Traefik

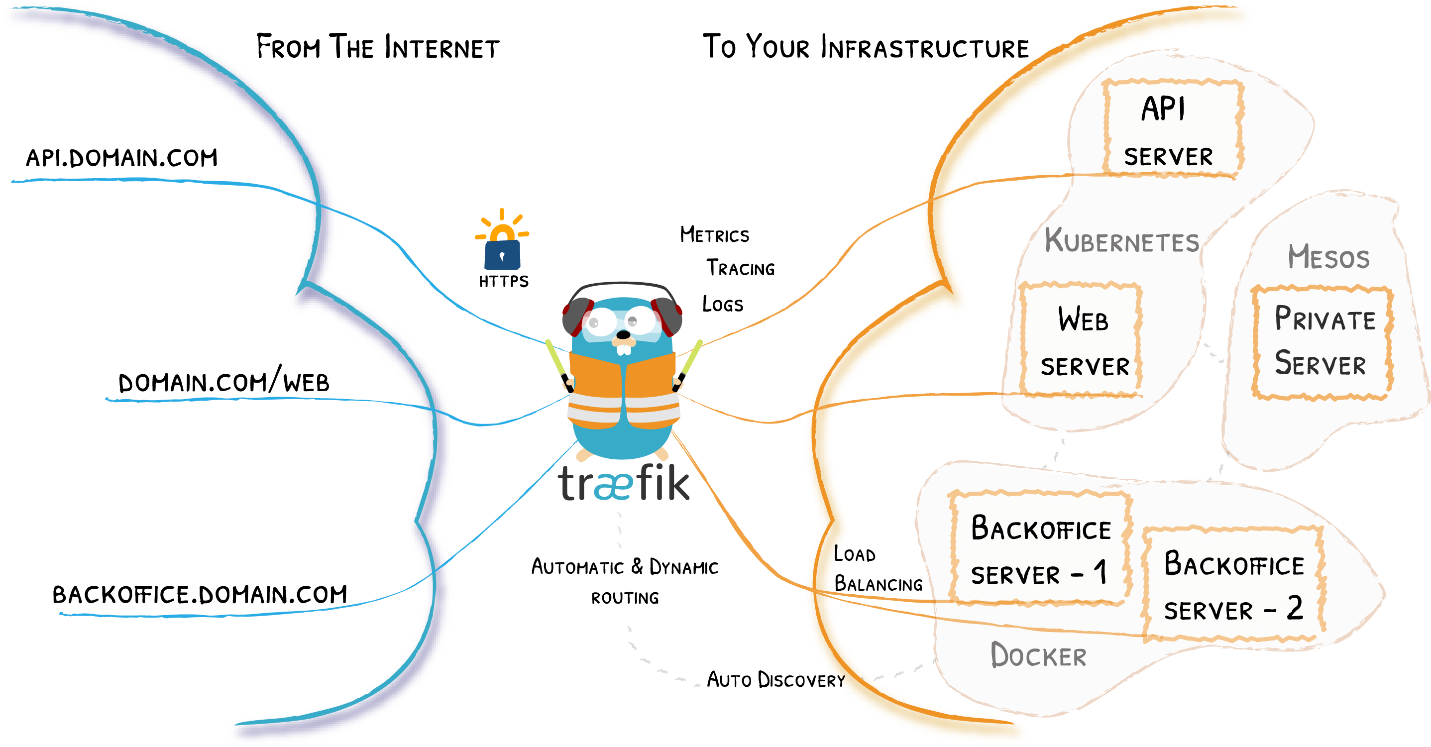
**Introduction:**

Traefik is an open-source Edge Router that makes publishing services a fun and easy experience. It receives requests on behalf of the system and finds out which components are responsible for handling them.

What sets Traefik apart, besides its many features, is that it automatically discovers the right configuration for specific services. The magic happens when Traefik inspects the infrastructure, where it finds relevant information and discovers which service serves which request. It simply means that Traefik by its own recognizes the infra and acts accordingly

Traefik is natively compliant with every major cluster technology, such as Kubernetes, Docker, Docker Swarm, AWS, Mesos, Marathon, and the list goes on; and can handle many at the same time. (It even works for legacy software running on bare metal.)

With Traefik, there is no need to maintain and synchronize a separate configuration file: everything happens automatically, in real time (no restarts, no connection interruptions). With Traefik, less time is spent on developing and deploying new features to the system, not on configuring and maintaining its working state.



**Concepts:**

Following are the concepts of Traefik:

* **Edge Router:** Traefik is an Edge Router, it means that it's the door to the specified platform, and that it intercepts and routes every incoming request: it knows all the logic and every rule that determine which services handle which requests (based on the path, the host, headers, and so on ...).
* **Auto Service Discovery:** Where traditionally edge routers (or reverse proxies) need a configuration file that contains every possible route to your services, Traefik gets them from the services themselves. Deploying with services, only attach information that tells Traefik the characteristics of the requests the services can handle.

