# Traefik & Docker


- Configuration Example
- Configuration Options
  - endpoint
  - useBindPortIP
  - defaultRule
  - constraints
- Routing Configuration
  - Routing Configuration with Labels
  - Port Detection
  - Host networking
  - IPv4 & IPv6
  - Docker API Access


!!! tip "The Quick Start Uses Docker"

```
If you have not already read it, maybe you would like to go through the [quick start guide](../../../getting-started/index.md) that uses the Docker provider.
```

# Configuration Example

You can enable the Docker provider as detailed below:

```
providers:
  docker: {}
```

```
[providers.docker]
```

```
--providers.docker=true
```

Attach labels to containers (in your Docker compose file)

```
services:
  my-container:
    # ...
    labels:
      - traefik.http.routers.my-container.rule=Host(`example.com`)
```

# Configuration Options

| Field | Description | Default | Required |
|---|---|---|---|
| `providers.providersThrottleDuration` | Minimum amount of time to wait for, after a configuration reload, before taking into account any new configuration refresh event.<br>If multiple events occur within this time, only the most recent one is taken into account, and all others are discarded.<br>**This option cannot be set per provider, but the throttling algorithm applies to each of them independently.** | 2s | No |
| `providers.docker.endpoint` | Specifies the Docker API endpoint. See here for more information | "unix:///var/run/docker.sock" | No |
| `providers.docker.username` | Defines the username for Basic HTTP authentication. This should be used when the Docker daemon socket is exposed through an HTTP proxy that requires Basic HTTP authentication. | "" | No |
| `providers.docker.password` | Defines the password for Basic HTTP authentication. This should be used when the Docker daemon socket is exposed through an HTTP proxy that requires Basic HTTP authentication. | "" | No |

| Field | Description | Default | Required |
| --- | --- | --- | --- |
| `providers.docker.useBindPortIP` | Instructs Traefik to use the IP/Port attached to the container's binding instead of its inner network IP/Port.<br>See here for more information | false | No |
| `providers.docker.exposedByDefault` | Expose containers by default through Traefik. If set to *false*, containers that do not have a `traefik.enable=true` label are ignored from the resulting routing configuration.<br>See here for additional information | true | No |
| `providers.docker.network` | Defines a default docker network to use for connections to all containers. This option can be overridden on a per-container basis with the `traefik.docker.network` label. | "" | No |
| `providers.docker.defaultRule` | Defines what routing rule to apply to a container if no rule is defined by a label.<br>See here for more information. | `"Host(`{{ normalize .Name }}`)"` | No |
| `providers.docker.httpClientTimeout` | Defines the client timeout for HTTP connections. Accepts a duration string (e.g., `30s`, `1m30s`) or an integer value in seconds. If its value is 0, no timeout is set. | 0 | No |
| `providers.docker.watch` | Instructs Traefik to watch Docker events or not. | true | No |
| `providers.docker.constraints` | Defines an expression that Traefik matches against the container labels to determine whether to create any route for that container. See here for more information. | "" | No |

| Field | Description | Default | Required |
|---|---|---|---|
| `providers.docker.allowEmptyServices` | Instructs the provider to create any servers load balancer defined for Docker containers regardless of the healthiness of the corresponding containers. | false | No |
| `providers.docker.tls.ca` | Defines the path to the certificate authority used for the secure connection to Docker, it defaults to the system bundle. | "" | No |
| `providers.docker.tls.cert` | Defines the path to the public certificate used for the secure connection to Docker. When using this option, setting the `key` option is required. | "" | Yes |
| `providers.docker.tls.key` | Defines the path to the private key used for the secure connection to Docker. When using this option, setting the `cert` option is required. | "" | Yes |
| `providers.docker.tls.insecureSkipVerify` | Instructs the provider to accept any certificate presented by the Docker server when establishing a TLS connection, regardless of the hostnames the certificate covers. | false | No |

# `endpoint`

See the Docker API Access section for more information.

