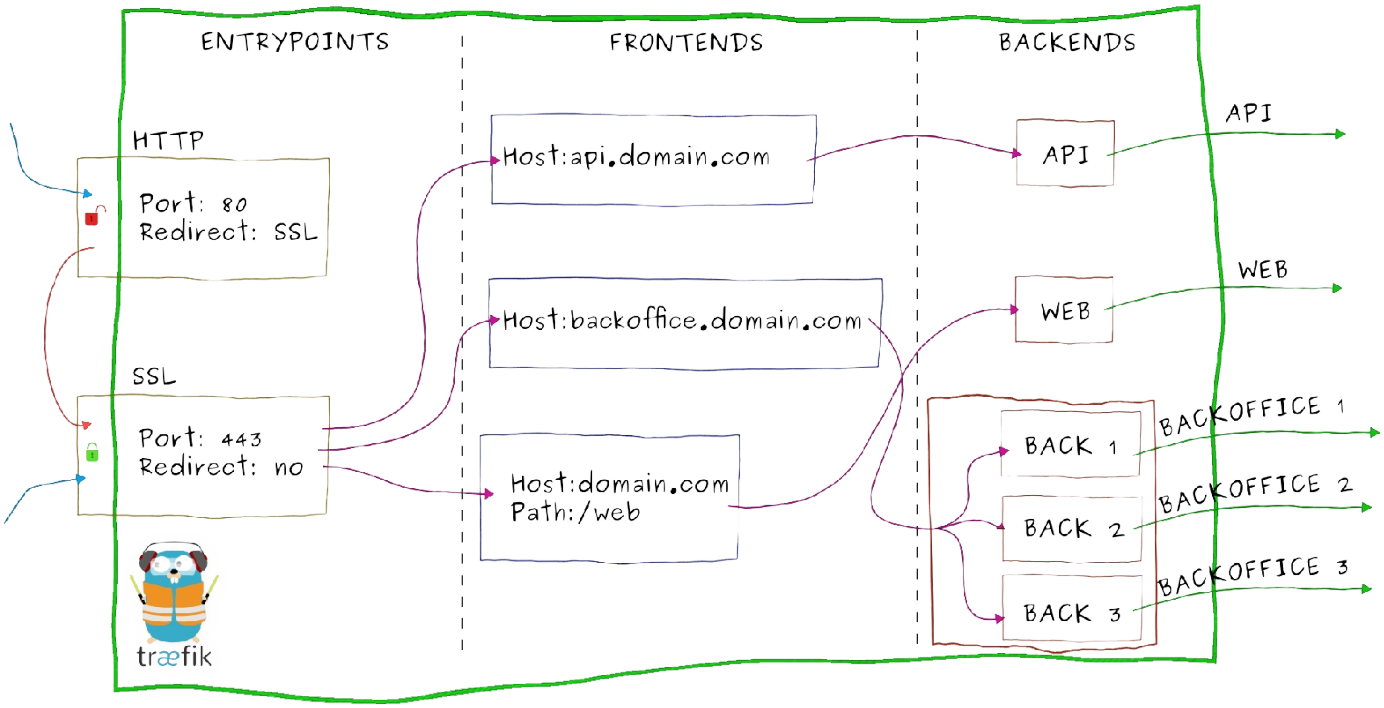
```
Host: 172.17.0.4:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 172.17.0.1
X-Forwarded-Host: 172.17.0.4:80
X-Forwarded-Proto: http
X-Forwarded-Server: dbb60406010d

$ curl -H Host:whoami.docker.localhost http://127.0.0.1
Hostname: 6c3c5df0c79a
IP: 127.0.0.1
IP: ::1
IP: 172.17.0.3
IP: fe80::42:acff:fe11:3
GET / HTTP/1.1
Host: 172.17.0.3:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 172.17.0.1
X-Forwarded-Host: 172.17.0.3:80
X-Forwarded-Proto: http
X-Forwarded-Server: dbb60406010d
```

概念

让我们回顾一下在 概览 举的例子：

> 假设你已经在你的基础设施上部署了一堆微服务。你可能使用了一个服务发现系统（例如 etcd 或 consul）或者一个资源管理框架（swarm，Mesos/Marathon）来管理所有这些服务。 如果你想让你的用户去从互联网访问你的某些微服务， 你就必需使用虚拟hosts或前缀路径来配置一个反向代理： - 域名 `api.domain.com` 将指向你的私有网络中的微服务 `api` - 路径 `domain.com/web` 将指向你的私有网络中的微服务 `web` - 域名 `backoffice.domain.com` 将指向你的私有网络中的微服务 `backoffice` ，在你的多台实例之间负载均衡
>
> 
>

我们将Træfik放大，一起看看它内部的结构：



- 请求在入口点处结束，顾名思义，它们是Træfik的网络入口(监听端口，SSL，流量重定向…)。
- 之后流量会导向一个匹配的前端。 前端是定义入口点 到 后端之间的路由的地方。 路由是通过请求字段(`Host` , `Path` , `Headers` …) 来定义的，它可以匹配或否定一个请求。
- 前端 将会把请求发送到 后端。后端可以由一台或一个通过负载均衡策略配置后的多台服务器组成。
- 最后， 服务器 将转发请求到对应私有网络的微服务当中去。

入口点

入口点是是Træfik的网络入口。 它们可以通过以下方式来定义:

- 一个端口 (80, 443…)
- SSL (证书, 密钥, 由受信任的CA签名的客户端证书的身份验证…)
- 重定向到其他的入口点 (重定向 `HTTP` 到 `HTTPS`)
  这里有一个入口点定义的例子:

```
[entryPoints]
  [entryPoints.http]
  address = ":80"
    [entryPoints.http.redirect]
    entryPoint = "https"
  [entryPoints.https]
  address = ":443"
```

```
    [entryPoints.https.tls]
      [[entryPoints.https.tls.certificates]]
      certFile = "tests/traefik.crt"
      keyFile = "tests/traefik.key"
```

- 这里定义了两个入口点，`http` 和 `https`.
- `http` 监听 `80` 端口，`https` 监听 `443` 端口
- 我们通过提供一个证书和一个密钥在 `https` 中开启SSL。
- 并且我们转发所有的 `http` 入口点请求到 `https` 入口点。
  这里还有一个证书身份验证的例子：

```
[entryPoints]
  [entryPoints.https]
  address = ":443"
  [entryPoints.https.tls]
  clientCAFiles = ["tests/clientca1.crt", "tests/clientca2.crt"]
    [[entryPoints.https.tls.certificates]]
    certFile = "tests/traefik.crt"
    keyFile = "tests/traefik.key"
```

- 我们通过提供一个证书和一个密钥在 `https` 中开启SSL。
- 添加了一个或多个包含证书身份验证的PEM格式文件
- 多个CA签名仔同一个文件或多个独立文件都是允许的。

前端

前端是入口流量从入口点转发到后端的一组规则。 前端可以通过以下规则定义：

- `Headers: Content-Type, application/json`：通过 Headers 可以添加一个匹配规则来匹配请求头部包含的值。它接受要匹配的键/值对序列。
- `HeadersRegexp: Content-Type, application/(text|json)`：也可以在 Headers 中使用正则表达式。它接受要匹配的键/值对序列，序列内容解析是通过正则匹配的。
- `Host: traefik.io, www.traefik.io`：匹配请求 Host 必需在给定域名列表内。
- `HostRegexp: traefik.io, {subdomain:[a-z]+}.traefik.io`：添加匹配请求 Host 的正则表达式。 它接受一个以 `{}` 包括起来的为空或更多url变量的模版。变量的值可以以一个可选的正则表达式来匹配。
- `Method: GET, POST, PUT`：Method 可以添加一个HTTP请求方法的匹配。它接受要匹配的一个或多个请求方法序列。
- `Path: /products/, /articles/{category}/{id:[0-9]+}`：Path 可以添加一个URL路径的匹配。它接受一个以 `{}` 包括起来的为空或更多url变量的模版。

- `PathStrip` : 和 `Path` 相同，但从请求的URL路径中去掉的给定的前缀。
- `PathPrefix` : PathPrefix 可以添加一个URL路径前缀的匹配。它匹配给定模版中的完整URL路径前缀。
- `PathPrefixStrip` : 和 `PathPrefix` 相同，但从请求的URL路径中去掉的给定的前缀。
- `AddPrefix` : 为请求URL路径天假前缀。
  你可以为一条规则添加多个值，通过用 `,` 分隔开。 你可以使用多条规则，通过用 `;` 分隔开。

你可以选择启用 `passHostHeader` 来转发客户端请求Header中的 `Host` 字段到后端

这里有一个前端定义的例子:

```
[frontends]
  [frontends.frontend1]
  backend = "backend2"
    [frontends.frontend1.routes.test_1]
    rule = "Host:test.localhost,test2.localhost"
  [frontends.frontend2]
  backend = "backend1"
  passHostHeader = true
  priority = 10
  entrypoints = ["https"] # overrides defaultEntryPoints
    [frontends.frontend2.routes.test_1]
    rule = "HostRegexp:localhost,{subdomain:[a-z]+}.localhost"
  [frontends.frontend3]
  backend = "backend2"
    [frontends.frontend3.routes.test_1]
    rule = "Host:test3.localhost;Path:/test"
```

- 这里定义了3个前端: `frontend1` , `frontend2` 和 `frontend3`
- 当规则 `Host:test.localhost,test2.localhost` 匹配时 `frontend1` 将把流量转发到 `backend2`
- 当规则 `Host:localhost,{subdomain:[a-z]+}.localhost` 匹配时 `frontend2` 将把流量转发到 `backend1` (转发客户端Header中的 `Host` 字段到后端)
- 当规则 `Host:test3.localhost` 与 `Path:/test` 同时匹配时 `frontend3` 将把流量转发到 `backend2`

合并多条规则

正如上面例子中所展示的，你可以合并多条规则。 在TOML文件中，你可以使用多条路由:

```
  [frontends.frontend3]
```

```
  backend = "backend2"
    [frontends.frontend3.routes.test_1]
    rule = "Host:test3.localhost"
    [frontends.frontend3.routes.test_2]
    rule = "Path:/test"
```

这里，当规则 `Host:test3.localhost` 与 `Path:/test` 同时匹配时 `frontend3` 将把流量转发到 `backend2` 。 或者你也可以使用 `;` 符号来分隔，结果是相同的：

```
  [frontends.frontend3]
  backend = "backend2"
    [frontends.frontend3.routes.test_1]
    rule = "Host:test3.localhost;Path:/test"
```

最后，你可以使用 `,` 符号分隔规则，为一个前端创建一个规则来绑定多个域名或路径：

```
 [frontends.frontend2]
    [frontends.frontend2.routes.test_1]
    rule = "Host:test1.localhost,test2.localhost"
  [frontends.frontend3]
  backend = "backend2"
    [frontends.frontend3.routes.test_1]
    rule = "Path:/test1,/test2"
```

优先级

默认情况下，路由会以规则长度（为了防止部分重叠情况）被排序（倒序）： `PathPrefix:/12345` 将会比 `PathPrefix:/1234` 优先被匹配到，最后才会匹配到 `PathPrefix:/1` 。

你可以在前端自定义优先级：

```
  [frontends]
    [frontends.frontend1]
    backend = "backend1"
    priority = 10
    passHostHeader = true
      [frontends.frontend1.routes.test_1]
      rule = "PathPrefix:/to"
    [frontends.frontend2]
    priority = 5
    backend = "backend2"
    passHostHeader = true
```

```
12.       [frontends.frontend2.routes.test_1]
13.       rule = "PathPrefix:/toto"
```

这里， `frontend1` 将会比 `frontend2` 优先被匹配（因为 `10 > 5` ）.

后端

后端用来负责将来自一个或多个前端的流量负载均衡到一组http服务器上。 这里支持多种负载均衡方法：

- `wrr` ： 加权轮询
- `drr` ： 动态轮询： 这会为表现比其他服务器好的服务器增加权重。当服务器表现有变化的时，它也会会退到正常权重。
  断路器也可以应用到后端，用于防止故障服务器上的高负载。 初始化状态是Standby。断路器只观察统计信息但并不修改请求。 当断路条件匹配时，断路器进入Tripped状态，它会返回与定义的http状态码或转发到其他前端。 一旦Tripped状态计时器超时，断路器会进入Recovering状态并重置所有统计数据。 当短路条件不匹配并且Recovery状态计时器超时时，断路器进入Standby状态。

它可以通过配置：

- 方法： `LatencyAtQuantileMS` , `NetworkErrorRatio` , `ResponseCodeRatio`
- 操作符： `AND` , `OR` , `EQ` , `NEQ` , `LT` , `LE` , `GT` , `GE`
  举个例子：
- `NetworkErrorRatio() > 0.5` ： 监控网络故障率大于0.5超过10秒后，为这个前端平滑切换，断路条件匹配
- `LatencyAtQuantileMS(50.0) > 50` ： 监控延迟超过50ms时断路条件匹配
- `ResponseCodeRatio(500, 600, 0, 600) > 0.5` ： 监控返回 `HTTP状态码在[500-600]之间的数量/HTTP状态码在[0-600]之间的数量` 的比例大于0.5时，断路条件匹配
  为了主动防治后端被高负载压垮，可以为每个后端设置最大连接数限制。

最大连接数限制可以通过为 `maxconn.amount` 配置一个整型值，同时 `maxconn.extractorfunc` 是用来配置通过什么样的维度来统计最大连接数。"例如下面例子中通过请求中host来统计连接数"

例如：

```
1.  [backends]
2.    [backends.backend1]
3.      [backends.backend1.maxconn]
4.         amount = 10
```

```
5.          extractorfunc = "request.host"
```

- 当已经有10个请求是同一个Host Header时，下一个请求 `backend1` 将返回 `HTTP code 429 Too Many Requests` 。
- 另外一个可可行的 `extractorfunc` 维度配置是 `client.ip` ，它将通过统计客户端请求IP来统计连接数。
- 其实 `extractorfunc` 可以设置 `request.header.ANY_HEADER` 中的任意维度，它将以你提供的 `ANY_HEADER` 任意维度来统计连接数。
  所有的负载平衡器都支持粘滞会话(sticky sessions)。当粘滞会话被开启时，会有一个名称叫做 `_TRAEFIK_BACKEND` 的cookie在请求被初始化时被设置在请求初始化时。在随后的请求中，客户端会被直接转发到这个cookie中存储的后端（当然它要是健康可用的），如果这个后端不可用，将会指定一个新的后端。