It means that when a service is deployed, Traefik detects it immediately and updates the routing rules in real time. The opposite is true: when you remove a service from your infrastructure, the route will disappear accordingly.

You no longer need to create and synchronize configuration files cluttered with IP addresses or other rules.

**Installation Procedure of Traefik:**

Traefik can be installed in following manner:

* Using the official Docker Image.
* Using Helm Chart.
* Using the Binary Distribution.
* Compiling the binaries from the source.

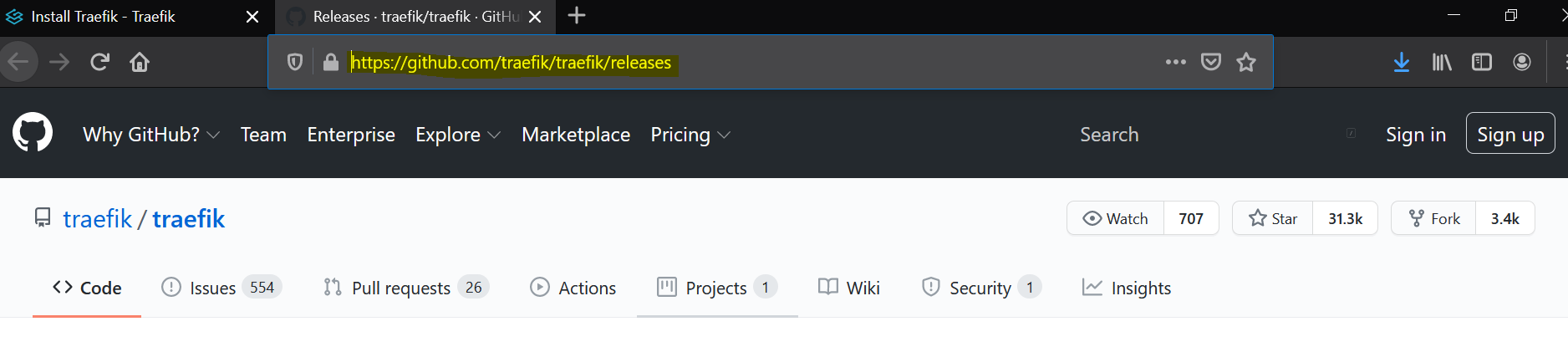
The best way to install Traefik is to use the official Docker Image. Here, We would be using the Binary Distribution approach.

Pre-requisites:

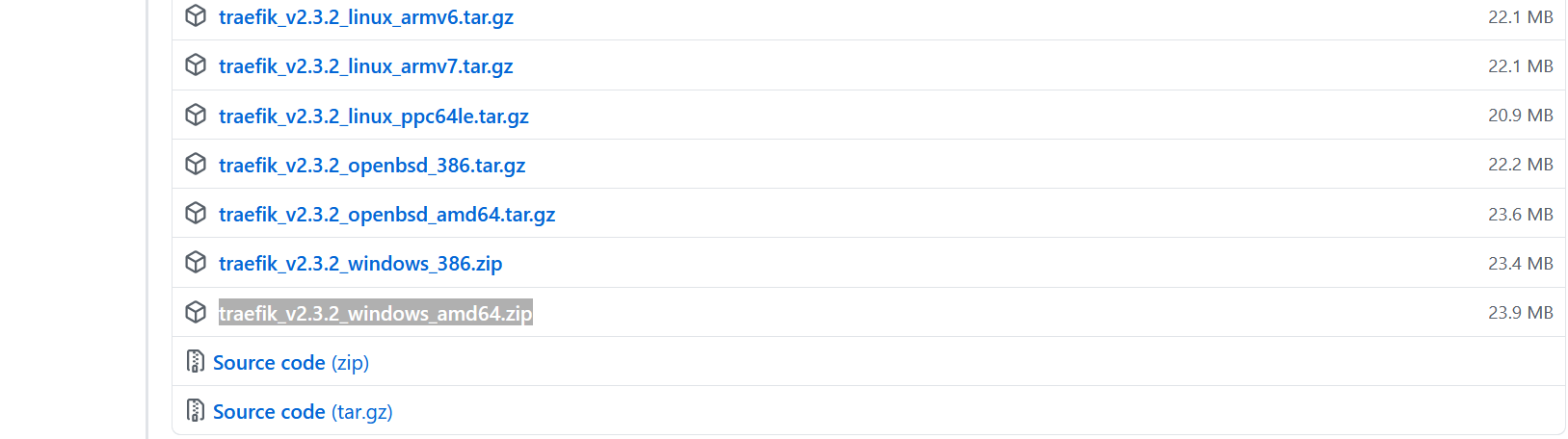
* RAM: 4gb
* CPU: 2 Cores
* OS: Windows Server 2012/2016/2019
* Storage: 50 GB or standard

The very first step is to download the latest binary file from the official Traefik releases pages:

Link: <https://github.com/traefik/traefik/releases>



Go to the above link and scroll down to the file traefik\_v2.3.2\_windows\_amd64.zip.



