

# Docker container management with Traefik v2 and Portainer

## Introduction

Article level: **Advanced**

I categorise every article based on the complexity. It's a good way to indicate whether you would be interested in reading this article. Click on the link above to see what this means.

## Prerequisites

- Ubuntu 20.04 server
- Docker & Docker-Compose installed
- Domain name

## What are we creating?

In this blog post we will dive into the world of containers. We will set-up a Traefik v2 reverse proxy along with Portainer, using Docker Compose.

This set-up makes container management & deployment a breeze and the reverse proxy allows for running multiple applications on one Docker host. This really brings down the overall overhead that would normally go along with running multiple docker applications, since everything is managed from one point. 🙌

Traefik will route all the incoming traffic to the appropriate docker containers and through the open-source app Portainer you can speed up software deployments, troubleshoot problems and simplify migrations.

As an final example we will deploy a containerized Node.js app into our new environment. Exciting, so let's start! 🚀

## What is Traefik v2?

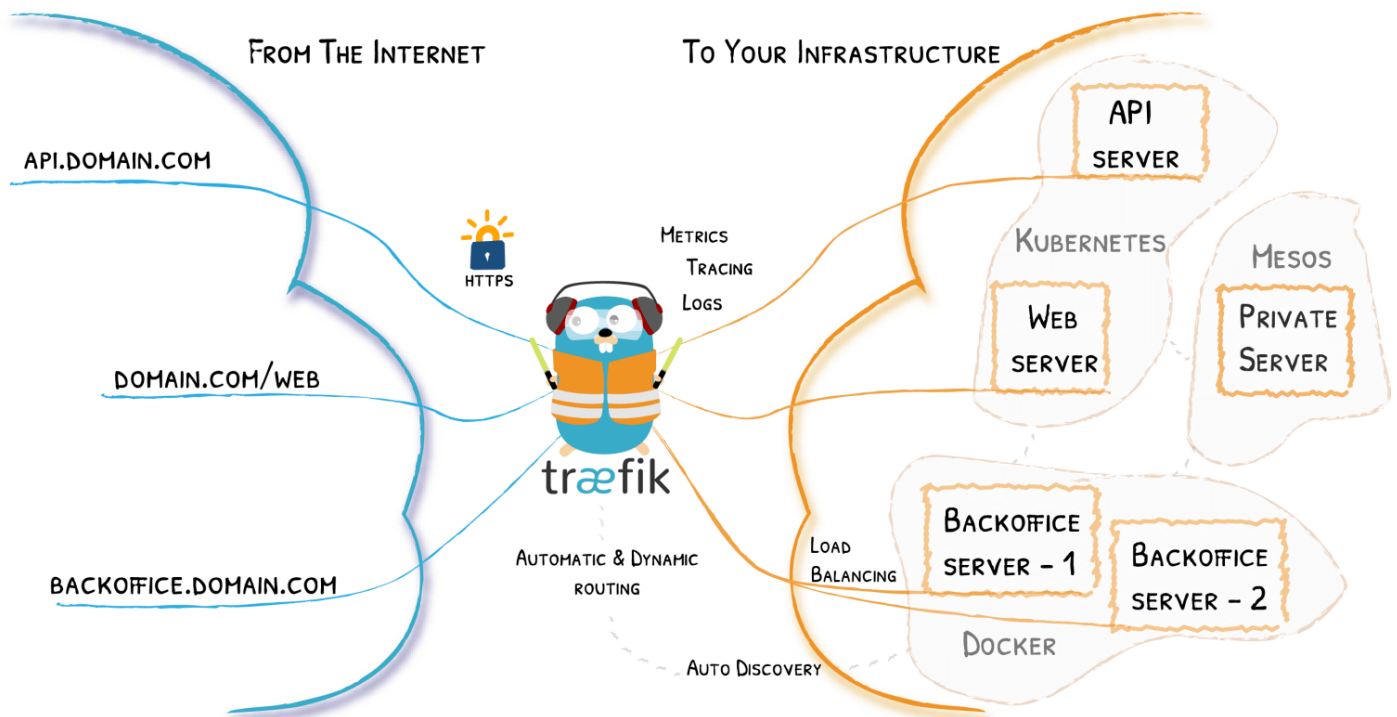
Traefik is a modern and lightweight reverse proxy and load balancer that makes deploying microservices very easy. It is designed to be as simple as possible to operate, but capable of handling large, highly-complex deployments.