例如：

```
1.  [backends]
2.    [backends.backend1]
3.      [backends.backend1.loadbalancer]
4.        sticky = true
```

服务器健康检查也是可配置的，当Traefik定期执行HTTP GET请求到后端时，后端返回的HTTP状态码不是200 OK，那么这个后端将被从负载均衡轮询列表中移除。健康检查可以以一个在后端URL后附加路径的路径地址与一个时间间隔（以 time.ParseDuration 所识别的格式给出）specifying how 配置多久健康检查应该执行一次（默认30秒）. 每个后端必需在5秒内回应健康检查。

当一个后端重新返回HTTP状态码200 OK时，将被重新添加回负载均衡轮询列表。

例如：

```
1.  [backends]
2.    [backends.backend1]
3.      [backends.backend1.healthcheck]
4.        path = "/health"
5.        interval = "10s"
```

服务器

服务器可以简单的被定义为 `URL` 。你也可以设置一个自定义的 `weight` 给每个服务器（这个 `weight` 将会被用于负载均衡）。

这里有一个关于后端和服务器的定义的例子：

```
[backends]
  [backends.backend1]
    [backends.backend1.circuitbreaker]
      expression = "NetworkErrorRatio() > 0.5"
    [backends.backend1.servers.server1]
    url = "http://172.17.0.2:80"
    weight = 10
    [backends.backend1.servers.server2]
    url = "http://172.17.0.3:80"
    weight = 1
  [backends.backend2]
    [backends.backend2.LoadBalancer]
      method = "drr"
    [backends.backend2.servers.server1]
    url = "http://172.17.0.4:80"
    weight = 1
    [backends.backend2.servers.server2]
    url = "http://172.17.0.5:80"
    weight = 2
```

- 定义了两个后端: `backend1` and `backend2`
- `backend1` 将把流量转发到两台服务器: `http://172.17.0.2:80&#34;` 权重 `10` 与 `http://172.17.0.3:80` 权重 `1` 并使用默认的 `wrr` 负载均衡策略。
- `backend2` 将把流量转发到两台服务器: `http://172.17.0.4:80&#34;` 权重 `1` 与 `http://172.17.0.5:80` 权重 `2` 并使用 `drr` 负载均衡策略。
- 一个断路器被添加到 `backend1` 使用 `NetworkErrorRatio() > 0.5` 表达式: 监控网络故障率大于0.5超过10秒为这个前端平滑切换，断路条件匹配

配置文件

Træfik的配置文件分为两部分:

- 静态 Træfik 配置 ，仅在启动时被加载。
- 动态 Træfik 配置 ，可被热更新（无需重启进程）。

静态 Træfik 配置

静态配置文件是一种全局配置文件，用来配置后端和入口点的连接。

可以通过许多方式配置Træfik，以下是各种配置方式的生效优先级。 上面的项目优先级大于下面的项目:

- Key-value 存储
- 参数
- 配置文件
- 默认
  这代表着参数会覆盖配置文件，Key-value存储会覆盖参数。

配置文件

在默认情况下， Træfik 会在以下几个地方寻找 `traefik.toml` 文件:

- `/etc/traefik/`
- `$HOME/.traefik/`
- `.` 工作目录
  你可以通过设置 `configFile` 参数来覆盖这种默认情况:

```
1.  $ traefik --configFile=foo/bar/myconfigfile.toml
```

请转到 全局配置 部分获取详细文档。

参数

每个参数（命令）在帮助部分都有定义:

```
1.  $ traefik --help
```

需要注意的是，所有默认值也会一同被展示出来。

Key-value 存储

Træfik 支持多种 Key-value 存储方式:

- Consul
- etcd
- ZooKeeper
- boltdb
  请转到 用户手册 Key-value 存储配置 部分获取详细文档。

动态 Træfik 配置

动态配置请关注:

- 前端
- 后端
- 服务器
  Træfik 可以将多个配置后端的规则热更新。

我们只需要开启 `watch` 选项来让 Træfik 监听配置文件变化并自动生成配置。 指向服务的路由在监测到任何变化时直接被创建或更新。

请转到后端配置 部分获取详细文档。

命令

使用方法： `traefik [command] [–flag=flag_argument]`

列出了所有Træfik可用的命令与命令描述：

- `version` ： 打印版本号
- `storeconfig` ： 将静态traefik配置存入Key-value存储。请转到存储 Træfik 配置 部分获取详细文档。
  每个命令都可能包含相关的标志。 所有相关的标志都会显示在：

```
1.  $ traefik [command] --help
```

需要注意的是，每个命令描述是在帮助的开始部分：

```
1.  $ traefik --help
```

全局配置文件

主体部分

```
# traefik.toml
###############################################################
# 全局配置文件
###############################################################

# 设置超时的时间（以秒为单位）
# 在热更新期间给还在活动中的请求来完成当前任务的超时时间
#
# 可选
# 默认: 10
#
# graceTimeOut = 10

# 开启调试模式
#
# 可选
# 默认: false
#
# debug = true

# 定期检查是否有新版本产生
#
# 可选
# 默认: true
#
# checkNewVersion = false

# Traefik 日志文件
# 如果没有定义, 日志文件输出到 stdout
#
# 可选
#
# traefikLogsFile = "log/traefik.log"

# 日志文件路径
#
# 可选
#
```

```
# accessLogsFile = "log/access.log"

# 日志等级
#
# 可选
# 默认: "ERROR"
# 接受以下值，按照严重程度排序: "DEBUG", "INFO", "WARN", "ERROR", "FATAL", "PANIC"
# 日志等级在配置值或配置值以上的信息将被日志记录。
#
# logLevel = "ERROR"

# 后端节流持续时间：在应用新配置之前，提供者的2个事件之间的最短持续时间（以秒为单位）
# 如果在短时间内发送多个事件，它可以避免不必要的重新加载。
#
# 可选
# 默认: "2"
#
# ProvidersThrottleDuration = "5"

# 为每个host控制最大空闲连接（keep-alive）。如果设置为0，那么将会使用
# Go语言基础库net/http中的DefaultMaxIdleConnsPerHost。
# 如果造成 'too many open files' 错误，你也可以增加这个值或改变 `ulimit`。
#
# 可选
# 默认: 200
#
# MaxIdleConnsPerHost = 200

# 如果设置为 true，无效的 SSL 证书也会被后端所接受。If set to true invalid SSL certificates are accepted for backends.
# 注意：这会禁用中间人攻击（man-in-the-middle attacks）监测，所以只能被用在安全的后端网络中。
#
# 可选
# 默认: false
#
# InsecureSkipVerify = true

# 被前端所使用的入口点未指定任何入口点。
# 任何前端可以指定它自己的入口点。
#
# 可选
```

```
# 默认: ["http"]
#
# defaultEntryPoints = ["http", "https"]
```

约束

在一个以中央服务发现的微服务架构中，配置文件会将Træfik的发现范围约束到一小部分路由上。

Træfik 根据你在配置后端时为服务设置的属性/标签来过滤服务。

支持的后端类型：

- Docker
- Consul K/V
- BoltDB
- Zookeeper
- Etcd

- Consul Catalog 支持的过滤方式：

- `tag`

```
# 定义约束条件
#
# 可选
#
# 简单约束匹配的条件
# constraints = ["tag==api"]
#
# 简单约束不匹配的条件
# constraints = ["tag!=api"]
#
# 约束全局匹配条件
# constraints = ["tag==us-*"]
#
# 后端指定约束条件
# [consulCatalog]
#   endpoint = 127.0.0.1:8500
#   constraints = ["tag==api"]
#
# 多个约束条件
#   - "tag==" 必需匹配至少一个标签
#   - "tag!=" 必需不匹配任何标签
```

```
# constraints = ["tag!=us-*", "tag!=asia-*"]
# [consulCatalog]
#   endpoint = 127.0.0.1:8500
#   constraints = ["tag==api", "tag!=v*-beta"]
```

入口点定义

```
# 入口点定义
#
# 可选
# 默认:
# [entryPoints]
#   [entryPoints.http]
#   address = ":80"
#
# 将一个http入口点转发到一个https入口点上(保留 SNI 支持)：
# [entryPoints]
#   [entryPoints.http]
#   address = ":80"
#     [entryPoints.http.redirect]
#       entryPoint = "https"
#   [entryPoints.https]
#   address = ":443"
#     [entryPoints.https.tls]
#       [[entryPoints.https.tls.certificates]]
#       CertFile = "integration/fixtures/https/snitest.com.cert"
#       KeyFile = "integration/fixtures/https/snitest.com.key"
#       [[entryPoints.https.tls.certificates]]
#       CertFile = "integration/fixtures/https/snitest.org.cert"
#       KeyFile = "integration/fixtures/https/snitest.org.key"
#
# 将一个入口点转发重写为URL：
# [entryPoints]
#   [entryPoints.http]
#   address = ":80"
#     [entryPoints.http.redirect]
#       regex = "^http://localhost/(.*)"
#       replacement = "http://mydomain/$1"
#
# 只接受提供了被指定证书认证机构（CA）签发的证书的客户端。
# ClientCAFiles 可以在同一个文件被配置为多个CA证书或使用一个或多个文件配置多个CA证书。
```

```
35. # CA证书必需是PEM文件格式的。
36. # 所有客户端将被要求提供一个有效的证书。
37. # 这个队客户端的要求会被应用到该入口点下的所有服务器证书
38. # 在下面的例子中 snitest.com 与 snitest.org 都需要提供客户端证书
39. #
40. # [entryPoints]
41. #   [entryPoints.https]
42. #   address = ":443"
43. #   [entryPoints.https.tls]
44. #   ClientCAFiles = ["tests/clientca1.crt", "tests/clientca2.crt"]
45. #     [[entryPoints.https.tls.certificates]]
46. #     CertFile = "integration/fixtures/https/snitest.com.cert"
47. #     KeyFile = "integration/fixtures/https/snitest.com.key"
48. #     [[entryPoints.https.tls.certificates]]
49. #     CertFile = "integration/fixtures/https/snitest.org.cert"
50. #     KeyFile = "integration/fixtures/https/snitest.org.key"
51. #
52. # 要为一个入口点开启基础认证（basic auth）
53. # 使用2组用户名/密码: test:test 与 test2:test2
54. # 密码可以以MD5、SHA1或BCrypt方式加密：你可以使用htpasswd来生成这些用户名密码。
55. # [entryPoints]
56. #   [entryPoints.http]
57. #   address = ":80"
58. #   [entryPoints.http.auth.basic]
59. #   users = ["test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/",
    "test2:$apr1$d9hr9HBB$4HxwgUir3HP4EsggP/QNo0"]
60. #
61. # 要为一个入口点开启摘要认证（digest auth）
62. # 使用2组用户名/域/密码： test:traefik:test 与 test2:traefik:test2
63. # 你可以使用htdigest来生成这些用户名/域/密码
64. # [entryPoints]
65. #   [entryPoints.http]
66. #   address = ":80"
67. #   [entryPoints.http.auth.basic]
68. #   users = ["test:traefik:a2688e031edb4be6a3797f3882655c05 ",
    "test2:traefik:518845800f9e2bfb1f1f740ec24f074e"]
69. #
70. # 要指定一个https入口点使用一个较低的TLS版本并且指定一组密钥（在crypto/tls包中定义）：
71. # [entryPoints]
72. #   [entryPoints.https]
73. #   address = ":443"
74. #     [entryPoints.https.tls]
```

```
#     MinVersion = "VersionTLS12"
#     CipherSuites = ["TLS_RSA_WITH_AES_256_GCM_SHA384"]
#       [[entryPoints.https.tls.certificates]]
#       CertFile = "integration/fixtures/https/snitest.com.cert"
#       KeyFile = "integration/fixtures/https/snitest.com.key"
#       [[entryPoints.https.tls.certificates]]
#       CertFile = "integration/fixtures/https/snitest.org.cert"
#       KeyFile = "integration/fixtures/https/snitest.org.key"

# 要启用gzip格式压缩支持：
# [entryPoints]
#   [entryPoints.http]
#   address = ":80"
#   compress = true

[entryPoints]
  [entryPoints.http]
  address = ":80"
```

重试配置

```
# 当网络故障时启用重发请求
#
# 可选
#
[retry]

# 重试次数
#
# 可选
# 默认: （后端服务器数量） -1
#
# attempts = 3
```