And download it. Once Downloaded open the windows PowerShell in your system.

Now run the following command in order to verify the integrity of the downloaded exe.

**${traefik\_version}\_checksums.txt**

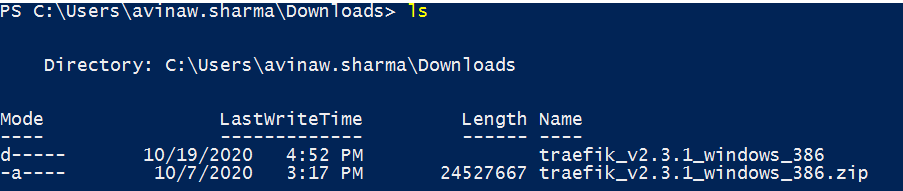
**Get-FileHash ./traefik\_v2.3.2\_windows\_amd64.zip -Algorithm SHA256**

Now compare the values and if the values don’t match that means the downloaded file is corrupted or incompletely downloaded.

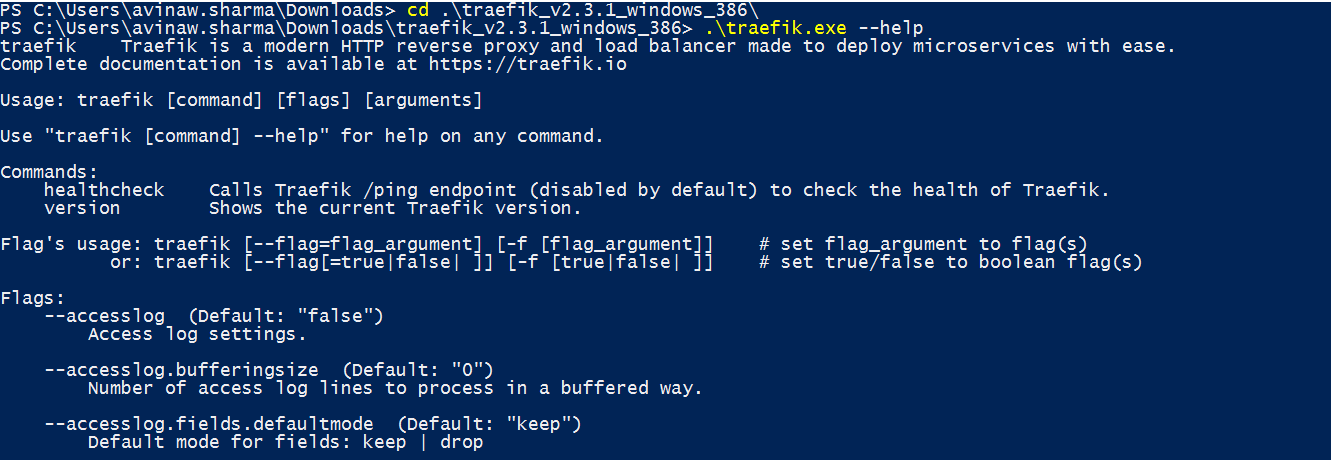
Now extract the downloaded archive:

**Expand-Archive traefik\_v2.3.2\_windows\_amd64.zip**

Now you will get a folder as:



Now cd traefik\_v2.3.1\_windows\_386.zip and run ./traefik.exe --help



**Configuration:**

Configuration in traefik can be defined in two ways:

* The fully dynamic routing configuration (referred to as the dynamic configuration).
* The startup configuration (referred to as the static configuration).

Dynamic Configuration:

Traefik gets its dynamic configuration from providers: whether an orchestrator, a service registry, or a plain old configuration file. Different methods based on the infrastructure which can be passed as a Dynamic Configuration file with Treafik.exe are:

* Docker
* Kubernetes Ingress Rule
* Consul catalog
* ECS
* Marathon
* Rancher
* File
* Consul
* Etcd
* Zookeeper
* Redis
* HTTP

Here we would be using the File provider to pass the Dynamic Configuration to the Traefik.exe.

**Static Configuration:**

There are three different, **mutually exclusive** (e.g. you can use only one at the same time), ways to define static configuration options in Traefik:

1. In a configuration file
2. In the command-line arguments.
3. As Environment Variables.

These ways are evaluated in the order listed above.If no value was provided for a given option, a default value applies. Moreover, if an option has sub-options, and any of these sub-options is not specified, a default value will apply as well.

