

# 使用 Traefik 路由到 Web 應用程式的路徑

[blog.sethcorker.com/traefik-routing-for-web-apps](https://blog.sethcorker.com/traefik-routing-for-web-apps)

賽斯考克

2021年6月30日

如果您正在使用處理路由的單個 Web 應用程式，那非常簡單，但是當使用多個 Web 應用程式時會發生什麼？讓它們輕鬆運行並正確設置路由可能會很痛苦。讓我們來看一個用例，我最近一直在嘗試構建一個由多個 Web 應用程式組成的網站，您可能會聽說這被稱為微前端。第一步是確保我的各種 Web 應用程式都可以在同一域中從不同路徑訪問。讓我們看看我們正在嘗試實現什麼，然後我們將使用一些工具來實現它。

## 我希望每個應用都在不同的路徑上

從本質上講，每個應用程式都應該擁有自己的路徑。

- Home 應用程式可以擁有 `/` 並提供單個頁面
- Shop 應用程式可以擁有 `/shop` 任何帶有該前綴的子路徑，例如 `/shop/2031/info`
- 展示櫃可以擁有 `/showcase`，你明白了。假設我們嘗試在本地運行此設置，您可能不會在其自己的端口上運行每個應用程式。不理想。這意味著每個應用程式都必須指定一個不會與其他應用程式衝突的端口。這適用於 3，如果它增長到 10 個不同的應用程式怎麼辦？我們遇到的另一個問題是 URL 每次都會更改，您會從 `localhost:8000` 到 `localhost:8080` 到其他任何地方。我們希望將這些應用程式放在同一個域中，並感受到凝聚力體驗的一部分。這是反向代理可以提供幫助的地方。

如果您想查看存儲庫，它位於我的 [GitHub - Darth-Knoppix/traefik-mfe-deno](https://github.com/Darth-Knoppix/traefik-mfe-deno) 上。這包括 `docker-compose.yml` 文件和 Deno 服務器。

## 什麼是反向代理？

位於我們的網絡應用程式前面並將請求轉發給它們的服務器。這正是我們需要確保在我訪問 `localhost/shop` Shop web 應用程式時響應請求的內容。從用戶的角度來看，這創造了一種凝聚力的體驗，而從開發的角度來看，每個應用程式都可以被隔離並使用我們喜歡的任何技術。

那麼，現在我們知道我們想要一個反向代理，我們有什麼選擇呢？

## Nginx

Nginx 不僅僅是一個反向代理，但就我們的目的而言，它會工作得很好。它靈活、快速，長期以來一直是我完成這項工作的首選工具。我對 Nginx 的最大問題是，為了得到我們想要的東西，我們仍然需要使用唯一的端口來啟動我們的服務器，這樣它們就不會衝突。如果這不是您關心的問題，那麼 Nginx 是一個很好的解決方案。不過這對我來說是個問題，所以我一直在尋找更好的方法和本地開發，Traefik 似乎符合要求。

## 交通

Traefik proxy 是這樣描述自己的：

Traefik 是一種開源邊緣路由器，它使您發布服務成為一種有趣而輕鬆的體驗。它代表您的系統接收請求並找出哪些組件負責處理它們。

為什麼我成功使用它歸結為易用性。該文檔很棒且易於遵循。我最喜歡的功能是與 Docker 的集成，它允許您運行所有 docker 圖像並添加一些標籤進行配置，以便 Traefik 可以找到它們。這解決了我之前描述的端口問題。這個解決方案允許你所有的應用程序在沒有顯式暴露端口的情況下運行，你只需要讓 Traefik 知道它應該將請求轉發到哪個端口以及它應該何時將這些請求發送到特定的應用程序。像 Traefik 這樣簡單的反向代理是微前端的一個很好的解決方案，甚至更容易為單頁 Web 應用程序和 API 設置前端和後端。

---

## 我們如何在本地使用 Docker 設置 Traefik ?

---

### 碼頭工人組成

---

我將使用一個 docker compose 文件來演示如何設置三個 Web 應用程序：Home、Shop 和 Showcase。這是為了簡化示例並專注於 Traefik 和配置。我也將它與多個目錄一起使用，其中每個項目都在不同的技術堆棧上運行，使用不同的端口等，但都使用相同的 docker 網絡，因此它們可以被 Traefik 獲取。