??? example "Using the docker.sock"

````
The docker-compose file shares the docker sock with the Traefik container

```yaml
services:
  traefik:
    image: traefik:v3.1 # The official v3 Traefik docker image
    ports:
````
````

````
        - "80:80"
      volumes:
        - /var/run/docker.sock:/var/run/docker.sock
```

We specify the docker.sock in traefik's configuration file.

```yaml tab="File (YAML)"
providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
     # ...
```

```toml tab="File (TOML)"
[providers.docker]
  endpoint = "unix:///var/run/docker.sock"
  # ...
```

```bash tab="CLI"
--providers.docker.endpoint=unix:///var/run/docker.sock
# ...
```
````

??? example "Using SSH"

````
Using Docker 18.09+ you can connect Traefik to daemon using SSH.
We specify the SSH host and user in Traefik's configuration file.
Note that if the server requires public keys for authentication, you must have them accessible
for the user running Traefik.

```yaml tab="File (YAML)"
providers:
  docker:
    endpoint: "ssh://traefik@192.168.2.5:2022"
     # ...
```

```toml tab="File (TOML)"
````

````
[providers.docker]
  endpoint = "ssh://traefik@192.168.2.5:2022"
  # ...
```

```bash tab="CLI"
--providers.docker.endpoint=ssh://traefik@192.168.2.5:2022
# ...
```
````

??? example "Using HTTP"

````
Using Docker Engine API you can connect Traefik to remote daemon using HTTP.

```yaml tab="File (YAML)"
providers:
  docker:
    endpoint: "http://127.0.0.1:2375"
     # ...
```

```toml tab="File (TOML)"
[providers.docker]
  endpoint = "http://127.0.0.1:2375"
  # ...
```

```bash tab="CLI"
--providers.docker.endpoint=http://127.0.0.1:2375
# ...
```
````

??? example "Using TCP"

````
Using Docker Engine API you can connect Traefik to remote daemon using TCP.

```yaml tab="File (YAML)"
providers:
  docker:
    endpoint: "tcp://127.0.0.1:2375"
````

````
    # ...
```


```toml tab="File (TOML)"
[providers.docker]
  endpoint = "tcp://127.0.0.1:2375"
  # ...
```


```bash tab="CLI"
--providers.docker.endpoint=tcp://127.0.0.1:2375
# ...
```
````

```
providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
```

```
[providers.docker]
  endpoint = "unix:///var/run/docker.sock"
```

```
--providers.docker.endpoint=unix:///var/run/docker.sock
```

# `useBindPortIP`

Traefik routes requests to the IP/port of the matching container. When setting `useBindPortIP=true`, you tell Traefik to use the IP/Port attached to the container's *binding* instead of its inner network IP/Port.

When used in conjunction with the `traefik.http.services.<name>.loadbalancer.server.port` label (that tells Traefik to route requests to a specific port), Traefik tries to find a binding on port `traefik.http.services.<name>.loadbalancer.server.port`. If it cannot find such a binding, Traefik falls back on the internal network IP of the container, but still uses the `traefik.http.services.<name>.loadbalancer.server.port` that is set in the label.

??? example "Examples of `usebindportip` in different situations."

```
| port label         | Container's binding                                 | Routes to      |
|--------------------|-----------------------------------------------------|----------------|
| <a id="opt-row" href="#opt-row" title="#opt-row">-</a> |           -
                                      | IntIP:IntPort  |
```
```

```
| <a id="opt-row-2" href="#opt-row-2" title="#opt-row-2">-</a> |
ExtPort:IntPort                                     | IntIP:IntPort  |
| <a id="opt-row-3" href="#opt-row-3" title="#opt-row-3">-</a> |
ExtIp:ExtPort:IntPort                               | ExtIp:ExtPort  |
| <a id="opt-LblPort" href="#opt-LblPort" title="#opt-LblPort">LblPort</a> |           -
                                        | IntIp:LblPort  |
| <a id="opt-LblPort-2" href="#opt-LblPort-2" title="#opt-LblPort-2">LblPort</a> |
ExtIp:ExtPort:LblPort                               | ExtIp:ExtPort  |
| <a id="opt-LblPort-3" href="#opt-LblPort-3" title="#opt-LblPort-3">LblPort</a> |
ExtIp:ExtPort:OtherPort                             | IntIp:LblPort  |
| <a id="opt-LblPort-4" href="#opt-LblPort-4" title="#opt-LblPort-4">LblPort</a> |
ExtIp1:ExtPort1:IntPort1 & ExtIp2:LblPort:IntPort2 | ExtIp2:LblPort |

!!! info ""
    In the above table:

    - `ExtIp` stands for "external IP found in the binding"
    - `IntIp` stands for "internal network container's IP",
    - `ExtPort` stands for "external Port found in the binding"
    - `IntPort` stands for "internal network container's port."
```

```
providers:
  docker:
    useBindPortIP: true
    # ...
```

```
[providers.docker]
  useBindPortIP = true
  # ...
```

```
--providers.docker.useBindPortIP=true
# ...
```

# `defaultRule`

The `defaultRule` option defines what routing rule to apply to a container if no rule is defined by a label.