ACME (Let's Encrypt) 配置

```
# 使用ACME的入口点配置样本
[entryPoints]
  [entryPoints.https]
  address = ":443"
```

```
    [entryPoints.https.tls]

# 启用 ACME (Let's Encrypt): 自动 SSL
#
# 可选
#
[acme]

# 用于注册的邮箱地址
#
# 必需
#
email = "[email protected]"

# 证书存储使用的文件或键。
# 警告，如果你在Docker中使用Traefik你有两种选择：
#  - 在你的服务器上创建一个文件并作为一个卷挂载
#      storageFile = "acme.json"
#      $ docker run -v "/my/host/acme.json:acme.json" traefik
#  - 将包含这个文件的目录作为一个卷挂载
#      storageFile = "/etc/traefik/acme/acme.json"
#      $ docker run -v "/my/host/acme:/etc/traefik/acme" traefik
#
# 必需
#
storage = "acme.json" # or "traefik/acme/account" if using KV store

# 代理acme验证证书challenge/apply的入口点。
# 警告，必需指向到一个443端口作为入口点
#
# 必需
#
entryPoint = "https"

# 使用一个基于DNS的acme challenge而不是使用额外的HTTPS接入，例如，一个带有前置防火墙的服务器的情况可能无法完成验证
# 选择一个可以设置 challenge TXT 匹配记录的DNS域名的提供者，
# 并提供以下环境变量来启用它：
#  - cloudflare: CLOUDFLARE_EMAIL, CLOUDFLARE_API_KEY
#  - digitalocean: DO_AUTH_TOKEN
#  - dnsimple: DNSIMPLE_EMAIL, DNSIMPLE_API_KEY
#  - dnsmadeeasy: DNSMADEEASY_API_KEY, DNSMADEEASY_API_SECRET
```

```
#  - exoscale: EXOSCALE_API_KEY, EXOSCALE_API_SECRET
#  - gandi: GANDI_API_KEY
#  - linode: LINODE_API_KEY
#  - manual: 无，但是需要以可交互模式运行traefik & 打开acmeLogging来查看说明 & 按下回车
#  - namecheap: NAMECHEAP_API_USER, NAMECHEAP_API_KEY
#  - rfc2136: RFC2136_TSIG_KEY, RFC2136_TSIG_SECRET, RFC2136_TSIG_ALGORITHM, RFC2136_NAMESERVER
#  - route53: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_REGION, or configured user/instance IAM profile
#  - dyn: DYN_CUSTOMER_NAME, DYN_USER_NAME, DYN_PASSWORD
#  - vultr: VULTR_API_KEY
#  - ovh: OVH_ENDPOINT, OVH_APPLICATION_KEY, OVH_APPLICATION_SECRET, OVH_CONSUMER_KEY
#  - pdns: PDNS_API_KEY, PDNS_API_URL
#
# 可选
#
# dnsProvider = "digitalocean"

# 默认情况下，DNS提供者会在ACME验证前验证TXT DNS challenge
# 如果 delayDontCheckDNS 是一个比0大的值，跳过这里 & 否则配置这里就是等待多长时间（以秒为单位）
# 当内网屏蔽了附加的DNS查询时非常有用
#
# Optional
#
# delayDontCheckDNS = 0

# 如果设置为true，将显示由acme客户端代码库中产生的调试日志信息
#
# 可选
#
# acmeLogging = true

# 启用按需证书。如果这个主机名还没有证书，这将会在与一个主机名发起请求的第一个TLS握手中向Let's Encrypt请求一个证书。
# 警告，第一次在请求中获取主机证书会导致TLS握手会非常慢，这会引起Dos攻击。
# 警告，值得注意的是Let's Encrypt是有请求上限的：https://letsencrypt.org/docs/rate-limits
#
# 可选
#
```

```
# onDemand = true

# 启用根据前端Host规则来生成证书。这将会为每个具有Host规则的前端生成一个Let's Encrypt的证书。
# 举个例子，一个具有规则的Host:test1.traefik.cn,test2.traefik.cn 将会为主域名test1.traefik.cn与SAN(替代域名) test2.traefik.cn生成一个证书。
#
# 可选
#
# OnHostRule = true

# 所使用的CA服务器
# 取消注释这行来使其运行在正式环境的Let's Encrypt证书服务器上
# 在生产环境中请取消注释
#
# 可选
#
# caServer = "https://acme-staging.api.letsencrypt.org/directory"

# 域名列表
# 你可以为每个主域名提供多个SANs（替代域名）
# 所有域名的A/AAAA记录必需指向 Traefik
# 警告，值得注意的是Let's Encrypt是有请求上限的：https://letsencrypt.org/docs/rate-limits
# 每个域名 & SANs（替代域名）都将会指向一个证书请求。
#
# [[acme.domains]]
#   main = "local1.com"
#   sans = ["test1.local1.com", "test2.local1.com"]
# [[acme.domains]]
#   main = "local2.com"
#   sans = ["test1.local2.com", "test2x.local2.com"]
# [[acme.domains]]
#   main = "local3.com"
# [[acme.domains]]
#   main = "local4.com"
[[acme.domains]]
   main = "local1.com"
   sans = ["test1.local1.com", "test2.local1.com"]
[[acme.domains]]
   main = "local3.com"
[[acme.domains]]
```

```
121.     main = "local4.com"
```

配置后端

文件后端

类似其他的反向代理工具，Træfik也可以使用文件来配置。你有两种选择：

- 在全局配置文件 `traefik.toml` 后追加你的配置：

```
# traefik.toml
logLevel = "DEBUG"
defaultEntryPoints = ["http", "https"]
[entryPoints]
  [entryPoints.http]
  address = ":80"
    [entryPoints.http.redirect]
      entryPoint = "https"
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
      [[entryPoints.https.tls.certificates]]
      CertFile = "integration/fixtures/https/snitest.com.cert"
      KeyFile = "integration/fixtures/https/snitest.com.key"
      [[entryPoints.https.tls.certificates]]
      CertFile = "integration/fixtures/https/snitest.org.cert"
      KeyFile = "integration/fixtures/https/snitest.org.key"

[file]

# rules
[backends]
  [backends.backend1]
    [backends.backend1.circuitbreaker]
      expression = "NetworkErrorRatio() > 0.5"
    [backends.backend1.servers.server1]
    url = "http://172.17.0.2:80"
    weight = 10
    [backends.backend1.servers.server2]
    url = "http://172.17.0.3:80"
    weight = 1
  [backends.backend2]
    [backends.backend1.maxconn]
      amount = 10
      extractorfunc = "request.host"
```

```
    [backends.backend2.LoadBalancer]
      method = "drr"
    [backends.backend2.servers.server1]
    url = "http://172.17.0.4:80"
    weight = 1
    [backends.backend2.servers.server2]
    url = "http://172.17.0.5:80"
    weight = 2

[frontends]
  [frontends.frontend1]
  backend = "backend2"
    [frontends.frontend1.routes.test_1]
    rule = "Host:test.localhost"
  [frontends.frontend2]
  backend = "backend1"
  passHostHeader = true
  priority = 10
  entrypoints = ["https"] # 覆盖 defaultEntryPoints
    [frontends.frontend2.routes.test_1]
    rule = "Host:{subdomain:[a-z]+}.localhost"
  [frontends.frontend3]
  entrypoints = ["http", "https"] # 覆盖 defaultEntryPoints
  backend = "backend2"
    rule = "Path:/test"
```

- 或者将你的规则分割成独立的文件，例如 `rules.toml`：

```
# traefik.toml
logLevel = "DEBUG"
[entryPoints]
  [entryPoints.http]
  address = ":80"
    [entryPoints.http.redirect]
      entryPoint = "https"
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
      [[entryPoints.https.tls.certificates]]
      CertFile = "integration/fixtures/https/snitest.com.cert"
      KeyFile = "integration/fixtures/https/snitest.com.key"
      [[entryPoints.https.tls.certificates]]
```

```
      CertFile = "integration/fixtures/https/snitest.org.cert"
      KeyFile = "integration/fixtures/https/snitest.org.key"

[file]
filename = "rules.toml"
```

```
# rules.toml
[backends]
  [backends.backend1]
    [backends.backend1.circuitbreaker]
      expression = "NetworkErrorRatio() > 0.5"
    [backends.backend1.servers.server1]
    url = "http://172.17.0.2:80"
    weight = 10
    [backends.backend1.servers.server2]
    url = "http://172.17.0.3:80"
    weight = 1
  [backends.backend2]
    [backends.backend1.maxconn]
      amount = 10
      extractorfunc = "request.host"
    [backends.backend2.LoadBalancer]
      method = "drr"
    [backends.backend2.servers.server1]
    url = "http://172.17.0.4:80"
    weight = 1
    [backends.backend2.servers.server2]
    url = "http://172.17.0.5:80"
    weight = 2

[frontends]
  [frontends.frontend1]
  backend = "backend2"
    [frontends.frontend1.routes.test_1]
    rule = "Host:test.localhost"
  [frontends.frontend2]
  backend = "backend1"
  passHostHeader = true
  priority = 10
  entrypoints = ["https"] # overrides defaultEntryPoints
    [frontends.frontend2.routes.test_1]
    rule = "Host:{subdomain:[a-z]+}.localhost"
```

```
  [frontends.frontend3]
  entrypoints = ["http", "https"] # overrides defaultEntryPoints
  backend = "backend2"
    rule = "Path:/test"
```

如果你希望 Træfık 动态监测这些文件的变化，只需添加:

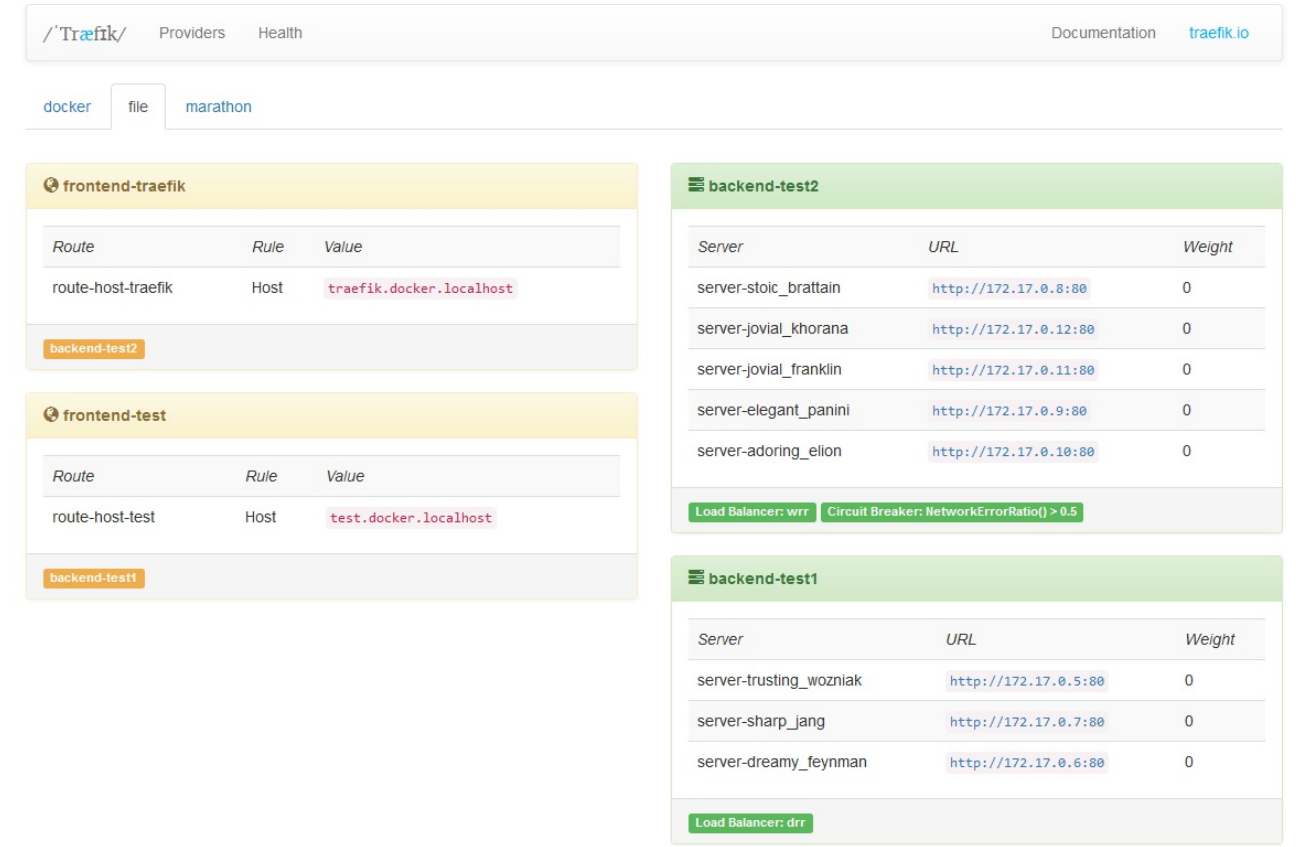
```
[file]
watch = true
```

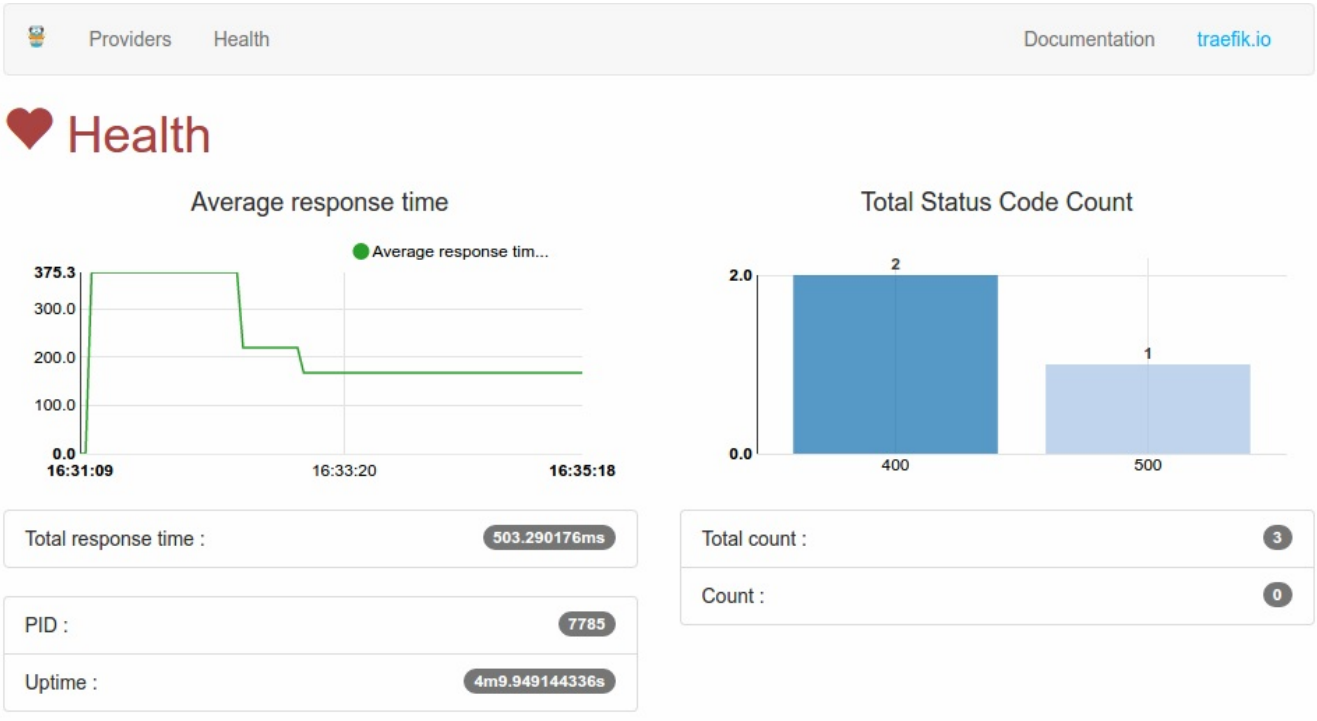
API 后端

Træfik 可以通过 RESTful api 进行配置. 想要开启它:

```
[web]
address = ":8080"

# SSL 证书与密钥文件配置
#
# 可选
#
# CertFile = "traefik.crt"
# KeyFile = "traefik.key"
#
# 将REST API设置为只读模式
#
# 可选
# ReadOnly = false
#
# 开启更多详细统计信息
# [web.statistics]
#   RecentErrors = 10
#
# 为Traefik启用向Prometheus提供内部维度
# [web.metrics.prometheus]
#   Buckets=[0.1,0.3,1.2,5]
#
# 在WebUI中启用基础认证（basic auth）
# 使用2组用户名/密码: test:test 与 test2:test2
# 密码可以以MD5、SHA1或BCrypt方式加密：你可以使用htpasswd来生成这些用户名密码。
#   [web.auth.basic]
#     users = ["test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/",
"test2:$apr1$d9hr9HBB$4HxwgUir3HP4EsggP/QNo0"]
# 为WebUI开启摘要认证（digest auth）
# 使用2组用户名/域/密码： test:traefik:test 与 test2:traefik:test2
# 你可以使用htdigest来生成这些用户名/域/密码
#   [web.auth.digest]
#     users = ["test:traefik:a2688e031edb4be6a3797f3882655c05 ",
"test2:traefik:518845800f9e2bfb1f1f740ec24f074e"]