我們的第一服務

設置反向代理比 Nginx 容易得多。這一切都可以在 docker-compose.yml 文件中用幾行來完成。只要 Traefik 在運行，它就會獲取我們添加到 Docker 的新服務。

```
services:
  reverse-proxy:
    image: traefik:v2.4
    command: --api.insecure=true --providers.docker
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    labels:
      - traefik.enable=false
```

### Web 應用程序配置

Web 應用程序本身都將使用相同的 Deno Web 服務器 源，但在配置略有不同的不同容器上運行。主要是 `APP_NAME` 環境變量會傳入到各個應用中，所以我們可以在標題中顯示出來，以區分它們。在真實場景中，每個 Web 應用程序可能會做一些更複雜的事情，而不僅僅是吐出一些基本的 HTML。

Deno Web 服務器容器將如下所示：

```

services:
  home:
    restart: always
    image: denoland/deno:1.11.2
    command: run --allow-net --allow-read --allow-env /app/server.ts
    labels:
      - traefik.http.routers.home.rule=Host(`deno-mfe.localhost`)
      - traefik.http.services.home.loadbalancer.server.port=8080
    environment:
      - APP_NAME=Home
    volumes:
      - ./app:/app

```

Traefik 使用的關鍵配置是標籤。

- `traefik.http.routers.home.rule=Host(`deno-mfe.localhost`)` 定義我們要使用的域，在本例中我選擇了 `deno-mfe.localhost` 這樣導航到它會很容易。
- `traefik.http.services.home.loadbalancer.server.port=8080` 是什麼告訴 Traefik 它應該向端口發送請求，`8080` 因為這就是 Deno 正在監聽的內容。您可以做更多的事情，但這對我們的目的來說已經很好了。為了支持指向每個應用程序的不同路徑，我們可以使用相同的規則，但添加 `PathPrefix` 這樣的  
`traefik.http.routers.shop.rule=Host(`deno-mfe.localhost`) && PathPrefix(`/shop`)` . 如果請求具有相同的主機和前綴，則此標籤告訴 Traefik 匹配請求。 `/shop`

我們將添加剩餘的應用程序，docker 配置現在如下所示：

```
version: "3.8"
```

```
services:
```

```
  reverse-proxy:
```

```
    image: traefik:v2.4
```

```
    command: --api.insecure=true --providers.docker
```

```
    ports:
```

```
      - "80:80"
```

```
      - "8080:8080"
```

```
    volumes:
```

```
      - /var/run/docker.sock:/var/run/docker.sock
```

```
    labels:
```

```
      - traefik.enable=false
```

```
  home:
```

```
    restart: always
```

```
    image: denoland/deno:1.11.2
```

```
    command: run --allow-net --allow-read --allow-env /app/server.ts
```

```
    labels:
```

```
      - traefik.http.routers.home.rule=Host(`deno-mfe.localhost`)
```

```
      - traefik.http.services.home.loadbalancer.server.port=8080
```

```
    environment:
```

```
      - APP_NAME=Home
```

```
    volumes:
```

```
      - ./app:/app
```

```
  shop:
```

```
    restart: always
```

```
    image: denoland/deno:1.11.2
```

```
    command: run --allow-net --allow-read --allow-env /app/server.ts
```

```
    labels:
```

```
      - traefik.http.routers.shop.rule=Host(`deno-mfe.localhost`) &&
```

```
PathPrefix(`/shop`)
```

```
      - traefik.http.services.shop.loadbalancer.server.port=8080
```

```
    environment:
```

```
      - APP_NAME=Shop
```

```
    volumes:
```

```
      - ./app:/app
```

```
  showcase:
```

```
    restart: always
```

```
    image: denoland/deno:1.11.2
```

```
    command: run --allow-net --allow-read --allow-env /app/server.ts
```

```
    labels:
```

```
      - traefik.http.routers.showcase.rule=Host(`deno-mfe.localhost`) &&
```

```
PathPrefix(`/showcase`)
```

```
      - traefik.http.services.showcase.loadbalancer.server.port=8080
```

```
    environment:
```

```
      - APP_NAME=Showcase
```

```
    volumes:
```

```
      - ./app:/app
```

```
    expose:
```

```
      - 8080
```