It also comes with a powerful set of middlewares that enhance its capabilities to include load balancing, API gateway, orchestrator ingress, as

well as east-west service communication and more. It is written in Go and is packaged as a single binary file and available as a tiny official docker image.

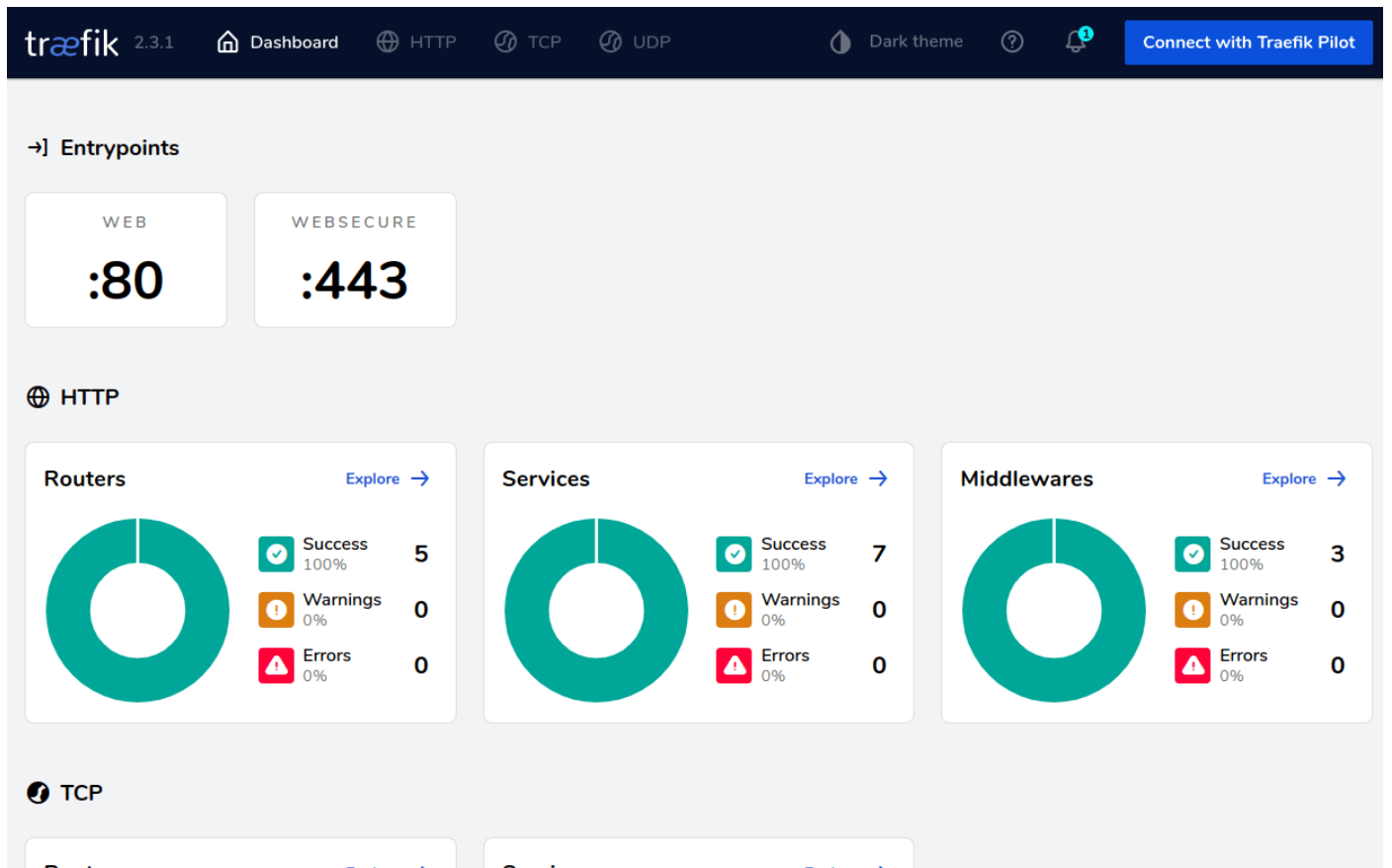
Traditional reverse-proxies require that you configure each route that will connect paths and subdomains to each microservice. In an environment where you add, remove, kill, upgrade, or scale your services many times a day, the task of keeping the routes up to date becomes tedious. 😞

Traefik listens to your service registry/orchestrator API and instantly generates the routes so your microservices are connected to the outside world – without further intervention from your part.