```

- `/` ： 为Træfik提供一个简单的HTML前端页面





Recent HTTP Errors

Status	Request	Time
400 — Bad Request	GET	2 minutes ago
400 — Bad Request	GET	3 minutes ago
500 — Internal Server Error	GET	4 minutes ago

- `/ping` : `GET` 为Træfik提供一个简单的查看进程活跃度的查询端口。

```
$ curl -sv "http://localhost:8080/ping"
*   Trying ::1...
* Connected to localhost (::1) port 8080 (#0)
> GET /ping HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Aug 2016 01:35:36 GMT
< Content-Length: 2
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host localhost left intact
OK
```

- `/health` : `GET` json metrics

```
$ curl -s "http://localhost:8080/health" | jq .
{
  // Træfık 进程ID（PID）
  "pid": 2458,
  // Træfık 服务器运行时间（格式化的时间）
  "uptime": "39m6.885931127s",
  //  Træfık 服务器运行时间（以秒为单位）
  "uptime_sec": 2346.885931127,
  // 当前服务器日期
  "time": "2015-10-07 18:32:24.362238909 +0200 CEST",
  // 当前服务器时间（以秒为单位）
  "unixtime": 1444235544,
  // HTTP请求返回状态码实时统计
  "status_code_count": {
    "502": 1
  },
  // HTTP请求返回状态码实时统计（自Træfık启动以来）
  "total_status_code_count": {
    "200": 7,
    "404": 21,
    "502": 13
  },
```

```
  // 统计HTTP请求
  "count": 1,
  // 统计HTTP请求总数
  "total_count": 41,
  // 统计所有响应时间之和（格式化的时间）
  "total_response_time": "35.456865605s",
  // 统计所有响应时间之和（以秒为单位）
  "total_response_time_sec": 35.456865605,
  // 平均响应时间（格式化的时间）
  "average_response_time": "864.8016ms",
  // 平均响应时间（以秒为单位）
  "average_response_time_sec": 0.8648016000000001,

  // 请求统计 [需要设置 --web.statistics ]
  // 近10个响应状态码为 4xx 与 5xx 的请求
  "recent_errors": [
    {
      // 状态码
      "status_code": 500,
      // 状态码秒数
      "status": "Internal Server Error",
      // HTTP请求方法
      "method": "GET",
      // 请求域名
      "host": "localhost",
      // 请求路径
      "path": "/path",
      // RFC 3339 格式化的 日期/时间
      "time": "2016-10-21T16:59:15.418495872-07:00"
    }
  ]
}
```

- `/api`： `GET` 所有提供者的配置文件

```
$ curl -s "http://localhost:8080/api" | jq .
{
  "file": {
    "frontends": {
      "frontend2": {
        "routes": {
          "test_2": {
```

```
            "rule": "Path:/test"
          }
        },
        "backend": "backend1"
      },
      "frontend1": {
        "routes": {
          "test_1": {
            "rule": "Host:test.localhost"
          }
        },
        "backend": "backend2"
      }
    },
    "backends": {
      "backend2": {
        "loadBalancer": {
          "method": "drr"
        },
        "servers": {
          "server2": {
            "weight": 2,
            "URL": "http://172.17.0.5:80"
          },
          "server1": {
            "weight": 1,
            "url": "http://172.17.0.4:80"
          }
        }
      },
      "backend1": {
        "loadBalancer": {
          "method": "wrr"
        },
        "circuitBreaker": {
          "expression": "NetworkErrorRatio() > 0.5"
        },
        "servers": {
          "server2": {
            "weight": 1,
            "url": "http://172.17.0.3:80"
          },
```

```
          "server1": {
            "weight": 10,
            "url": "http://172.17.0.2:80"
          }
        }
      }
    }
  }
}
```

- `/api/providers` : `GET` 提供者
- `/api/providers/{provider}` : `GET` or `PUT` 提供者
- `/api/providers/{provider}/backends` : `GET` 后端列表
- `/api/providers/{provider}/backends/{backend}` : `GET` 一个后端
- `/api/providers/{provider}/backends/{backend}/servers` : `GET` 后端的服务器列表
- `/api/providers/{provider}/backends/{backend}/servers/{server}` : `GET` 后端的一个服务器
- `/api/providers/{provider}/frontends` : `GET` 前端列表
- `/api/providers/{provider}/frontends/{frontend}` : `GET` 一个前端
- `/api/providers/{provider}/frontends/{frontend}/routes` : `GET` 前端的路由列表
- `/api/providers/{provider}/frontends/{frontend}/routes/{route}` : `GET` 前端的一个路由
- `/metrics` : 你可以为Traefik开启向其他不同的监控系统输出的内部维度（当前只支持Prometheus）。

```
$ traefik --web.metrics.prometheus --
web.metrics.prometheus.buckets="0.1,0.3,1.2,5"
```

Docker 后端

Træfik可以使用Docker作为后端配置:

```
################################################################
# Docker 后端配置
################################################################

# 启用Docker后端配置
#
# 可选
#
[docker]

# Docker服务器端口。可以是一个TCP或者一个unix socket端口。
#
# 必需
#
endpoint = "unix:///var/run/docker.sock"

# 使用默认域名。
# 可以通过为容器设置"traefik.domain" label来覆盖。
#
# 必需
#
domain = "docker.localhost"

# 启用监控docker变化
#
# 可选
#
watch = true

# 覆盖默认配置文件模版。为高级用户：）
#
# 可选
#
# filename = "docker.tmpl"

# 默认将容器暴露给traefik
# 如果设置为false，没有设置`traefik.enable=true`的容器将被忽略
```

```
#
# 可选
# 默认: true
#
exposedbydefault = true

# 使用绑定端口的IP地址而不是使用内部网络。为了特殊使用场景：）

#
# 可选
# 默认: false
#
usebindportip = true
# 使用 Swarm Mode 服务作为数据提供者
#
# 可选
# 默认: false
#
swarmmode = false


# 启用docker TLS连接
#
#  [docker.tls]
#  ca = "/etc/ssl/ca.crt"
#  cert = "/etc/ssl/docker.crt"
#  key = "/etc/ssl/docker.key"
#  insecureskipverify = true
```

可以给容器用来覆盖默认表现方式的Label:

- `traefik.backend=foo`：将容器指向 `foo` 后端
- `traefik.backend.maxconn.amount=10`：设置后端连接的最大数量。必需与以下label配合使用才能生效。
- `traefik.backend.maxconn.extractorfunc=client.ip`：设置后端连接最大数量所依赖的维度。必需与上面的label配合使用才能生效。
- `traefik.backend.loadbalancer.method=drr`：覆盖默认的 `wrr` 负载均衡算法逻辑
- `traefik.backend.loadbalancer.sticky=true`：启用后端粘滞session
- `traefik.backend.loadbalancer.swarm=true`：使用 Swarm 内置的负载均衡器（只有使用Swarm Mode时生效）。
- `traefik.backend.circuitbreaker.expression=NetworkErrorRatio() > 0.5`：为后端创建一个 断路器 。

- `traefik.port=80` ： 注册使用这个端口。当容器暴露出多个端口时非常有效。
- `traefik.protocol=https` ： 覆盖默认的 `http` 协议
- `traefik.weight=10` ： 为容器指定权重
- `traefik.enable=false` ： 为Træfik禁用这个容器
- `traefik.frontend.rule=Host:test.traefik.io` ： 覆盖默认前端规则（默认： `Host:{containerName}.{domain}` ）。
- `traefik.frontend.passHostHeader=true` ： 将客户端header中的 `Host` 转发到后端。
- `traefik.frontend.priority=10` ： 覆盖默认的前端优先级
- `traefik.frontend.entryPoints=http,https` ： 将这个前端指向入口点 `http` 与 `https` 。 覆盖 `defaultEntryPoints` 。
- `traefik.docker.network` ： 设置连接到这个容器的docker网络。 如果容易被链接到多个网络，一定要设置合适的网络名称（你可以使用docker检查）否则它将自动选择一个（取决于docker如何返回它们）。例如，当通过compose文件部署docker `stack` ，compose定义的网络名将以 `stack` 为前缀。
  NB: 当运行在一个容器中时，Træfik需要网络访问权限以运行 `docker network connect <network> <traefik-container>`

Marathon 后端

Træfik 可以使用Marathon作为后端配置:

```
1.  ################################################################
2.  # Mesos/Marathon 后端配置
3.  ################################################################
4.
5.  # 启用 Marathon 后端配置
6.  #
7.  # Optional
8.  #
9.  [marathon]
10.
11. # Marathon server endpoint.
12. # You can also specify multiple endpoint for Marathon:
13. # endpoint := "http://10.241.1.71:8080,10.241.1.72:8080,10.241.1.73:8080"
14. #
15. # Required
16. #
17. endpoint = "http://127.0.0.1:8080"
18.
19. # Enable watch Marathon changes
20. #
21. # Optional
22. #
23. watch = true
24.
25. # Default domain used.
26. #
27. # Required
28. #
29. domain = "marathon.localhost"
30.
31. # Override default configuration template. For advanced users :)
32. #
33. # Optional
34. #
35. # filename = "marathon.tmpl"
36.
37. # Expose Marathon apps by default in traefik
```

```
38. #
39. # Optional
40. # Default: true
41. #
42. # exposedByDefault = true
43.
44. # Convert Marathon groups to subdomains
45. # Default behavior: /foo/bar/myapp => foo-bar-myapp.{defaultDomain}
46. # with groupsAsSubDomains enabled: /foo/bar/myapp => myapp.bar.foo.
    {defaultDomain}
47. #
48. # Optional
49. # Default: false
50. #
51. # groupsAsSubDomains = true
52.
53. # Enable compatibility with marathon-lb labels
54. #
55. # Optional
56. # Default: false
57. #
58. # marathonLBCompatibility = true
59.
60. # Enable Marathon basic authentication
61. #
62. # Optional
63. #
64. #  [marathon.basic]
65. #  httpBasicAuthUser = "foo"
66. #  httpBasicPassword = "bar"
67.
68. # TLS client configuration. https://golang.org/pkg/crypto/tls/#Config
69. #
70. # Optional
71. #
72. # [marathon.TLS]
73. # CA = "/etc/ssl/ca.crt"
74. # Cert = "/etc/ssl/marathon.cert"
75. # Key = "/etc/ssl/marathon.key"
76. # InsecureSkipVerify = true
77.
78. # DCOSToken for DCOS environment, This will override the Authorization header
```

```
#
# Optional
#
# dcosToken = "xxxxxx"

# Override DialerTimeout
# Amount of time in seconds to allow the Marathon provider to wait to open a TCP
# connection to a Marathon master
#
# Optional
# Default: 60
# dialerTimeout = 5

# Set the TCP Keep Alive interval (in seconds) for the Marathon HTTP Client
#
# Optional
# Default: 10
#
# keepAlive = 10
```

Labels can be used on containers to override default behaviour:

- `traefik.backend=foo`: assign the application to `foo` backend
- `traefik.backend.maxconn.amount=10`: set a maximum number of connections to the backend. Must be used in conjunction with the below label to take effect.
- `traefik.backend.maxconn.extractorfunc=client.ip`: set the function to be used against the request to determine what to limit maximum connections to the backend by. Must be used in conjunction with the above label to take effect.
- `traefik.backend.loadbalancer.method=drr`: override the default `wrr` load balancer algorithm
- `traefik.backend.loadbalancer.sticky=true`: enable backend sticky sessions
- `traefik.backend.circuitbreaker.expression=NetworkErrorRatio() > 0.5`: create a circuit breaker to be used against the backend
- `traefik.portIndex=1`: register port by index in the application's ports array. Useful when the application exposes multiple ports.
- `traefik.port=80`: register the explicit application port value. Cannot be used alongside `traefik.portIndex`.
- `traefik.protocol=https`: override the default `http` protocol

- `traefik.weight=10`: assign this weight to the application
- `traefik.enable=false`: disable this application in Træfık
- `traefik.frontend.rule=Host:test.traefik.io`: override the default frontend rule (Default: `Host:{containerName}.{domain}`).
- `traefik.frontend.passHostHeader=true`: forward client `Host` header to the backend.
- `traefik.frontend.priority=10`: override default frontend priority
- `traefik.frontend.entryPoints=http,https`: assign this frontend to entry points `http` and `https`. Overrides `defaultEntryPoints`.