Deno 網絡服務器

Deno 是 JavaScript 的運行時，與Node.js非常相似，我為這個示例選擇它的原因是所有內容都適合 24 行代碼。Web 服務器並不重要，您可以輕鬆地設置 Flask 和 React 應用程序或任何您喜歡的應用程序。

此示例設置了一個簡單的 HTTP 服務器，它以一些 HTML 進行響應。我們在 `APP_NAME` 測試時使用環境變量來區分應用程序。

```
import { serve } from "https://deno.land/std@0.99.0/http/server.ts"

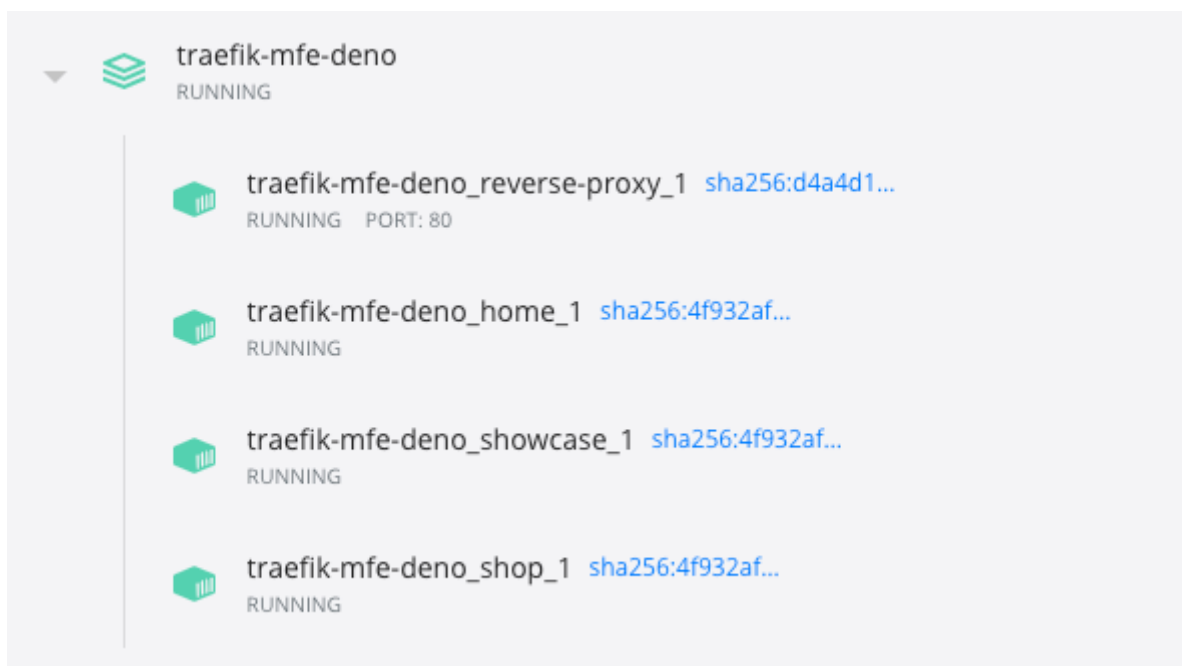
const port = parseInt(`${Deno.env.get("PORT")}`) || 8080
const appName = Deno.env.get("APP_NAME") || "Unknown"

const s = serve({ port })

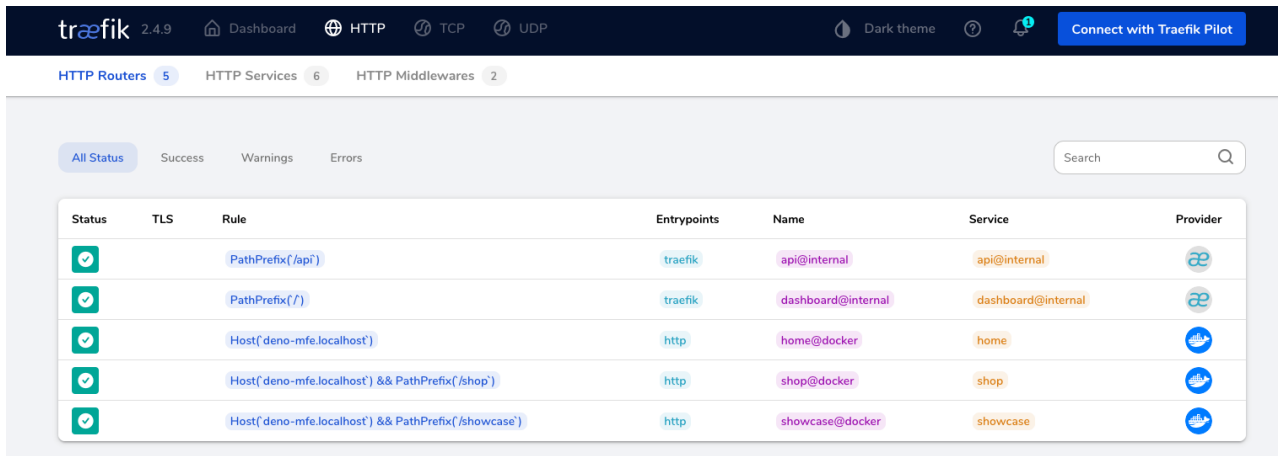
for await (const req of s) {
  const headers = new Headers()
  headers.set("Content-Type", "text/html")
  req.respond({
    body: `
<html>
  <h1>Hello from ${appName}</h1>
  <ul>
    <li><a href="/">Home</a>
    <li><a href="/shop">Shop</a>
    <li><a href="/showcase">Showcase</a>
  </ul>
</html>
`,
    headers,
  })
}
```