Some of Traefik's features further explained:

- **Dynamic Routing:** Once properly set-up, Traefik will dynamically add new services and containers as they come up to provide traffic routing to them. Let's say you have Traefik running and you want to add a new app, you just build your container and register a new endpoint and Traefik will automatically detect it and start routing traffic to it.
- **Load balancer:** If you have multiple instances of a container, then Traefik can provide load balancing between those instances.
- **Letsencrypt:** When properly configured, Traefik can not only route traffic to a newly discovered service, but also set up free SSL certs from Let's Encrypt. Afterwards it can then redirect all the http traffic to https through middlewares for enhanced security of your application.
- **Web UI:** It comes packed with a very useful management dashboard that helps you visualize all the traffic endpoints, services, middlewares and docker containers while showing potential warnings and errors as well.

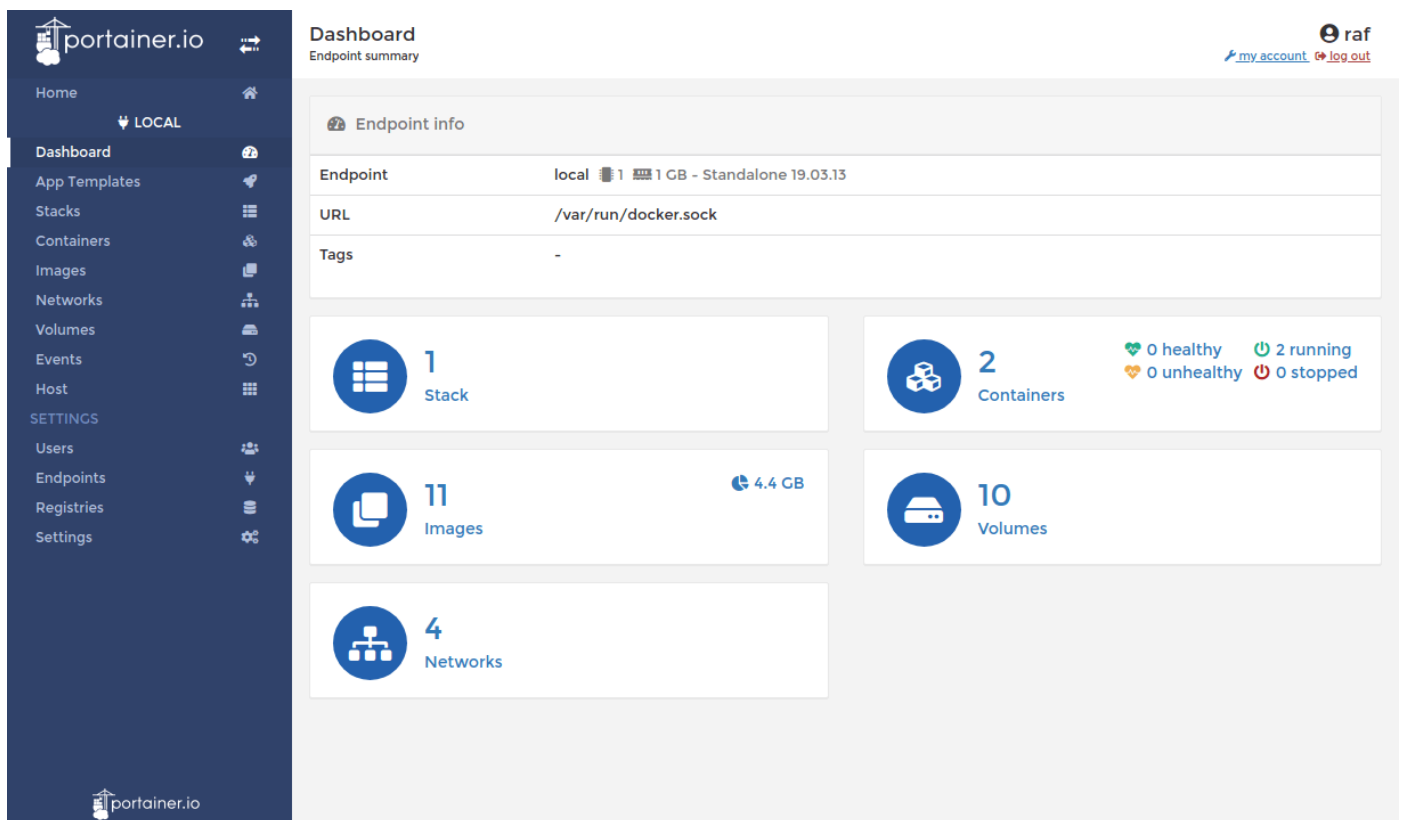


## What is Portainer?

Portainer is a lightweight management UI which allows you to easily manage your Docker host or Swarm cluster.

It is meant to be as simple to deploy as it is to use. It consists of a single container that can run on any Docker engine. It allows you to manage your Docker stacks, containers, images, volumes, networks and more! This will help with speeding up software deployments, troubleshooting problems and simplifying migrations. 

Portainer works by hiding the complexity that makes managing containers hard behind an easy to use GUI. By negating the need for users to use CLI, write YAML or understand manifests, Portainer makes deploying apps and troubleshooting problems so simple, anyone can do it.



## Building our stack 🚧

From this point on I am going to assume you have docker and docker–compose installed on your server and you are running Ubuntu 20.04. I used a Digital Ocean \$5 droplet for this. If you sign up [through this link](#) you can get \$100 worth of credit for free on there!

### I. Setting up DNS records

Alright so the first thing to do is setting up the appropriate domains so we can access our Portainer and Traefik dashboard. Just pick one of the domains you have hoarded over the years 😊

Set them up like this, point to your server:

```
traefik.yourdomain.com
portainer.yourdomain.com
```

In this way our Portainer & Traefik dashboard will be available at the appropriate subdomains.