Mesos generic backend

Træfik can be configured to use Mesos as a backend configuration:

```
###################################################################
# Mesos configuration backend
###################################################################

# Enable Mesos configuration backend
#
# Optional
#
[mesos]

# Mesos server endpoint.
# You can also specify multiple endpoint for Mesos:
# endpoint = "192.168.35.40:5050,192.168.35.41:5050,192.168.35.42:5050"
# endpoint =
"zk://192.168.35.20:2181,192.168.35.21:2181,192.168.35.22:2181/mesos"
#
# Required
#
endpoint = "http://127.0.0.1:8080"

# Enable watch Mesos changes
#
# Optional
#
watch = true

# Default domain used.
# Can be overridden by setting the "traefik.domain" label on an application.
#
# Required
#
domain = "mesos.localhost"

# Override default configuration template. For advanced users :)
#
# Optional
#
```

```
# filename = "mesos.tmpl"

# Expose Mesos apps by default in traefik
#
# Optional
# Default: false
#
# ExposedByDefault = true

# TLS client configuration. https://golang.org/pkg/crypto/tls/#Config
#
# Optional
#
# [mesos.TLS]
# InsecureSkipVerify = true

# Zookeeper timeout (in seconds)
#
# Optional
# Default: 30
#
# ZkDetectionTimeout = 30

# Polling interval (in seconds)
#
# Optional
# Default: 30
#
# RefreshSeconds = 30

# IP sources (e.g. host, docker, mesos, rkt)
#
# Optional
#
# IPSources = "host"

# HTTP Timeout (in seconds)
#
# Optional
# Default: 30
#
# StateTimeoutSecond = "30"
```

Kubernetes Ingress backend

Træfik can be configured to use Kubernetes Ingress as a backend configuration:

```
1.  ################################################################
2.  # Kubernetes Ingress configuration backend
3.  ################################################################
4.  # Enable Kubernetes Ingress configuration backend
5.  #
6.  # Optional
7.  #
8.  [kubernetes]
9.
10. # Kubernetes server endpoint
11. #
12. # When deployed as a replication controller in Kubernetes,
13. # Traefik will use env variable KUBERNETES_SERVICE_HOST
14. # and KUBERNETES_SERVICE_PORT_HTTPS as endpoint
15. # Secure token will be found in
    /var/run/secrets/kubernetes.io/serviceaccount/token
16. # and SSL CA cert in /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
17. #
18. # Optional
19. #
20. # endpoint = "http://localhost:8080"
21. # namespaces = ["default","production"]
22. #
23. # See: http://kubernetes.io/docs/user-guide/labels/#list-and-watch-filtering
24. # labelselector = "A and not B"
25. #
```

Annotations can be used on containers to override default behaviour for the whole Ingress resource:

- `traefik.frontend.rule.type: PathPrefixStrip` : override the default frontend rule type (Default: `PathPrefix`).
  Annotations can be used on the Kubernetes service to override default behaviour:
- `traefik.backend.loadbalancer.method=drr` : override the default `wrr` load

balancer algorithm

- `traefik.backend.loadbalancer.sticky=true`: enable backend sticky sessions
  You can find here an example ingress and replication controller.

Additionally, an annotation can be used on Kubernetes services to set the circuit breaker expression for a backend.

- `traefik.backend.circuitbreaker: <expression>`: set the circuit breaker expression for the backend (Default: nil).

Consul backend

Træfik can be configured to use Consul as a backend configuration:

```
################################################################
# Consul KV configuration backend
################################################################

# Enable Consul KV configuration backend
#
# Optional
#
[consul]

# Consul server endpoint
#
# Required
#
endpoint = "127.0.0.1:8500"

# Enable watch Consul changes
#
# Optional
#
watch = true

# Prefix used for KV store.
#
# Optional
#
prefix = "traefik"

# Override default configuration template. For advanced users :)
#
# Optional
#
# filename = "consul.tmpl"

# Enable consul TLS connection
#
# Optional
```

```
38. #
39. # [consul.tls]
40. # ca = "/etc/ssl/ca.crt"
41. # cert = "/etc/ssl/consul.crt"
42. # key = "/etc/ssl/consul.key"
43. # insecureskipverify = true
```

Please refer to the Key Value storage structure section to get documentation on traefik KV structure.

Consul catalog backend

Træfik can be configured to use service discovery catalog of Consul as a backend configuration:

```
1.  ################################################################
2.  # Consul Catalog configuration backend
3.  ################################################################
4.
5.  # Enable Consul Catalog configuration backend
6.  #
7.  # Optional
8.  #
9.  [consulCatalog]
10.
11. # Consul server endpoint
12. #
13. # Required
14. #
15. endpoint = "127.0.0.1:8500"
16.
17. # Default domain used.
18. #
19. # Optional
20. #
21. domain = "consul.localhost"
22.
23. # Prefix for Consul catalog tags
24. #
25. # Optional
26. #
27. prefix = "traefik"
```

This backend will create routes matching on hostname based on the service name used in consul.

Additional settings can be defined using Consul Catalog tags:

- `traefik.enable=false` : disable this container in Træfik
- `traefik.protocol=https` : override the default `http` protocol
- `traefik.backend.weight=10` : assign this weight to the container

- `traefik.backend.circuitbreaker=NetworkErrorRatio() > 0.5`
- `traefik.backend.loadbalancer=drr`: override the default load balancing mode
- `traefik.backend.maxconn.amount=10`: set a maximum number of connections to the backend. Must be used in conjunction with the below label to take effect.
- `traefik.backend.maxconn.extractorfunc=client.ip`: set the function to be used against the request to determine what to limit maximum connections to the backend by. Must be used in conjunction with the above label to take effect.
- `traefik.frontend.rule=Host:test.traefik.io`: override the default frontend rule (Default: `Host:{containerName}.{domain}`).
- `traefik.frontend.passHostHeader=true`: forward client `Host` header to the backend.
- `traefik.frontend.priority=10`: override default frontend priority
- `traefik.frontend.entryPoints=http,https`: assign this frontend to entry points `http` and `https`. Overrides `defaultEntryPoints`.

Etcd backend

Træfık can be configured to use Etcd as a backend configuration:

```
##################################################################
# Etcd configuration backend
##################################################################

# Enable Etcd configuration backend
#
# Optional
#
[etcd]

# Etcd server endpoint
#
# Required
#
endpoint = "127.0.0.1:2379"

# Enable watch Etcd changes
#
# Optional
#
watch = true

# Prefix used for KV store.
#
# Optional
#
prefix = "/traefik"

# Override default configuration template. For advanced users :)
#
# Optional
#
# filename = "etcd.tmpl"

# Enable etcd TLS connection
#
# Optional
```

```
38. #
39. # [etcd.tls]
40. # ca = "/etc/ssl/ca.crt"
41. # cert = "/etc/ssl/etcd.crt"
42. # key = "/etc/ssl/etcd.key"
43. # insecureskipverify = true
```

Please refer to the Key Value storage structure section to get documentation on traefik KV structure.

Zookeeper backend

Træfik can be configured to use Zookeeper as a backend configuration:

```
1.  ##################################################################
2.  # Zookeeper configuration backend
3.  ##################################################################
4.
5.  # Enable Zookeeperconfiguration backend
6.  #
7.  # Optional
8.  #
9.  [zookeeper]
10.
11. # Zookeeper server endpoint
12. #
13. # Required
14. #
15. endpoint = "127.0.0.1:2181"
16.
17. # Enable watch Zookeeper changes
18. #
19. # Optional
20. #
21. watch = true
22.
23. # Prefix used for KV store.
24. #
25. # Optional
26. #
27. prefix = "traefik"
28.
29. # Override default configuration template. For advanced users :)
30. #
31. # Optional
32. #
33. # filename = "zookeeper.tmpl"
```

Please refer to the Key Value storage structure section to get documentation on traefik KV structure.

BoltDB backend

Træfık can be configured to use BoltDB as a backend configuration:

```
################################################################
# BoltDB configuration backend
################################################################

# Enable BoltDB configuration backend
#
# Optional
#
[boltdb]

# BoltDB file
#
# Required
#
endpoint = "/my.db"

# Enable watch BoltDB changes
#
# Optional
#
watch = true

# Prefix used for KV store.
#
# Optional
#
prefix = "/traefik"

# Override default configuration template. For advanced users :)
#
# Optional
#
# filename = "boltdb.tmpl"
```

Eureka backend

Træfik can be configured to use Eureka as a backend configuration:

```
##################################################################
# Eureka configuration backend
##################################################################

# Enable Eureka configuration backend
#
# Optional
#
[eureka]

# Eureka server endpoint.
# endpoint := "http://my.eureka.server/eureka"
#
# Required
#
endpoint = "http://my.eureka.server/eureka"

# Override default configuration time between refresh
#
# Optional
# default 30s
delay = "1m"

# Override default configuration template. For advanced users :)
#
# Optional
#
# filename = "eureka.tmpl"
```

Please refer to the Key Value storage structure section to get documentation on traefik KV structure.

ECS backend

Træfık can be configured to use Amazon ECS as a backend configuration:

```
################################################################
# ECS configuration backend
################################################################

# Enable ECS configuration backend
#
# Optional
#
[ecs]

# ECS Cluster Name
#
# Optional
# Default: "default"
#
Cluster = "default"

# Enable watch ECS changes
#
# Optional
# Default: true
#
Watch = true

# Polling interval (in seconds)
#
# Optional
# Default: 15
#
RefreshSeconds = 15

# Expose ECS services by default in traefik
#
# Optional
# Default: true
#
ExposedByDefault = false
```

```
38.
39.  # Region to use when connecting to AWS
40.  #
41.  # Optional
42.  #
43.  # Region = "us-east-1"
44.
45.  # AccessKeyID to use when connecting to AWS
46.  #
47.  # Optional
48.  #
49.  # AccessKeyID = "abc"
50.
51.  # SecretAccessKey to use when connecting to AWS
52.  #
53.  # Optional
54.  #
55.  # SecretAccessKey = "123"
56.
```

Labels can be used on task containers to override default behaviour:

- `traefik.protocol=https` : override the default `http` protocol
- `traefik.weight=10` : assign this weight to the container
- `traefik.enable=false` : disable this container in Træfik
- `traefik.frontend.rule=Host:test.traefik.io` : override the default frontend rule (Default: `Host:{containerName}.{domain}`).
- `traefik.frontend.passHostHeader=true` : forward client `Host` header to the backend.
- `traefik.frontend.priority=10` : override default frontend priority
- `traefik.frontend.entryPoints=http,https` : assign this frontend to entry points `http` and `https` . Overrides `defaultEntryPoints` .
  If `AccessKeyID` / `SecretAccessKey` is not given credentials will be resolved in the following order:
- From environment variables; `AWS_ACCESS_KEY_ID` , `AWS_SECRET_ACCESS_KEY` , and `AWS_SESSION_TOKEN` .
- Shared credentials, determined by `AWS_PROFILE` and `AWS_SHARED_CREDENTIALS_FILE` , defaults to `default` and `~/.aws/credentials` .
- EC2 instance role or ECS task role

Træfık needs the following policy to read ECS information:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Traefik ECS read access",
            "Effect": "Allow",
            "Action": [
                "ecs:ListTasks",
                "ecs:DescribeTasks",
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTaskDefinition",
                "ec2:DescribeInstances"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Rancher backend

Træfik can be configured to use Rancher as a backend configuration:

```
################################################################
# Rancher configuration backend
################################################################

# Enable Rancher configuration backend
#
# Optional
#
[rancher]

# Default domain used.
# Can be overridden by setting the "traefik.domain" label on an service.
#
# Required
#
domain = "rancher.localhost"

# Enable watch Rancher changes
#
# Optional
# Default: true
#
Watch = true

# Expose Rancher services by default in traefik
#
# Optional
# Default: true
#
ExposedByDefault = false

# Endpoint to use when connecting to Rancher
#
# Optional
# Endpoint = "http://rancherserver.example.com"

# AccessKey to use when connecting to Rancher
```

```
38. #
39. # Optional
40. # AccessKey = "XXXXXXXXX"
41.
42. # SecretKey to use when connecting to Rancher
43. #
44. # Optional
45. # SecretKey = "XXXXXXXXXXX"
46.
```

If you're deploying traefik as a service within rancher, you can alternatively set these labels on the service to let it only fetch data of its current environment. The settings `endpoint`, `accesskey` and `secretkey` can be omitted then.

- `io.rancher.container.create_agent=true`
- `io.rancher.container.agent.role=environment`
  Labels can be used on task containers to override default behaviour:
- `traefik.protocol=https` : override the default `http` protocol
- `traefik.weight=10` : assign this weight to the container
- `traefik.enable=false` : disable this container in Træfık
- `traefik.frontend.rule=Host:test.traefik.io` : override the default frontend rule (Default: `Host:{containerName}.{domain}`).
- `traefik.frontend.passHostHeader=true` : forward client `Host` header to the backend.
- `traefik.frontend.priority=10` : override default frontend priority
- `traefik.frontend.entryPoints=http,https` : assign this frontend to entry points `http` and `https` . Overrides `defaultEntryPoints` .

- 配置样本
- Swarm 集群
- Swarm mode 集群
- Kubernetes
- Key-value 存储配置
- Clustering/HA

Examples

You will find here some configuration examples of Træfik.

HTTP only

```
defaultEntryPoints = ["http"]
[entryPoints]
  [entryPoints.http]
  address = ":80"
```

HTTP + HTTPS (with SNI)

```
defaultEntryPoints = ["http", "https"]
[entryPoints]
  [entryPoints.http]
  address = ":80"
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
      [[entryPoints.https.tls.certificates]]
      CertFile = "integration/fixtures/https/snitest.com.cert"
      KeyFile = "integration/fixtures/https/snitest.com.key"
      [[entryPoints.https.tls.certificates]]
      CertFile = "integration/fixtures/https/snitest.org.cert"
      KeyFile = "integration/fixtures/https/snitest.org.key"
```

Note that we can either give path to certificate file or directly the file content itself (like in this TOML example).