For example, the --providers.docker option is enough by itself to enable the Docker provider, even though sub-options like --providers.docker.endpoint exist. Once positioned, this option sets (and resets) all the default values of the sub-options of --providers.docker.

**Configuration File:**

At startup, Traefik searches for a file named traefik.toml/traefik.yaml/traefik.yml.

Note: Don’t use any other name.

The search order is:

* /etc/traefik
* $XDG\_CONFIG\_HOME/
* $HOME/.config/
* . (the Working Directory)

In order to override the configuration, a configuration file path can be defined as the command line argument.

**traefik --configFile=foo/bar/myconfigfile.toml**

**Arguments:**

To have a full list of Arguments use:

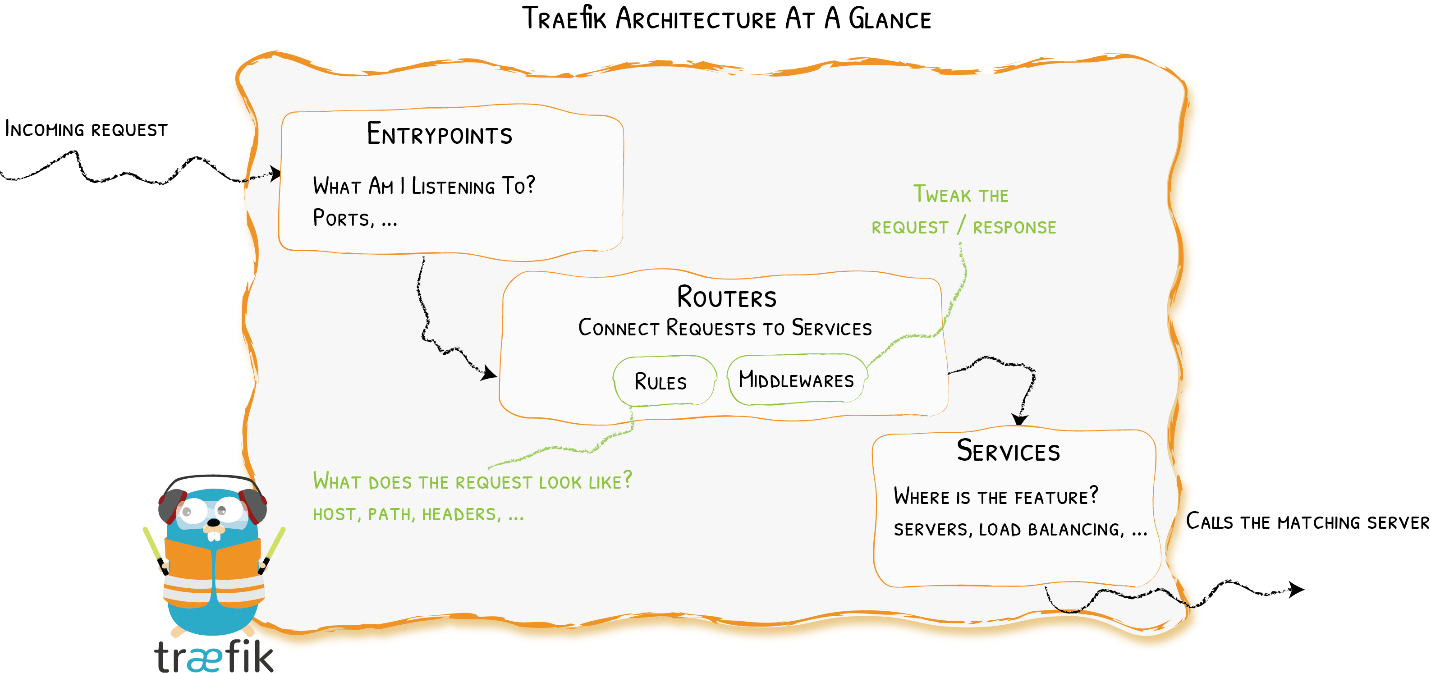
**traefik --help**

**Environment Variables:**

The available Environment Variables can be found at:

“https://doc.traefik.io/traefik/reference/static-configuration/env/”

**Load balancing with Traefik:**



In order to achieve load balancing with traefik few pre-requisites are to be fulfilled:

* 2 Webservers, in this case 2 simple webpages hosted on IIS.
* Understanding the Configuration file.

Assuming 2 web Servers are already hosted let’s discuss about the Configuration Files:

Here we would be using the dynamic configuration file strategy that means we would be having 2 separate files:

* Main Traefik.toml to run the traefik and expose the dashboard.
* Config.toml specifying the Load Balancers and its different properties.