### II. Creating a user & setting up the directory

Generally you want to avoid using your server as root, so register a user and add them to the sudo group and then switch to that user:

```
$ adduser raf
$ usermod -aG sudo raf
$ su - raf
```

Now it's time to set-up our directory. I already made the whole configuration and published it and you can therefore just clone or fork my repo. I will go over all the files to explain what is going on. So just run:

```
$ git clone https://github.com/rafrasenberg/docker-traefik-portainer ./src
```

Now cd into src and you should be greeted with this tree structure:

```
.
└─ src/
    └─ core/
        └─ traefik-data/
            └─ configurations/
                └─ dynamic.yml
            └─ traefik.yml
            └─ acme.json
        └─ docker-compose.yml
    └─ apps/
```

## File explanation

### I. traefik.yml

The first file we will go over is the traefik.yml file as seen in the code snippet below. This is the static, base configuration of Traefik.

First we tell Traefik that we want the Web GUI by setting dashboard:true

After that we define our two entrypoints web (http) and websecure (https). For our secure https endpoint we set-up the certResolver so we can enjoy automatic certificates from Let's Encrypt! 😊 Next up we load the appropriate middleware so that all our traffic will be forwarded to https.

In the providers part we specify that this file will be passed to a docker container using bind mount. We also tell Traefik to find our dynamic configuration in configurations/dynamic.yml. And at last is the configuration for our SSL certificate resolver.

```
api:
  dashboard: true
```

```
entryPoints:
  web:
    address: :80
    http:
      redirections:
        entryPoint:
          to: websecure

  websecure:
    address: :443
    http:
      middlewares:
        - secureHeaders@file
      tls:
        certResolver: letsencrypt

providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
    exposedByDefault: false
  file:
    filename: /configurations/dynamic.yml

certificatesResolvers:
  letsencrypt:
    acme:
      email: raf@yourdomain.com
      storage: acme.json
      keyType: EC384
      httpChallenge:
        entryPoint: web
```

**Note:** Make sure to configure an email in this file for the Let's Encrypt renewal. @yourdomain.com might throw an error when you want to run your docker container!

## II. dynamic.yml

This file contains our middlewares to make sure all our traffic is fully secure and runs over TLS. We also set the basic auth here for our Traefik dashboard, because by default it is accessible for everyone.