HTTP redirect on HTTPS

```
defaultEntryPoints = ["http", "https"]
[entryPoints]
  [entryPoints.http]
  address = ":80"
    [entryPoints.http.redirect]
```

```
    entryPoint = "https"
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
      [[entryPoints.https.tls.certificates]]
      certFile = "tests/traefik.crt"
      keyFile = "tests/traefik.key"
```

Let's Encrypt support

```
[entryPoints]
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
      # certs used as default certs
      [[entryPoints.https.tls.certificates]]
      certFile = "tests/traefik.crt"
      keyFile = "tests/traefik.key"
[acme]
email = "[email protected]"
storageFile = "acme.json"
onDemand = true
caServer = "http://172.18.0.1:4000/directory"
entryPoint = "https"

[[acme.domains]]
  main = "local1.com"
  sans = ["test1.local1.com", "test2.local1.com"]
[[acme.domains]]
  main = "local2.com"
  sans = ["test1.local2.com", "test2x.local2.com"]
[[acme.domains]]
  main = "local3.com"
[[acme.domains]]
  main = "local4.com"
```

Override entrypoints in frontends

```
[frontends]
  [frontends.frontend1]
```

```
  backend = "backend2"
    [frontends.frontend1.routes.test_1]
    rule = "Host:test.localhost"
  [frontends.frontend2]
  backend = "backend1"
  passHostHeader = true
  entrypoints = ["https"] # overrides defaultEntryPoints
    [frontends.frontend2.routes.test_1]
    rule = "Host:{subdomain:[a-z]+}.localhost"
  [frontends.frontend3]
  entrypoints = ["http", "https"] # overrides defaultEntryPoints
  backend = "backend2"
    rule = "Path:/test"
```

Enable Basic authentication in an entrypoint

With two user/pass:

- `test` : `test`
- `test2` : `test2`
  Passwords are encoded in MD5: you can use htpasswd to generate those ones.

```
defaultEntryPoints = ["http"]
[entryPoints]
  [entryPoints.http]
  address = ":80"
  [entryPoints.http.auth.basic]
  users = ["test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/",
"test2:$apr1$d9hr9HBB$4HxwgUir3HP4EsggP/QNo0"]
```

Pass Authenticated user to application via headers

Providing an authentication method as described above, it is possible to pass the user to the application via a configurable header value

```
defaultEntryPoints = ["http"]
[entryPoints]
```

```
3.    [entryPoints.http]
4.    address = ":80"
5.    [entryPoints.http.auth]
6.      headerField = "X-WebAuth-User"
7.      [entryPoints.http.auth.basic]
8.      users = ["test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/",
   "test2:$apr1$d9hr9HBB$4HxwgUir3HP4EsggP/QNo0"]
```

Swarm cluster

This section explains how to create a multi-host swarm cluster using docker-machine and how to deploy Træfik on it. The cluster consists of:

- 2 servers
- 1 swarm master
- 2 swarm nodes
- 1 overlay network (multi-host networking)

Prerequisites

- You need to install docker-machine
- You need the latest VirtualBox

Cluster provisioning

We first follow this guide to create the cluster.

Create machine mh-keystore

This machine is the service registry of our cluster.

```
1.  docker-machine create -d virtualbox mh-keystore
```

Then we install the service registry Consul on this machine:

```
1.  eval "$(docker-machine env mh-keystore)"
2.  docker run -d \
3.      -p "8500:8500" \
4.      -h "consul" \
5.      progrium/consul -server -bootstrap
```

Create machine mhs-demo0

This machine is a swarm master and a swarm agent on it.

```
1.  docker-machine create -d virtualbox \
2.      --swarm --swarm-master \
```

```
    --swarm-discovery="consul://$(docker-machine ip mh-keystore):8500" \
    --engine-opt="cluster-store=consul://$(docker-machine ip mh-keystore):8500"
\
    --engine-opt="cluster-advertise=eth1:2376" \
    mhs-demo0
```

Create machine mhs-demo1

This machine have a swarm agent on it.

```
docker-machine create -d virtualbox \
    --swarm \
    --swarm-discovery="consul://$(docker-machine ip mh-keystore):8500" \
    --engine-opt="cluster-store=consul://$(docker-machine ip mh-keystore):8500"
\
    --engine-opt="cluster-advertise=eth1:2376" \
    mhs-demo1
```

Create the overlay Network

Create the overlay network on the swarm master:

```
eval $(docker-machine env --swarm mhs-demo0)
docker network create --driver overlay --subnet=10.0.9.0/24 my-net
```

Deploy Træfık

Deploy Træfık:

```
docker $(docker-machine config mhs-demo0) run \
    -d \
    -p 80:80 -p 8080:8080 \
    --net=my-net \
    -v /var/lib/boot2docker/:/ssl \
    traefik \
    -l DEBUG \
    -c /dev/null \
    --docker \
    --docker.domain=traefik \
    --docker.endpoint=tcp://$(docker-machine ip mhs-demo0):3376 \
```

```
    --docker.tls \
    --docker.tls.ca=/ssl/ca.pem \
    --docker.tls.cert=/ssl/server.pem \
    --docker.tls.key=/ssl/server-key.pem \
    --docker.tls.insecureSkipVerify \
    --docker.watch \
    --web
```

Let's explain this command:

- `-p 80:80 -p 8080:8080` : we bind ports 80 and 8080
- `—net=my-net` : run the container on the network my-net
- `-v /var/lib/boot2docker/:/ssl` : mount the ssl keys generated by docker-machine
- `-c /dev/null` : empty config file
- `—docker` : enable docker backend
- `—docker.endpoint=tcp://172.18.0.1:3376` : connect to the swarm master using the docker_gwbridge network
- `—docker.tls` : enable TLS using the docker-machine keys
- `—web` : activate the webUI on port 8080

Deploy your apps

We can now deploy our app on the cluster, here whoami, a simple web server in GO, on the network `my-net` :

```
eval $(docker-machine env --swarm mhs-demo0)
docker run -d --name=whoami0 --net=my-net --env="constraint:node==mhs-demo0" emilevauge/whoami
docker run -d --name=whoami1 --net=my-net --env="constraint:node==mhs-demo1" emilevauge/whoami
```

Check that everything is started:

```
docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS
NAMES
ba2c21488299        emilevauge/whoami   "/whoamI"                8 seconds ago
Up 9 seconds        80/tcp
mhs-demo1/whoami1
```

```
4. 8147a7746e7a        emilevauge/whoami   "/whoamI"               19 seconds ago
   Up 20 seconds       80/tcp
   mhs-demo0/whoami0
5. 8fbc39271b4c        traefik             "/traefik -l DEBUG -c"   36 seconds ago
   Up 37 seconds       192.168.99.101:80->80/tcp, 192.168.99.101:8080->8080/tcp
   mhs-demo0/serene_bhabha
```

Access to your apps through Træfik

```
1. curl -H Host:whoami0.traefik http://$(docker-machine ip mhs-demo0)
2. Hostname: 8147a7746e7a
3. IP: 127.0.0.1
4. IP: ::1
5. IP: 10.0.9.3
6. IP: fe80::42:aff:fe00:903
7. IP: 172.18.0.3
8. IP: fe80::42:acff:fe12:3
9. GET / HTTP/1.1
10. Host: 10.0.9.3:80
11. User-Agent: curl/7.35.0
12. Accept: */*
13. Accept-Encoding: gzip
14. X-Forwarded-For: 192.168.99.1
15. X-Forwarded-Host: 10.0.9.3:80
16. X-Forwarded-Proto: http
17. X-Forwarded-Server: 8fbc39271b4c
18.
19. curl -H Host:whoami1.traefik http://$(docker-machine ip mhs-demo0)
20. Hostname: ba2c21488299
21. IP: 127.0.0.1
22. IP: ::1
23. IP: 10.0.9.4
24. IP: fe80::42:aff:fe00:904
25. IP: 172.18.0.2
26. IP: fe80::42:acff:fe12:2
27. GET / HTTP/1.1
28. Host: 10.0.9.4:80
29. User-Agent: curl/7.35.0
30. Accept: */*
31. Accept-Encoding: gzip
32. X-Forwarded-For: 192.168.99.1
```

```
33.  X-Forwarded-Host: 10.0.9.4:80
34.  X-Forwarded-Proto: http
35.  X-Forwarded-Server: 8fbc39271b4c
```

Docker Swarm (mode) cluster

This section explains how to create a multi-host docker cluster with swarm mode using docker-machine and how to deploy Træfɪk on it.

The cluster consists of:

- 3 servers
- 1 manager
- 2 workers
- 1 overlay network (multi-host networking)

Prerequisites

- You will need to install docker-machine
- You will need the latest VirtualBox

Cluster provisioning

First, let's create all the required nodes. It's a shorter version of the swarm tutorial.

```
1. docker-machine create -d virtualbox manager
2. docker-machine create -d virtualbox worker1
3. docker-machine create -d virtualbox worker2
```

Then, let's setup the cluster, in order :

- initialize the cluster
- get the token for other host to join
- on both workers, join the cluster with the token

```
1. docker-machine ssh manager "docker swarm init \
2.     --listen-addr $(docker-machine ip manager) \
3.     --advertise-addr $(docker-machine ip manager)"
4.
5. export worker_token=$(docker-machine ssh manager "docker swarm \
6. join-token worker -q")
7.
8. docker-machine ssh worker1 "docker swarm join \
```

```
    --token=${worker_token} \
    --listen-addr $(docker-machine ip worker1) \
    --advertise-addr $(docker-machine ip worker1) \
    $(docker-machine ip manager)"

docker-machine ssh worker2 "docker swarm join \
    --token=${worker_token} \
    --listen-addr $(docker-machine ip worker2) \
    --advertise-addr $(docker-machine ip worker2) \
    $(docker-machine ip manager)"
```

Let's validate the cluster is up and running.

```
docker-machine ssh manager docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
2a770ov9vixeadep674265u1n    worker1   Ready   Active
dbi3or4q8ii8elbws70g4hkdh *  manager   Ready   Active        Leader
esbhhy6vnqv90xomjaomdgy46    worker2   Ready   Active
```

Finally, let's create a network for Træfik to use.

```
docker-machine ssh manager "docker network create --driver=overlay traefik-net"
```

Deploy Træfik

Let's deploy Træfik as a docker service in our cluster. The only requirement for Træfik to work with swarm mode is that it needs to run on a manager node – we are going to use a constraint for that.

```
docker-machine ssh manager "docker service create \
    --name traefik \
    --constraint=node.role==manager \
    --publish 80:80 --publish 8080:8080 \
    --mount type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock \
    --network traefik-net \
    traefik \
    --docker \
    --docker.swarmmode \
    --docker.domain=traefik \
    --docker.watch \
    --web"
```

Let's explain this command:

- `–publish 80:80 –publish 8080:8080` : we publish port `80` and `8080` on the cluster.
- `–constraint=node.role==manager` : we ask docker to schedule Træfik on a manager node.
- `–mount type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock` : we bind mount the docker socket where Træfik is scheduled to be able to speak to the daemon.
- `–network traefik-net` : we attach the Træfik service (and thus the underlying container) to the `traefik-net` network.
- `–docker` : enable docker backend, and `–docker.swarmmode` to enable the swarm mode on Træfik.
- `–web` : activate the webUI on port 8080

Deploy your apps

We can now deploy our app on the cluster, here whoami, a simple web server in Go. We start 2 services, on the `traefik-net` network.

```
docker-machine ssh manager "docker service create \
    --name whoami0 \
    --label traefik.port=80 \
    --network traefik-net \
    emilevauge/whoami"

docker-machine ssh manager "docker service create \
    --name whoami1 \
    --label traefik.port=80 \
    --network traefik-net \
    --label traefik.backend.loadbalancer.sticky=true \
    emilevauge/whoami"
```

Note that we set whoami1 to use sticky sessions (`–label traefik.backend.loadbalancer.sticky=true`). We'll demonstrate that later. If using `docker stack deploy` , there is a specific way that the labels must be defined in the docker-compose file.

Check that everything is scheduled and started:

```
docker-machine ssh manager "docker service ls"
ID            NAME     REPLICAS  IMAGE              COMMAND
ab046gpaqtln  whoami0  1/1       emilevauge/whoami
cgfg5ifzrpgm  whoami1  1/1       emilevauge/whoami
dtpl249tfghc  traefik  1/1       traefik            --docker --docker.swarmmode
--docker.domain=traefik --docker.watch --web
```

Access to your apps through Træfik

```
curl -H Host:whoami0.traefik http://$(docker-machine ip manager)
Hostname: 8147a7746e7a
IP: 127.0.0.1
IP: ::1
IP: 10.0.9.3
IP: fe80::42:aff:fe00:903
IP: 172.18.0.3
IP: fe80::42:acff:fe12:3
GET / HTTP/1.1
Host: 10.0.9.3:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 192.168.99.1
X-Forwarded-Host: 10.0.9.3:80
X-Forwarded-Proto: http
X-Forwarded-Server: 8fbc39271b4c

curl -H Host:whoami1.traefik http://$(docker-machine ip manager)
Hostname: ba2c21488299
IP: 127.0.0.1
IP: ::1
IP: 10.0.9.4
IP: fe80::42:aff:fe00:904
IP: 172.18.0.2
IP: fe80::42:acff:fe12:2
GET / HTTP/1.1
Host: 10.0.9.4:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 192.168.99.1
```

```
X-Forwarded-Host: 10.0.9.4:80
X-Forwarded-Proto: http
X-Forwarded-Server: 8fbc39271b4c
```

Note that as Træfik is published, you can access it from any machine and not only the manager.

```
curl -H Host:whoami0.traefik http://$(docker-machine ip worker1)
Hostname: 8147a7746e7a
IP: 127.0.0.1
IP: ::1
IP: 10.0.9.3
IP: fe80::42:aff:fe00:903
IP: 172.18.0.3
IP: fe80::42:acff:fe12:3
GET / HTTP/1.1
Host: 10.0.9.3:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 192.168.99.1
X-Forwarded-Host: 10.0.9.3:80
X-Forwarded-Proto: http
X-Forwarded-Server: 8fbc39271b4c

curl -H Host:whoami1.traefik http://$(docker-machine ip worker2)
Hostname: ba2c21488299
IP: 127.0.0.1
IP: ::1
IP: 10.0.9.4
IP: fe80::42:aff:fe00:904
IP: 172.18.0.2
IP: fe80::42:acff:fe12:2
GET / HTTP/1.1
Host: 10.0.9.4:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 192.168.99.1
X-Forwarded-Host: 10.0.9.4:80
X-Forwarded-Proto: http
X-Forwarded-Server: 8fbc39271b4c
```

Scale both services

```
docker-machine ssh manager "docker service scale whoami0=5"

docker-machine ssh manager "docker service scale whoami1=5"
```

Check that we now have 5 replicas of each `whoami` service:

```
docker-machine ssh manager "docker service ls"
ID            NAME     REPLICAS  IMAGE              COMMAND
ab046gpaqtln  whoami0  5/5       emilevauge/whoami
cgfg5ifzrpgm  whoami1  5/5       emilevauge/whoami
dtpl249tfghc  traefik  1/1       traefik            --docker --docker.swarmmode
--docker.domain=traefik --docker.watch --web
```

Access to your whoami0 through Træfık multiple times.