Illustration of Traefik.Toml:

The Traefik.toml file is created in the working directory.

defaultEntryPoints = ["http"]

[api]

    dashboard = true

    insecure  = true

[entryPoints.ping]

    address = ":8082"

[ping]

  entryPoint = "ping"

[entryPoints]

    [entryPoints.http]

        address = ":8888"

#[web]

#address = ":8080"

[providers]

  # Enable the file provider to define routers / middlewares / services in file

  [providers.file]

   Directory = "C:\\rename"

   watch = true

###############################################################

# Log Configuration

###############################################################

[accessLog]

  filePath = "C:\\Users\\avinaw.sharma\\Desktop\\New folder\\access.log"

  format = "json"

[log]

  level = "INFO"

  filePath = "C:\\Users\\avinaw.sharma\\Desktop\\New folder\\Traefik.log"

  format   = "json"

The above is the traefik.toml file to be stored in the same directory as that of the traefik.exe as traefik searches for the configuration in the working directory.

Major building blocks:

* Providers discover the services that live on your infrastructure (their IP, health, ...)
* Entrypoints listen for incoming traffic (ports, ...)
* Routers analyze the requests (host, path, headers, SSL, ...)
* Services forward the request to your services (load balancing, ...)
* Middlewares may update the request or make decisions based on the request (authentication, rate limiting, headers, ...)

Here, the default EntryPoint is “HTTP” i.e., the incoming traffic is HTTP. The [api] option is used to enable or disable all the different api’s pre embedded in traefik. Like, here the traefik dashboard is enabled which means that an api named dashboard is enabled.

Traefik dashboard is important as it shows the entries on a web browser and through It all changes are reflected. The below is our traefik dashboard which can be accessed on:

**http://localhost:8080/dashboard/#/**

There are 2 ways to configure and access the dashboard:

* Secure mode (Recommended)
* Insecure mode

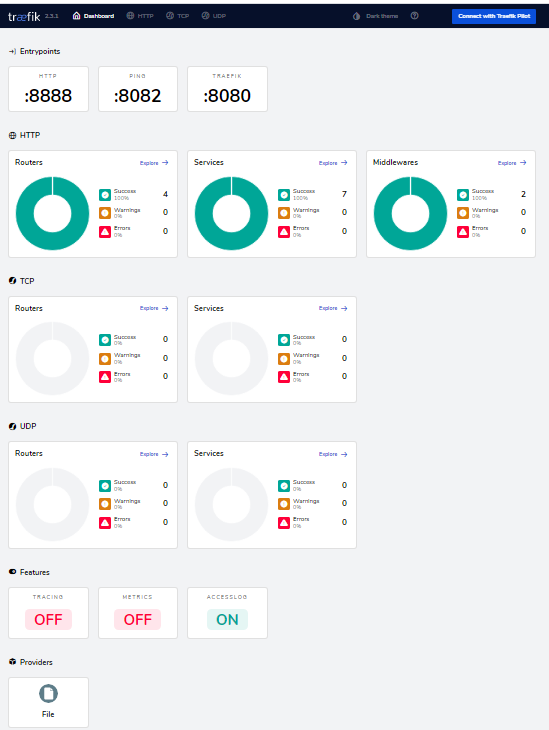
**Secure mode:** This method is recommended and can be enabled by making insecure = false or not defining insecure as it is the default value.

Then define a routing configuration on Traefik itself, with a router attached to the service api@internal in the dynamic configuration, to allow defining:

* One or more security features through middlewares like authentication (basicAuth, digestAuth, forwardAuth) or whitelisting.
* A router rule for accessing the dashboard, through Traefik itself (sometimes referred as "Traefik-ception").

**Insecure mode:** This mode is not recommended because it does not allow the use of security features. It is defined as above.

Note: Make sure to have the trailing slash after dashboard in the URL.



**Ping:** Ping is the health check for traefik itself. The /ping health-check URL is enabled with the command-line --ping or config file option [ping].

The entryPoint where the /ping is active can be customized with the entryPoint option, whose default value is traefik (port 8080) as done above.

**manualRouting:**

Optional, Default=false

If manualRouting is true, it disables the default internal router in order to allow one to create a custom router for the ping@internal service.

[ping]

manualRouting = true

**terminatingStatusCode**:

Optional, Default=503

During the period in which Traefik is gracefully shutting down, the ping handler returns a 503 status code by default. If Traefik is behind e.g. a load-balancer doing health checks (such as the Kubernetes LivenessProbe), another code might be expected as the signal for graceful termination. In which case, the terminatingStatusCode can be used to set the code returned by the ping handler during termination.