The file is fully dynamic and can be edited on the fly, without restarting our container.

```
http:
```

```

middlewares:
  secureHeaders:
    headers:
      sslRedirect: true
      forceSTSHeader: true
      stsIncludeSubdomains: true
      stsPreload: true
      stsSeconds: 31536000

  user-auth:
    basicAuth:
      users:
        - "raf:$apr1$MTqfVwiE$FKkzT5ERGFqwH9f3uipxA1"

tls:
  options:
    default:
      cipherSuites:
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305
        - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
      minVersion: VersionTLS12

```

### III. docker-compose.yml

The most important file. This is where the good stuff happens. So the beauty of Traefik is that once you have done the initial set-up, deploying new containers is very easy. It works by specifying labels for your containers.

```

version: "3"

services:
  traefik:
    image: traefik:latest
    container_name: traefik
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    networks:
      - proxy
    ports:
      - 80:80
      - 443:443

```

volumes:

- /etc/localtime:/etc/localtime:ro
- /var/run/docker.sock:/var/run/docker.sock:ro
- ./traefik-data/traefik.yml:/traefik.yml:ro
- ./traefik-data/acme.json:/acme.json
- ./traefik-data/configurations:/configurations

labels:

- "traefik.enable=true"
- "traefik.docker.network=proxy"
- "traefik.http.routers.traefik-secure.entrypoints=websecure"
- "traefik.http.routers.traefik-secure.rule=Host(`traefik.yourdomain.com`)"
- "traefik.http.routers.traefik-secure.service=api@internal"
- "traefik.http.routers.traefik-secure.middlewares=user-auth@file"

portainer:

image: portainer/portainer-ce:latest

container\_name: portainer

restart: unless-stopped

security\_opt:

- no-new-privileges:true

networks:

- proxy

volumes:

- /etc/localtime:/etc/localtime:ro
- /var/run/docker.sock:/var/run/docker.sock:ro
- ./portainer-data:/data

labels:

- "traefik.enable=true"
- "traefik.docker.network=proxy"
- "traefik.http.routers.portainer-secure.entrypoints=websecure"
- "traefik.http.routers.portainer-secure.rule=Host(`portainer.yourdomain.com`)"
- "traefik.http.routers.portainer-secure.service=portainer"
- "traefik.http.services.portainer.loadbalancer.server.port=9000"

networks:

proxy:

external: true

For every container that you want Traefik to handle, you add labels so Traefik knows where it should route it. So when we look at the file above, let's quickly check what is going on at the traefik container.

So we attach the first label, which tells Traefik that it should route this container because we specify `enable=true`. This is the result of the configuration in the static `traefik.yml` file where we explicitly stated `exposedByDefault: false` so therefore we have to specify that.



The second label tells us that we should use the network proxy, which we will create later on. After that we tell Traefik to use our websecure endpoint (https). We then specify our host name with the appropriate domain. 🍑

The final to last label specifies the API handler. It exposes information such as the configuration of all routers, services, middlewares, etc. To see all the available endpoints you can check [the docs](#).

The very last label is our basic auth middleware, remember? Because the Traefik dashboard is exposed by default so we add a basic security layer over it. It will also protect our API.

labels:

- "traefik.enable=true"
- "traefik.docker.network=proxy"
- "traefik.http.routers.traefik-secure.entrypoints=websecure"
- "traefik.http.routers.traefik-secure.rule=Host(`traefik.yourdomain.com`)"
- "traefik.http.routers.traefik-secure.service=api@internal"
- "traefik.http.routers.traefik-secure.middlewares=user-auth@file"

## Running our stack 🚀

### I. Creating credentials

So the first thing we should do is generate the password for basic auth that will be stored in the `dynamic.yml` file. These credentials will be required when trying to log into our Traefik Web UI and it will protect the API.

Make sure your server has `htpasswd` installed. If it doesn't you can do so with the following command:

```
$ sudo apt install apache-utils
```

Then run the below command, replacing the username and password with the one you want to use.

```
$ echo $(htpasswd -nb <username> <password>)
```

Edit the `dynamic.yml` file and add your auth string under the `user-auth` middleware as seen in the example code.