Repeat the following command multiple times and note that the Hostname changes each time as Traefik load balances each request against the 5 tasks.

```
curl -H Host:whoami0.traefik http://$(docker-machine ip manager)
Hostname: 8147a7746e7a
IP: 127.0.0.1
IP: ::1
IP: 10.0.9.3
IP: fe80::42:aff:fe00:903
IP: 172.18.0.3
IP: fe80::42:acff:fe12:3
GET / HTTP/1.1
Host: 10.0.9.3:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 192.168.99.1
X-Forwarded-Host: 10.0.9.3:80
X-Forwarded-Proto: http
X-Forwarded-Server: 8fbc39271b4c
```

Do the same against whoami1.

```
curl -H Host:whoami1.traefik http://$(docker-machine ip manager)
Hostname: ba2c21488299
IP: 127.0.0.1
IP: ::1
IP: 10.0.9.4
IP: fe80::42:aff:fe00:904
IP: 172.18.0.2
IP: fe80::42:acff:fe12:2
GET / HTTP/1.1
Host: 10.0.9.4:80
User-Agent: curl/7.35.0
Accept: */*
Accept-Encoding: gzip
X-Forwarded-For: 192.168.99.1
X-Forwarded-Host: 10.0.9.4:80
X-Forwarded-Proto: http
X-Forwarded-Server: 8fbc39271b4c
```

Wait, I thought we added the sticky flag to whoami1? Traefik relies on a cookie to maintain stickyness so you'll need to test this with a browser.

First you need to add whoami1.traefik to your hosts file:

```
if [ -n "$(grep whoami1.traefik /etc/hosts)" ];
then
echo "whoami1.traefik already exists (make sure the ip is current)";
else
sudo -- sh -c -e "echo '$(docker-machine ip manager)\twhoami1.traefik'
>> /etc/hosts";
fi
```

Now open your browser and go to http://whoami1.traefik/

You will now see that stickyness is maintained.

Kubernetes Ingress Controller

This guide explains how to use Træfik as an Ingress controller in a Kubernetes cluster. If you are not familiar with Ingresses in Kubernetes you might want to read the Kubernetes user guide

The config files used in this guide can be found in the examples directory

Prerequisites

- A working Kubernetes cluster. If you want to follow along with this guide, you should setup minikube on your machine, as it is the quickest way to get a local Kubernetes cluster setup for experimentation and development.
- The `kubectl` binary should be installed on your workstation.

Deploy Træfik

We are going to deploy Træfik with a Deployment, as this will allow you to easily roll out config changes or update the image.

```
1.  kind: Deployment
2.  apiVersion: extensions/v1beta1
3.  metadata:
4.    name: traefik-ingress-controller
5.    namespace: kube-system
6.    labels:
7.      k8s-app: traefik-ingress-lb
8.  spec:
9.    replicas: 1
10.   selector:
11.     matchLabels:
12.       k8s-app: traefik-ingress-lb
13.   template:
14.     metadata:
15.       labels:
16.         k8s-app: traefik-ingress-lb
17.         name: traefik-ingress-lb
18.     spec:
```

```
      terminationGracePeriodSeconds: 60
      containers:
      - image: traefik
        name: traefik-ingress-lb
        resources:
          limits:
            cpu: 200m
            memory: 30Mi
          requests:
            cpu: 100m
            memory: 20Mi
        ports:
        - containerPort: 80
          hostPort: 80
        - containerPort: 8080
        args:
        - --web
        - --kubernetes
```

examples/k8s/traefik.yaml

> notice that we binding port 80 on the Træfık container to port 80 on the host. With a multi node cluster we might expose Træfık with a NodePort or LoadBalancer service and run more than 1 replica of Træfık for high availability.

To deploy Træfık to your cluster start by submitting the deployment to the cluster with `kubectl`:

```
kubectl apply -f examples/k8s/traefik.yaml
```

Check the deployment

Now lets check if our deployment was successful.

Start by listing the pods in the `kube-system` namespace:

```
$kubectl --namespace=kube-system get pods

NAME                                    READY     STATUS    RESTARTS   AGE
kube-addon-manager-minikubevm           1/1       Running   0          4h
kubernetes-dashboard-s8krj              1/1       Running   0          4h
traefik-ingress-controller-678226159-eqseo   1/1       Running   0          7m
```

You should see that after submitting the Deployment to Kubernetes it has launched a pod, and it is now running. *It might take a few moments for kubernetes to pull the Træfık image and start the container.*

> You could also check the deployment with the Kubernetes dashboard, run `minikube dashboard` to open it in your browser, then choose the `kube-system` namespace from the menu at the top right of the screen.

You should now be able to access Træfık on port 80 of your minikube instance.

```
curl $(minikube ip)
404 page not found
```

> We expect to see a 404 response here as we haven't yet given Træfık any configuration.

Submitting An Ingress to the cluster.

Lets start by creating a Service and an Ingress that will expose the Træfık Web UI.

```
apiVersion: v1
kind: Service
metadata:
  name: traefik-web-ui
  namespace: kube-system
spec:
  selector:
    k8s-app: traefik-ingress-lb
  ports:
  - port: 80
    targetPort: 8080
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: traefik-web-ui
  namespace: kube-system
spec:
  rules:
```

```
    - host: traefik-ui.local
      http:
        paths:
        - backend:
            serviceName: traefik-web-ui
            servicePort: 80
```

examples/k8s/ui.yaml

```
kubectl apply -f examples/k8s/ui.yaml
```

Now lets setup an entry in our /etc/hosts file to route `traefik-ui.local` to our cluster.

> In production you would want to set up real dns entries. You can get the ip address of your minikube instance by running `minikube ip`

```
echo "$(minikube ip) traefik-ui.local" | sudo tee -a /etc/hosts
```

We should now be able to visit traefik-ui.local in the browser and view the Træfık Web UI.

Name based routing

In this example we are going to setup websites for 3 of the United Kingdoms best loved cheeses, Cheddar, Stilton and Wensleydale.

First lets start by launching the 3 pods for the cheese websites.

```
---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: stilton
  labels:
    app: cheese
    cheese: stilton
spec:
  replicas: 2
  selector:
    matchLabels:
```

```
      app: cheese
      task: stilton
  template:
    metadata:
      labels:
        app: cheese
        task: stilton
        version: v0.1
    spec:
      containers:
      - name: cheese
        image: errm/cheese:stilton
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
          limits:
            cpu: 100m
            memory: 50Mi
        ports:
        - containerPort: 80
---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: cheddar
  labels:
    app: cheese
    cheese: cheddar
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cheese
      task: cheddar
  template:
    metadata:
      labels:
        app: cheese
        task: cheddar
        version: v0.1
    spec:
```

```
      containers:
      - name: cheese
        image: errm/cheese:cheddar
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
          limits:
            cpu: 100m
            memory: 50Mi
        ports:
        - containerPort: 80
---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: wensleydale
  labels:
    app: cheese
    cheese: wensleydale
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cheese
      task: wensleydale
  template:
    metadata:
      labels:
        app: cheese
        task: wensleydale
        version: v0.0.1
    spec:
      containers:
      - name: cheese
        image: errm/cheese:wensleydale
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
          limits:
            cpu: 100m
```

```
            memory: 50Mi
        ports:
        - containerPort: 80
```

examples/k8s/cheese-deployments.yaml

```
kubectl apply -f examples/k8s/cheese-deployments.yaml
```

Next we need to setup a service for each of the cheese pods.

```
---
apiVersion: v1
kind: Service
metadata:
  name: stilton
spec:
  ports:
  - name: http
    targetPort: 80
    port: 80
  selector:
    app: cheese
    task: stilton
---
apiVersion: v1
kind: Service
metadata:
  name: cheddar
spec:
  ports:
  - name: http
    targetPort: 80
    port: 80
  selector:
    app: cheese
    task: cheddar
---
apiVersion: v1
kind: Service
metadata:
  name: wensleydale
  annotations:
```

```
    traefik.backend.circuitbreaker: "NetworkErrorRatio() > 0.5"
spec:
  ports:
  - name: http
    targetPort: 80
    port: 80
  selector:
    app: cheese
    task: wensleydale
```

> Notice that we also set a circuit breaker expression for one of the backends by setting the `traefik.backend.circuitbreaker` annotation on the service.

examples/k8s/cheese-services.yaml

```
kubectl apply -f examples/k8s/cheese-services.yaml
```

Now we can submit an ingress for the cheese websites.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: cheese
spec:
  rules:
  - host: stilton.local
    http:
      paths:
      - path: /
        backend:
          serviceName: stilton
          servicePort: http
  - host: cheddar.local
    http:
      paths:
      - path: /
        backend:
          serviceName: cheddar
          servicePort: http
  - host: wensleydale.local
    http:
      paths:
```

```
      - path: /
        backend:
          serviceName: wensleydale
          servicePort: http
```

examples/k8s/cheese-ingress.yaml

> Notice that we list each hostname, and add a backend service.

```
kubectl apply -f examples/k8s/cheese-ingress.yaml
```

Now visit the Træfık dashboard and you should see a frontend for each host. Along with a backend listing for each service with a Server set up for each pod.

If you edit your `/etc/hosts` again you should be able to access the cheese websites in your browser.

```
echo "$(minikube ip) stilton.local cheddar.local wensleydale.local" | sudo tee -a /etc/hosts
```

- Stilton
- Cheddar
- Wensleydale

Path based routing

Now lets suppose that our fictional client has decided that while they are super happy about our cheesy web design, when they asked for 3 websites they had not really bargained on having to buy 3 domain names.

No problem, we say, why don't we reconfigure the sites to host all 3 under one domain.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: cheeses
  annotations:
    traefik.frontend.rule.type: pathprefixstrip
spec:
```

```
  rules:
  - host: cheeses.local
    http:
      paths:
      - path: /stilton
        backend:
          serviceName: stilton
          servicePort: http
      - path: /cheddar
        backend:
          serviceName: cheddar
          servicePort: http
      - path: /wensleydale
        backend:
          serviceName: wensleydale
          servicePort: http
```

examples/k8s/cheeses-ingress.yaml

> Notice that we are configuring Træfık to strip the prefix from the url path with the `traefik.frontend.rule.type` annotation so that we can use the containers from the previous example without modification.

```
kubectl apply -f examples/k8s/cheeses-ingress.yaml
```

```
echo "$(minikube ip) cheeses.local" | sudo tee -a /etc/hosts
```

You should now be able to visit the websites in your browser.

- cheeses.local/stilton
- cheeses.local/cheddar
- cheeses.local/wensleydale

Key-value store configuration

Both static global configuration and dynamic configuration can be sorted in a Key-value store.

This section explains how to launch Træfık using a configuration loaded from a Key-value store.

Træfık supports several Key-value stores:

- Consul
- etcd
- ZooKeeper
- boltdb

Static configuration in Key-value store

We will see the steps to set it up with an easy example. Note that we could do the same with any other Key-value Store.

docker-compose file for Consul

The Træfık global configuration will be getted from a Consul store.

First we have to launch Consul in a container. The docker-compose file allows us to launch Consul and four instances of the trivial app emilevauge/whoamI :

```
consul:
  image: progrium/consul
  command: -server -bootstrap -log-level debug -ui-dir /ui
  ports:
    - "8400:8400"
    - "8500:8500"
    - "8600:53/udp"
  expose:
    - "8300"
    - "8301"
    - "8301/udp"
    - "8302"
```

```
    - "8302/udp"

whoami1:
  image: emilevauge/whoami

whoami2:
  image: emilevauge/whoami

whoami3:
  image: emilevauge/whoami

whoami4:
  image: emilevauge/whoami
```

Upload the configuration in the Key-value store

We should now fill the store with the Træfik global configuration, as we do with a TOML file configuration. To do that, we can send the Key-value pairs via curl commands or via the Web UI.

Hopefully, Træfik allows automation of this process using the `storeconfig` subcommand. Please refer to the store Træfik configuration section to get documentation on it.

Here is the toml configuration we would like to store in the Key-value Store :

```
logLevel = "DEBUG"

defaultEntryPoints = ["http", "https"]

[entryPoints]
  [entryPoints.http]
  address = ":80"
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
      [[entryPoints.https.tls.certificates]]
      CertFile = "integration/fixtures/https/snitest.com.cert"
      KeyFile = "integration/fixtures/https/snitest.com.key"
```

```
      [[entryPoints.https.tls.certificates]]
      CertFile = """-----BEGIN CERTIFICATE-----
                    <cert file content>
                    -----END CERTIFICATE-----"""
      KeyFile = """-----BEGIN CERTIFICATE-----
                    <key file content>
                    -----END CERTIFICATE-----"""


[consul]
  endpoint = "127.0.0.1:8500"
  watch = true
  prefix = "traefik"

[web]
  address = ":8081"
```

And there, the same global configuration in the Key-value Store (using `prefix = "traefik"`):

Key	Value
`/traefik/loglevel`	`DEBUG`
`/traefik/defaultentrypoints/0`	`http`
`/traefik/defaultentrypoints/1`	`https`
`/traefik/entrypoints/http/address`	`:80`
`/traefik/entrypoints/https/address`	`:443`
`/traefik/entrypoints/https/tls/certificates/0/certfile`	`integration/fixtures/https/snitest.com`
`/traefik/entrypoints/https/tls/certificates/0/keyfile`	`integration/fixtures/https/snitest.com`
`/traefik/entrypoints/https/tls/certificates/1/certfile`	`-BEGIN CERTIFICATE-<cert file content>-CERTIFICATE-`
`/traefik/entrypoints/https/tls/certificates/1/keyfile`	`-BEGIN CERTIFICATE-<key file content>-CERTIFICATE-`
`/traefik/consul/endpoint`	`127.0.0.1:8500`
`/traefik/consul/watch`	`true`
`/traefik/consul/prefix`	`traefik`
`/traefik/web/address`	`:8081`

In case you are setting key values manually,: - Remember to specify the indexes (`0` , `1` , `2` , …) under prefixes `/traefik/defaultentrypoints/` and `/traefik/entrypoints/https/tls/certificates/` in order to match the global configuration structure. - Be careful to give the correct IP address and port on the key `/traefik/consul/endpoint` .

Note that we can either give path to certificate file or directly the file content itself.

Launch Træfik

We will now launch Træfik in a container. We use CLI flags to setup the connection between Træfik and Consul. All the rest of the global configuration is stored in Consul.

Here is the docker-compose file :

```
1.  traefik:
2.    image: traefik
3.    command: --consul --consul.endpoint=127.0.0.1:8500
4.    ports:
5.      - "80:80"
6.      - "8080:8080"
```

NB : Be careful to give the correct IP address and port in the flag `–consul.endpoint` .