[ping]

terminatingStatusCode = 204

Here, in the above configuration ping can be accessed on a web browser at:

**http://localhost:8082/ping/**

Now we need to define the entryPoint of the load balancer defined as:

[entryPoints]

    [entryPoints.http]

        address = ":8888"

Now before accessing the load balancer at port 8888 we need to configure the provider file. In the provider file we need to define all the configurations of the load balancer. The connection of traefik with provider is done as:

[providers]

  # Enable the file provider to define routers / middlewares / services in file

  [providers.file]

   Directory = "C:\\rename"

   watch = true

**watch**

Set the watch option to true to allow Traefik to automatically watch for file changes.  
It works with both the filename and the directory options.

**LOGS:**

By default, logs are written to stdout, in text format. can configure a file path instead using the filePath option.

In order to enable different logs, we can define as:

[accessLog]

  filePath = "C:\\Users\\avinaw.sharma\\Desktop\\New folder\\access.log"

  format = "json"

[log]

  level = "INFO"

  filePath = "C:\\Users\\avinaw.sharma\\Desktop\\New folder\\Traefik.log"

  format   = "json"

By default, the level is set to ERROR. Alternative logging levels are DEBUG, PANIC, FATAL, ERROR, WARN, and INFO.

Illustration of Config.Toml:

[http]

# Create router for http

    [http.routers.my-routers]

        entryPoints = ["http"]

        service = "service-http"

        #rule = "PathPrefix('/')"

        rule = "Path(`/`)"

## create service

    [http.services]

       [http.services.service-http.loadBalancer]

        [http.services.my-service.loadBalancer.sticky.cookie]

            name = "my\_sticky\_cookie"

            secure = true

            httpOnly = true

            sameSite = "none"

        [http.services.healthcheck.loadBalancer.healthCheck]

            path = "/health"

            interval = "10s"

            timeout = "3s"

           [[http.services.service-http.loadBalancer.servers]]

               url = "http://10.1.160.106/"

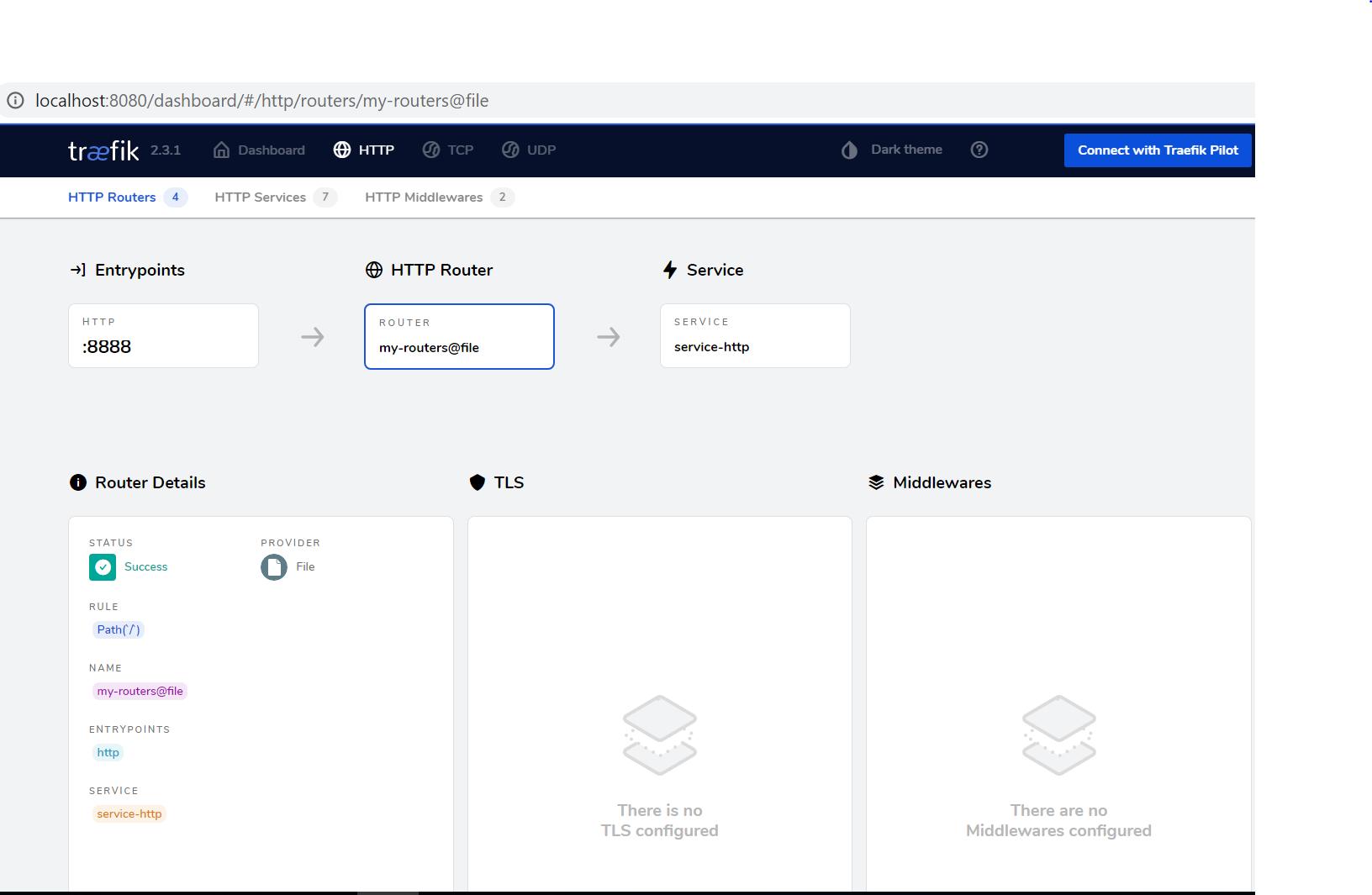
           [[http.services.service-http.loadBalancer.servers]]