### II. Creating the proxy network

We need to create a new Docker network that will allow outside traffic. This should be called proxy as we specified in our docker-compose.yml file:

To create a docker network use:

```
$ docker network create proxy
```

### III. Editing the domain names

Open the docker-compose.yml file and make sure you replace the domain values in the Traefik labels to the domains that you send to the server as done earlier:

```
traefik.yourdomain.com  
portainer.yourdomain.com
```

### IV. Giving the proper permissions to acme.json

By default the file acme.json has the permission set to 644, this will result in a error when running docker-compose. So make sure you set the permissions of that particular file to 600. cd into the core folder and run the following command:

```
$ sudo chmod 600 ./traefik-config/acme.json
```

### V. Running the stack

Now it is time to run the stack. Make sure you are in the core folder so docker can find the docker-compose file. On the first run I always like to check the process for errors before we use the docker-compose --detach flag. Run the following command:

```
$ sudo docker-compose up
```

Right now the Traefik dashboard should be available at traefik.yourdomain.com and portainer.yourdomain.com, awesome! 🔥

When you are sure that your containers are running correctly, run them in the background by using the --detach option:

```
$ sudo docker-compose down && sudo docker-compose up -d
```

# Adding docker applications to our server 🏆

Alright so our environment is all configured, let me show you now how easy it is to deploy containers to our new Traefik set-up.

This is where the magic happens. It took me 2 minutes in total to find a dockerized app on the internet, deploy it and make it available to the world wide web. Can you believe that? **TWO MINUTES!** If all my deployments were that easy..

Anyway, here are the steps I took.

I pointed the domain I want to use for the app, to the server. For this example I used `express.domain.com`.

After that I google'd "docker express starter" and I found a repo and forked it. Then on the server I switched to the apps folder and ran `git clone`

```
$ git clone https://github.com/rafrasenberg/docker-express-postgres ./express
```

After that it was time to edit the `docker-compose.yml` file of our app:

```
version: "3"
services:
  app:
    build: .
    depends_on:
      - postgres
    environment:
      DATABASE_URL: postgres://user:pass@postgres:5432/db
      NODE_ENV: development
      PORT: 3000
    ports:
      - "3000:3000"
    command: npm run dev
    volumes:
      - ./app/
      - /app/node_modules

  postgres:
    image: postgres:10.4
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: user
```

```
POSTGRES_PASSWORD: pass
POSTGRES_DB: db
```

Now you might be wondering, how should I approach this? The first thing we do is remove the `ports` section, as Traefik will take care of this. For the whole Traefik config we only have to add 4 labels:

```
labels:
  - "traefik.enable=true"
  - "traefik.docker.network=proxy"
  - "traefik.http.routers.app-secure.entrypoints=websecure"
  - "traefik.http.routers.app-secure.rule=Host(`express.yourdomain.com`)"
```

So what is happening here?

First we enable this container with `enable=true`, then we add it to the proxy network. After that we specify the routers and the entrypoints.

Note that this part: `traefik.http.router.app-secure` should have an unique router identification. So make sure you haven't used that name yet. Let's say you want to deploy the exact same app on a different domain and container instance, you could use this label: `traefik.http.router.app1-secure`. Just make sure it's an unique value.

Now the last part that we need to do in the `docker-compose.yml` file is specifying the networks. So the final `docker-compose.yml` file will look like this:

```
version: "3"
services:
  app:
    build: .
    depends_on:
      - postgres
    environment:
      DATABASE_URL: postgres://user:pass@postgres:5432/db
      NODE_ENV: development
      PORT: 3000
    command: npm run dev
    volumes:
      - ./app/
      - /app/node_modules
    networks:
      - proxy
```

```

- default
labels:
- "traefik.enable=true"
- "traefik.docker.network=proxy"
- "traefik.http.routers.app-secure.entrypoints=websecure"
- "traefik.http.routers.app-secure.rule=Host(`express.rasenberg.tech`)"

```

```

postgres:
  image: postgres:10.4
  ports:
    - "35432:5432"
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: pass
    POSTGRES_DB: db