TLS support

So far, only Consul and etcd support TLS connections. To set it up, we should enable consul security (or etcd security).

Then, we have to provide CA, Cert and Key to Træfik using `consul` flags :

- `–consul.tls`
- `–consul.tls.ca=path/to/the/file`
- `–consul.tls.cert=path/to/the/file`
- `–consul.tls.key=path/to/the/file`

  Or etcd flags :

- `–etcd.tls`
- `–etcd.tls.ca=path/to/the/file`
- `–etcd.tls.cert=path/to/the/file`
- `–etcd.tls.key=path/to/the/file`

  Note that we can either give directly directly the file content itself (instead of the path to certificate) in a TOML file configuration.

Remember the command `traefik —help` to display the updated list of flags.

Dynamic configuration in Key-value store

Following our example, we will provide backends/frontends rules to Træfık.

Note that this section is independent of the way Træfık got its static configuration. It means that the static configuration can either come from the same Key-value store or from any other sources.

Key-value storage structure

Here is the toml configuration we would like to store in the store :

```
[file]

# rules
[backends]
  [backends.backend1]
    [backends.backend1.circuitbreaker]
      expression = "NetworkErrorRatio() > 0.5"
    [backends.backend1.servers.server1]
    url = "http://172.17.0.2:80"
    weight = 10
    [backends.backend1.servers.server2]
    url = "http://172.17.0.3:80"
    weight = 1
  [backends.backend2]
    [backends.backend1.maxconn]
      amount = 10
      extractorfunc = "request.host"
    [backends.backend2.LoadBalancer]
      method = "drr"
    [backends.backend2.servers.server1]
    url = "http://172.17.0.4:80"
    weight = 1
    [backends.backend2.servers.server2]
    url = "http://172.17.0.5:80"
    weight = 2

[frontends]
```

```
  [frontends.frontend1]
  backend = "backend2"
    [frontends.frontend1.routes.test_1]
    rule = "Host:test.localhost"
  [frontends.frontend2]
  backend = "backend1"
  passHostHeader = true
  priority = 10
  entrypoints = ["https"] # overrides defaultEntryPoints
    [frontends.frontend2.routes.test_1]
    rule = "Host:{subdomain:[a-z]+}.localhost"
  [frontends.frontend3]
  entrypoints = ["http", "https"] # overrides defaultEntryPoints
  backend = "backend2"
    rule = "Path:/test"
```

And there, the same dynamic configuration in a KV Store (using `prefix = "traefik"`):

- backend 1

Key	Value
`/traefik/backends/backend1/circuitbreaker/expression`	`NetworkErrorRatio() > 0.5`
`/traefik/backends/backend1/servers/server1/url`	`http://172.17.0.2:80`
`/traefik/backends/backend1/servers/server1/weight`	`10`
`/traefik/backends/backend1/servers/server2/url`	`http://172.17.0.3:80`
`/traefik/backends/backend1/servers/server2/weight`	`1`
`/traefik/backends/backend1/servers/server2/tags`	`api,helloworld`

- backend 2

Key	Value
`/traefik/backends/backend2/maxconn/amount`	`10`
`/traefik/backends/backend2/maxconn/extractorfunc`	`request.host`
`/traefik/backends/backend2/loadbalancer/method`	`drr`
`/traefik/backends/backend2/servers/server1/url`	`http://172.17.0.4:80`
`/traefik/backends/backend2/servers/server1/weight`	`1`
`/traefik/backends/backend2/servers/server2/url`	`http://172.17.0.5:80`
`/traefik/backends/backend2/servers/server2/weight`	`2`
`/traefik/backends/backend2/servers/server2/tags`	`web`

- frontend 1

Key	Value
`/traefik/frontends/frontend1/backend`	`backend2`
`/traefik/frontends/frontend1/routes/test_1/rule`	`Host:test.localhost`

- frontend 2

Key	Value
`/traefik/frontends/frontend2/backend`	`backend1`
`/traefik/frontends/frontend2/passHostHeader`	`true`
`/traefik/frontends/frontend2/priority`	`10`
`/traefik/frontends/frontend2/entrypoints`	`http,https`
`/traefik/frontends/frontend2/routes/test_2/rule`	`PathPrefix:/test`

Atomic configuration changes

Træfık can watch the backends/frontends configuration changes and generate its configuration automatically.

Note that only backends/frontends rules are dynamic, the rest of the Træfık configuration stay static.

The Etcd and Consul backends do not support updating multiple keys atomically. As a result, it may be possible for Træfık to read an intermediate configuration state despite judicious use of the `–providersThrottleDuration` flag. To solve this problem, Træfık supports a special key called `/traefik/alias`. If set, Træfık use the value as an alternative key prefix.

Given the key structure below, Træfık will use the `http://172.17.0.2:80` as its only backend (frontend keys have been omitted for brevity).

Key	Value
`/traefik/alias`	`/traefik_configurations/1`
`/traefik_configurations/1/backends/backend1/servers/server1/url`	`http://172.17.0.2:80`
`/traefik_configurations/1/backends/backend1/servers/server1/weight`	`10`

When an atomic configuration change is required, you may write a new configuration at an alternative prefix. Here, although the `/traefik_configurations/2/…` keys have been set, the old configuration is still active because the `/traefik/alias` key still points to `/traefik_configurations/1`:

Key	Value
`/traefik/alias`	`/traefik_configurations/1`
`/traefik_configurations/1/backends/backend1/servers/server1/url`	`http://172.17.0.2:80`
`/traefik_configurations/1/backends/backend1/servers/server1/weight`	`10`
`/traefik_configurations/2/backends/backend1/servers/server1/url`	`http://172.17.0.2:80`
`/traefik_configurations/2/backends/backend1/servers/server1/weight`	`5`
`/traefik_configurations/2/backends/backend1/servers/server2/url`	`http://172.17.0.3:80`
`/traefik_configurations/2/backends/backend1/servers/server2/weight`	`5`

Once the `/traefik/alias` key is updated, the new `/traefik_configurations/2` configuration becomes active atomically. Here, we have a 50% balance between the `http://172.17.0.3:80` and the `http://172.17.0.4:80` hosts while no traffic is sent to the `172.17.0.2:80` host:

Key	Value
`/traefik/alias`	`/traefik_configurations/2`
`/traefik_configurations/1/backends/backend1/servers/server1/url`	`http://172.17.0.2:80`
`/traefik_configurations/1/backends/backend1/servers/server1/weight`	`10`
`/traefik_configurations/2/backends/backend1/servers/server1/url`	`http://172.17.0.3:80`
`/traefik_configurations/2/backends/backend1/servers/server1/weight`	`5`
`/traefik_configurations/2/backends/backend1/servers/server2/url`	`http://172.17.0.4:80`
`/traefik_configurations/2/backends/backend1/servers/server2/weight`	`5`

Note that Træfik *will not watch for key changes in the* `/traefik_configurations` *prefix*. It will only watch for changes in the `/traefik/alias`. Further, if the `/traefik/alias` key is set, all other configuration with `/traefik/backends` or `/traefik/frontends` prefix are ignored.

Store configuration in Key-value store

Don't forget to setup the connection between Træfik and Key-value store. The static Træfik configuration in a key-value store can be automatically created and updated, using the `storeconfig` subcommand.

```
1.  $ traefik storeconfig [flags] ...
```

This command is here only to automate the process which upload the configuration into the Key-value store. Træfik will not start but the static configuration will be uploaded into the Key-value store. If you

configured ACME (Let's Encrypt), your registration account and your certificates will also be uploaded.

To upload your ACME certificates to the KV store, get your traefik TOML file and add the new `storage` option in the `acme` section:

```
1. [acme]
2. email = "[email protected]"
3. storage = "traefik/acme/account" # the key where to store your certificates in the KV store
4. storageFile = "acme.json" # your old certificates store
```

Call `traefik storeconfig` to upload your config in the KV store. Then remove the line `storageFile = "acme.json"` from your TOML config file. That's it!

Clustering / High Availability

This guide explains how tu use Træfık in high availability mode. In order to deploy and configure multiple Træfık instances, without copying the same configuration file on each instance, we will use a distributed Key-Value store.

Prerequisites

You will need a working KV store cluster.

File configuration to KV store migration

We created a special Træfık command to help configuring your Key Value store from a Træfık TOML configuration file. Please refer to this section to get more details.

Deploy a Træfık cluster

Once your Træfık configuration is uploaded on your KV store, you can start each Træfık instance. A Træfık cluster is based on a master/slave model. When starting, Træfık will elect a master. If this instance fails, another master will be automatically elected.

Benchmarks

Configuration

I would like to thanks vincentbernat from exoscale.ch who kindly provided the infrastructure needed for the benchmarks.

I used 4 VMs for the tests with the following configuration:

- 32 GB RAM
- 8 CPU Cores
- 10 GB SSD
- Ubuntu 14.04 LTS 64-bit

Setup

- One VM used to launch the benchmarking tool wrk
- One VM for traefik (v1.0.0-beta.416) / nginx (v1.4.6)
- Two VMs for 2 backend servers in go whoami
  Each VM has been tuned using the following limits:

```
sysctl -w fs.file-max="9999999"
sysctl -w fs.nr_open="9999999"
sysctl -w net.core.netdev_max_backlog="4096"
sysctl -w net.core.rmem_max="16777216"
sysctl -w net.core.somaxconn="65535"
sysctl -w net.core.wmem_max="16777216"
sysctl -w net.ipv4.ip_local_port_range="1025       65535"
sysctl -w net.ipv4.tcp_fin_timeout="30"
sysctl -w net.ipv4.tcp_keepalive_time="30"
sysctl -w net.ipv4.tcp_max_syn_backlog="20480"
sysctl -w net.ipv4.tcp_max_tw_buckets="400000"
sysctl -w net.ipv4.tcp_no_metrics_save="1"
sysctl -w net.ipv4.tcp_syn_retries="2"
sysctl -w net.ipv4.tcp_synack_retries="2"
sysctl -w net.ipv4.tcp_tw_recycle="1"
sysctl -w net.ipv4.tcp_tw_reuse="1"
sysctl -w vm.min_free_kbytes="65536"
sysctl -w vm.overcommit_memory="1"
```

```
ulimit -n 9999999
```

Nginx

Here is the config Nginx file use `/etc/nginx/nginx.conf` :

```
user www-data;
worker_processes auto;
worker_rlimit_nofile 200000;
pid /var/run/nginx.pid;

events {
    worker_connections 10000;
    use epoll;
    multi_accept on;
}

http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 300;
    keepalive_requests 10000;
    types_hash_max_size 2048;

    open_file_cache max=200000 inactive=300s;
    open_file_cache_valid 300s;
    open_file_cache_min_uses 2;
    open_file_cache_errors on;

    server_tokens off;
    dav_methods off;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    access_log /var/log/nginx/access.log combined;
    error_log /var/log/nginx/error.log warn;

    gzip off;
    gzip_vary off;

```

```
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*.conf;
}
```

Here is the Nginx vhost file used:

```
upstream whoami {
    server IP-whoami1:80;
    server IP-whoami2:80;
    keepalive 300;
}

server {
    listen 8001;
    server_name test.traefik;
    access_log off;
    error_log /dev/null crit;
    if ($host != "test.traefik") {
        return 404;
    }
    location / {
        proxy_pass http://whoami;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    proxy_set_header  X-Forwarded-Host $host;
    }
}
```

Traefik

Here is the `traefik.toml` file used:

```
MaxIdleConnsPerHost = 100000
defaultEntryPoints = ["http"]

[entryPoints]
  [entryPoints.http]
  address = ":8000"

[file]
[backends]
  [backends.backend1]
```

```
    [backends.backend1.servers.server1]
    url = "http://IP-whoami1:80"
    weight = 1
    [backends.backend1.servers.server2]
    url = "http://IP-whoami2:80"
    weight = 1

[frontends]
  [frontends.frontend1]
  backend = "backend1"
    [frontends.frontend1.routes.test_1]
    rule = "Host: test.traefik"
```

Results

whoami:

```
wrk -t20 -c1000 -d60s -H "Host: test.traefik" --latency  http://IP-whoami:80/bench
Running 1m test @ http://IP-whoami:80/bench
  20 threads and 1000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    70.28ms  134.72ms   1.91s    89.94%
    Req/Sec     2.92k   742.42     8.78k    68.80%
  Latency Distribution
     50%   10.63ms
     75%   75.64ms
     90%  205.65ms
     99%  668.28ms
  3476705 requests in 1.00m, 384.61MB read
  Socket errors: connect 0, read 0, write 0, timeout 103
Requests/sec:  57894.35
Transfer/sec:      6.40MB
```

nginx:

```
wrk -t20 -c1000 -d60s -H "Host: test.traefik" --latency  http://IP-nginx:8001/bench
Running 1m test @ http://IP-nginx:8001/bench
  20 threads and 1000 connections
```

```
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   101.25ms  180.09ms   1.99s    89.34%
    Req/Sec     1.69k   567.69     9.39k    72.62%
  Latency Distribution
     50%   15.46ms
     75%  129.11ms
     90%  302.44ms
     99%  846.59ms
  2018427 requests in 1.00m, 298.36MB read
  Socket errors: connect 0, read 0, write 0, timeout 90
Requests/sec:  33591.67
Transfer/sec:      4.97MB
```

traefik:

```
wrk -t20 -c1000 -d60s -H "Host: test.traefik" --latency  http://IP-traefik:8000/bench
Running 1m test @ http://IP-traefik:8000/bench
  20 threads and 1000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    91.72ms  150.43ms   2.00s    90.50%
    Req/Sec     1.43k   266.37     2.97k    69.77%
  Latency Distribution
     50%   19.74ms
     75%  121.98ms
     90%  237.39ms
     99%  687.49ms
  1705073 requests in 1.00m, 188.63MB read
  Socket errors: connect 0, read 0, write 0, timeout 7
Requests/sec:  28392.44
Transfer/sec:      3.14MB
```

Conclusion

Traefik is obviously slower than Nginx, but not so much: Traefik can serve 28392 requests/sec and Nginx 33591 requests/sec which gives a ratio of 85%. Not bad for young project :) !

Some areas of possible improvements:

- Use GO_REUSEPORT listener

- Run a separate server instance per CPU core with `GOMAXPROCS=1` (it appears during benchmarks that there is a lot more context switches with traefik than with nginx)