               url = "http://10.1.160.124/"

The above is Config.toml which is passed as input in the provider configuration block in traefik.toml.

Config.toml is the file in which we define the logic to be achieved with traefik. Here, we are using traefik as a load balancer thereby applying the logic respective to that.

http is the entryPoint definition is given in this file. http is the entryPoint defined for load balancer in traefik.toml. Before creating load balancer, a router needs to be created as it is in charge of connecting incoming requests to the services that can handle them. A load balancer is connected as a service with rule of path /.



Now a service is to be created for the load balancer named my-service.

[http.services]

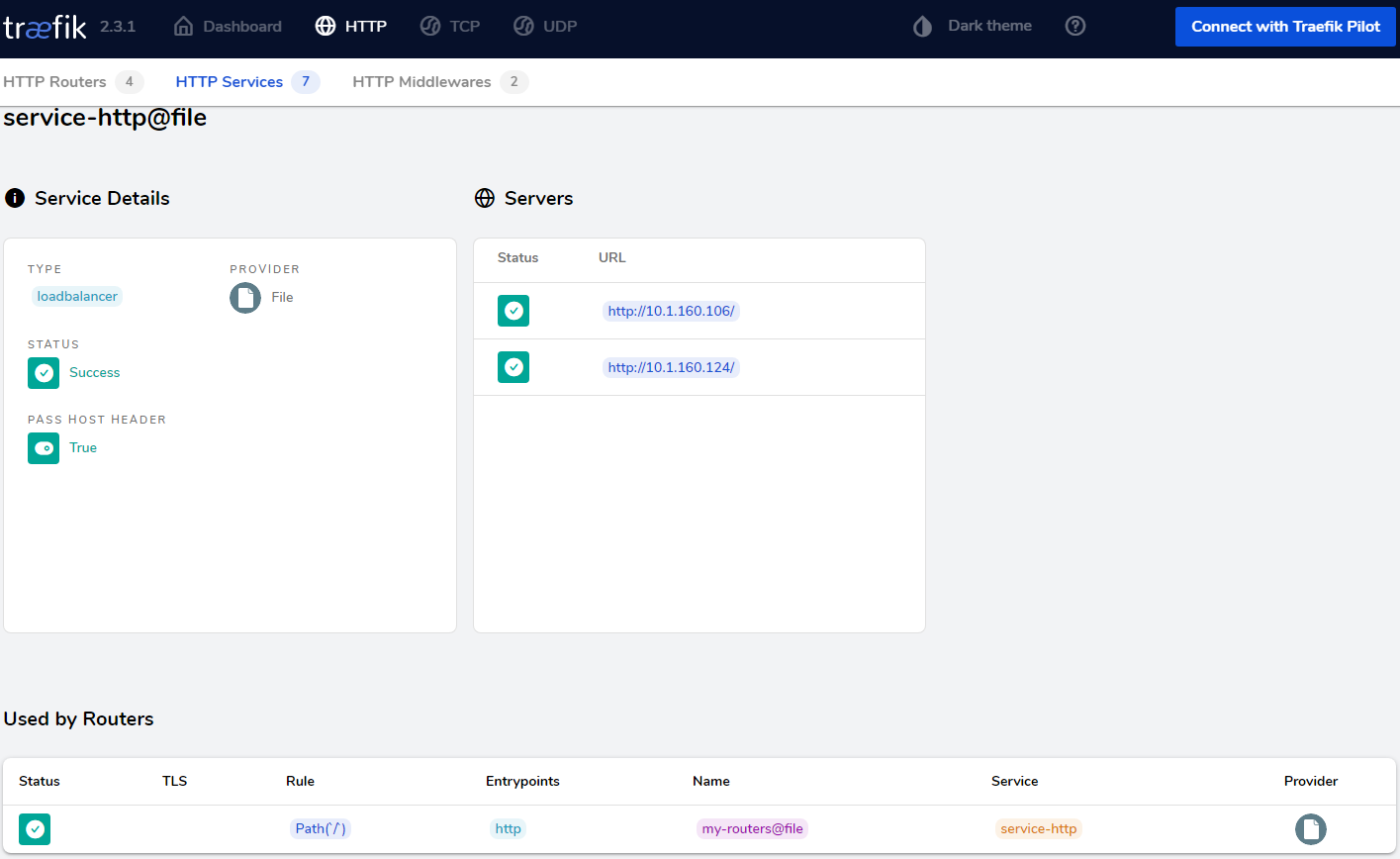
[[http.services.service-http.loadBalancer.servers]]