```

```

networks:
  proxy:
    external: true

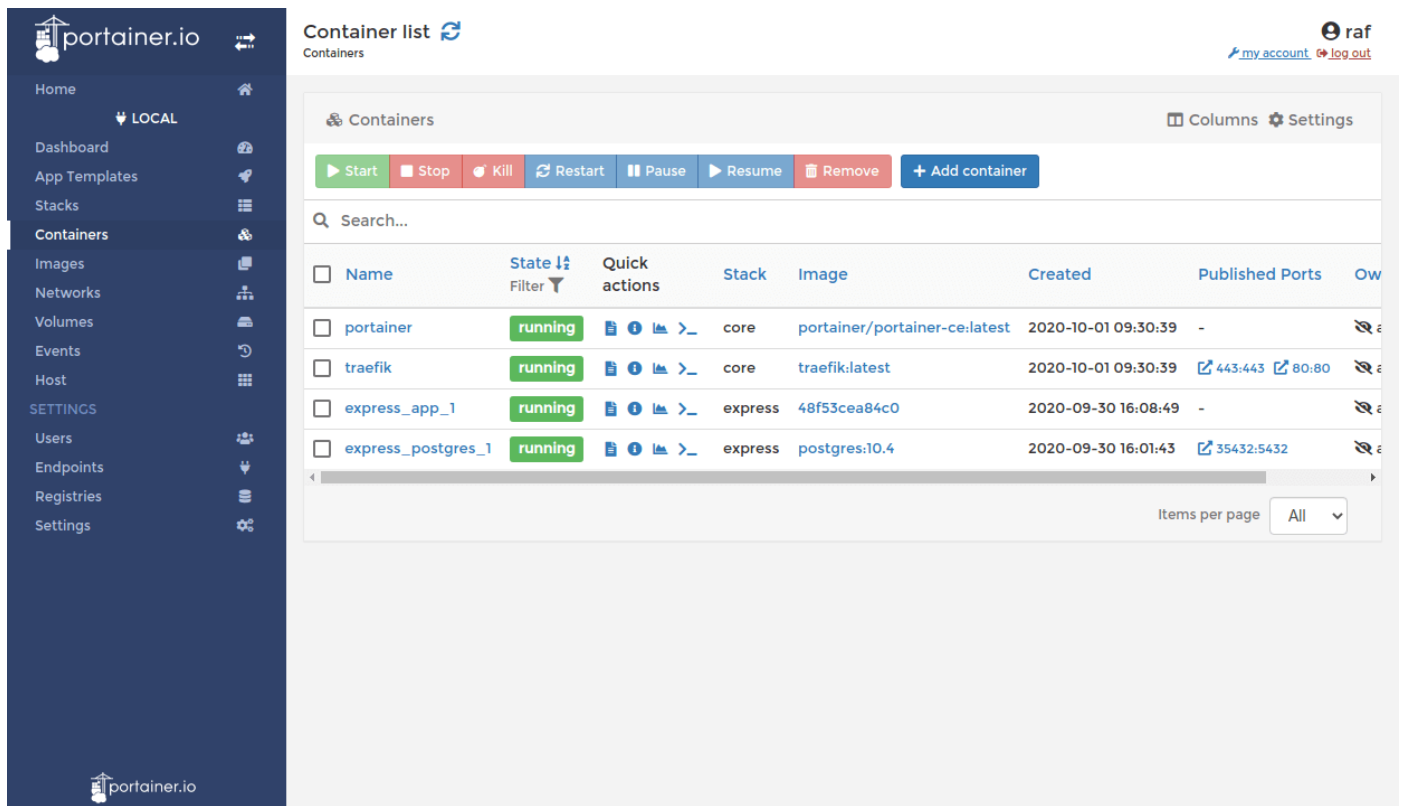
```

Now let’s run our container:

```
$ sudo docker-compose up -d
```

That’s all! We literally only added less than 10 lines to our docker-compose.yml file and our container is deployed and ready to receive traffic. Awesome right! 🙌

Now our new app is also showing up in Portainer:



Now whenever you want to add a new applications on your server, just

repeat the last few steps. Easy as that! 🚀

## Conclusion ⚡

Let's recap this.

We have set-up an awesome configuration stack for running and managing multiple docker containers on one server. Deploying new projects will be very easy after this initial set-up.

Something I want to cover in the next post is integrating a basic CI pipeline that connects with our droplet so we can automatically update our containers on a code push to Github. So stay tuned for that!

See you next time! 🙌