## 它是什麼樣子的？

設置完成後，我們可以使用 `docker compose up --build` .



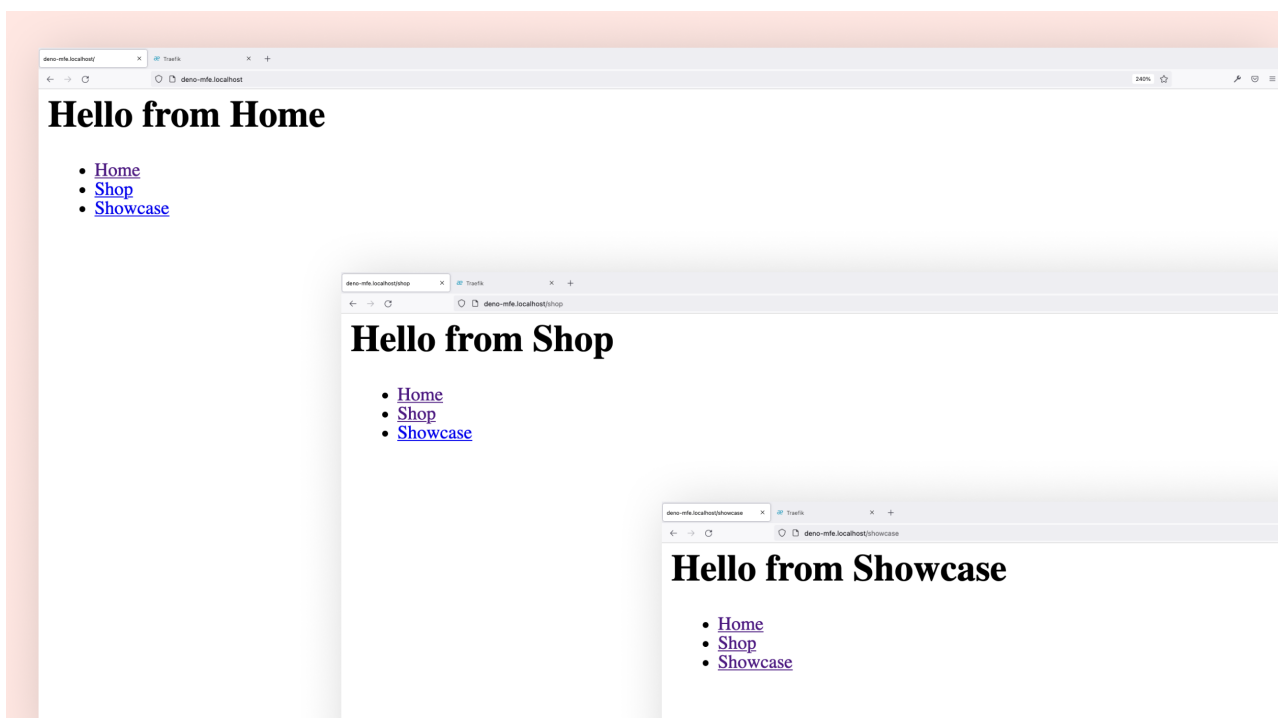
你會注意到我們沒有從 Docker 的角度暴露端口，Traefik 會為我們處理。現在好了，讓我們看一下儀錶盤。我們可以導航到<http://localhost:8080>來查看我們的代理在做什麼。您將看到已設置的配置，如果您啟動任何其他服務（假設它們在同一個 docker 網絡上），Traefik 將選擇它們並開始路由到它們。