               url = "http://10.1.160.106/"

           [[http.services.service-http.loadBalancer.servers]]

               url = "http://10.1.160.124/"

url has the IP addresses of the web servers to be kept under the load balancer.



After this an healthcheck need to be created for the load balancer as:

[http.services.healthcheck.loadBalancer.healthCheck]

            path = "/health"

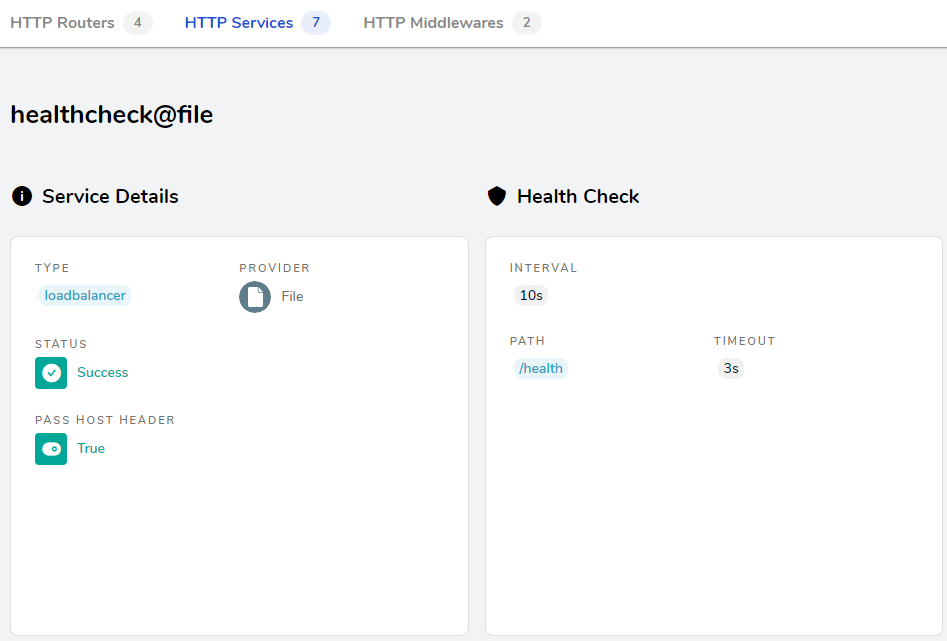
            interval = "10s"

            timeout = "3s"

Traefik will consider your servers healthy as long as they return status codes between 2XX and 3XX to the health check requests (carried out every interval).

Below are the available options for the health check mechanism:

* path is appended to the server URL to set the health check endpoint.
* scheme, if defined, will replace the server URL scheme for the health check endpoint
* hostname, if defined, will apply Host header hostname to the health check request.
* port, if defined, will replace the server URL port for the health check endpoint.
* interval defines the frequency of the health check calls.
* timeout defines the maximum duration Traefik will wait for a health check request before considering the server failed (unhealthy).
* headers define custom headers to be sent to the health check endpoint.
* followRedirects defines whether redirects should be followed during the health check calls (default: true).



Now Finally a sticky session bit needs to be created for the user session maintenance:

[http.services.service-http.loadBalancer]

        [http.services.my-service.loadBalancer.sticky.cookie]

            name = "my\_sticky\_cookie"

            secure = true

            httpOnly = true

            sameSite = "none"

When sticky sessions are enabled, a cookie is set on the initial request and response to let the client know which server handles the first response. On subsequent requests, to keep the session alive with the same server, the client should resend the same cookie.

* If the server specified in the cookie becomes unhealthy, the request will be forwarded to a new server (and the cookie will keep track of the new server).
* The default cookie name is an abbreviation of a sha1 (ex: \_1d52e).
* By default, the affinity cookie is created without those flags. One however can change that through configuration.
* SameSite can be none, lax, strict or empty.