The screenshot shows the Traefik dashboard interface. At the top, there's a navigation bar with 'traefik 2.4.9' and tabs for 'Dashboard', 'HTTP', 'TCP', and 'UDP'. Below this, there are sub-tabs for 'HTTP Routers' (5), 'HTTP Services' (6), and 'HTTP Middlewares' (2). The main content area has a filter bar with 'All Status', 'Success', 'Warnings', and 'Errors'. A search bar is on the right. The central table lists the following configurations:

Status	TLS	Rule	Entrypoints	Name	Service	Provider
✔		PathPrefix(`/api`)	traefik	api@internal	api@internal	ae
✔		PathPrefix(`/`)	traefik	dashboard@internal	dashboard@internal	ae
✔		Host(`deno-mfe.localhost`)	http	home@docker	home	df
✔		Host(`deno-mfe.localhost`) && PathPrefix(`/shop`)	http	shop@docker	shop	df
✔		Host(`deno-mfe.localhost`) && PathPrefix(`/showcase`)	http	showcase@docker	showcase	df

最後，我們可以導航到我們指定的主機—— <http://deno-mfe.localhost> 查看我們的 Home 應用程序提供的頁面。分別導航到Shop 應用程序和 Showcase 應用程序 </shop> 並將 </showcase> 請求路由到它們。



好了，這是一種更友好的方式，可以將請求路由到不同的 Web 服務器，而不會弄亂端口。

## 我喜歡 Traefik 和 Docker 的地方

在本地使用多個服務可能很難設置，Docker 和 Docker Compose 在均衡環境和使啟動這些服務更直接方面做得非常出色，但是，當您處理更多服務時；可能很難將所有這些設置得恰到好處並相互交談。當您與其他團隊打交道時，這個問題會加劇，這些團隊創建他們自

己的默認使用相同端口的服務。您最終會為開發的 docker-compose 文件創建覆蓋，並為每個服務仔細分配端口，更新 .env 文件並確保一切按預期工作。Traefik 使這變得更加簡單。

對於多個項目，每個服務只需指定它們擁有的路徑和服務偵聽的端口。然後使用簡單的配置運行 Traefik，它將發現服務。通過外部網絡，團隊可以創建自己的服務，並在本地開發期間使用 Traefik 以更易於推理的方式路由請求。不需要更新引用端口的環境變量，沒有額外的覆蓋，一切都容易得多。它也適用於不同的 url 要求，我已經展示了每個路徑的應用程序方法，但您可以輕鬆地為每個服務提供更友好的子域 url 或按查詢字符串、標題等拆分它。它很容易上手，而且非常可定制。如果您正在處理配置端口和處理許多小型服務，Traefik 可能只會讓推理這些服務變得更容易。搏一搏。