It must be a valid Go template, and can use sprig template functions. The container name can be accessed with the `ContainerName` identifier. The service name can be accessed with the `Name`

identifier. The template has access to all the labels defined on this container with the `Labels` identifier.

```
providers:
  docker:
    defaultRule: "Host(`{{ .Name }}.{{ index .Labels \"customLabel\"}}`)"
    # ...
```

```
[providers.docker]
  defaultRule = "Host(`{{ .Name }}.{{ index .Labels \"customLabel\"}}`)"
  # ...
```

```
--providers.docker.defaultRule='Host(`{{ .Name }}.{{ index .Labels "customLabel"}}`)'
# ...
```

??? info "Default rule and Traefik service"

```
The exposure of the Traefik container, combined with the default rule mechanism,
can lead to create a router targeting itself in a loop.
In this case, to prevent an infinite loop,
Traefik adds an internal middleware to refuse the request if it comes from the same router.
```

## `constraints`

The `constraints` option can be set to an expression that Traefik matches against the container labels to determine whether to create any route for that container. If none of the container labels match the expression, no route for that container is created. If the expression is empty, all detected containers are included.

The expression syntax is based on the `Label("key", "value")`, and `LabelRegex("key", "value")` functions, as well as the usual boolean logic, as shown in examples below.

!!! tip "Constraints key limitations"

```
Note that `traefik.*` is a reserved label namespace for configuration and can not be used as a
key for custom constraints.
```

??? example "Constraints Expression Examples"

```
```toml
# Includes only containers having a label with key `a.label.name` and value `foo`
```
```

```
constraints = "Label(`a.label.name`, `foo`)"
```

```toml
# Excludes containers having any label with key `a.label.name` and value `foo`
constraints = "!Label(`a.label.name`, `value`)"
```

```toml
# With logical AND.
constraints = "Label(`a.label.name`, `valueA`) && Label(`another.label.name`, `valueB`)"
```

```toml
# With logical OR.
constraints = "Label(`a.label.name`, `valueA`) || Label(`another.label.name`, `valueB`)"
```

```toml
# With logical AND and OR, with precedence set by parentheses.
constraints = "Label(`a.label.name`, `valueA`) && (Label(`another.label.name`, `valueB`) ||
Label(`yet.another.label.name`, `valueC`))"
```

```toml
# Includes only containers having a label with key `a.label.name` and a value matching the
`a.+` regular expression.
constraints = "LabelRegex(`a.label.name`, `a.+`)"
```

For additional information, refer to Restrict the Scope of Service Discovery.

```
providers:
  docker:
    constraints: "Label(`a.label.name`,`foo`)"
    # ...
```

```
[providers.docker]
  constraints = "Label(`a.label.name`,`foo`)"
  # ...
```

```
--providers.docker.constraints=Label(`a.label.name`,`foo`)
# ...
```

# Routing Configuration

When using Docker as a provider, Traefik uses container labels to retrieve its routing configuration.

See the list of labels in the dedicated routing section.

# Routing Configuration with Labels

By default, Traefik watches for container level labels on a standalone Docker Engine.

When using Docker Compose, labels are specified by the directive `labels` from the "services" objects.

!!! tip "Not Only Docker"

```
Please note that any tool like Nomad, Terraform, Ansible, etc.
that is able to define a Docker container with labels can work
with Traefik and the Docker provider.
```

# Port Detection

Traefik retrieves the private IP and port of containers from the Docker API.

Port detection for private communication works as follows:

- If a container exposes a single port, then Traefik uses this port.
- If a container exposes multiple ports, then Traefik uses the lowest port. E.g. if `80` and `8080` are exposed, Traefik will use `80`.
- If a container does not expose any port, or the selection from multiple ports does not fit, then you must manually specify which port Traefik should use for communication by using the label `traefik.http.services.<service_name>.loadbalancer.server.port` (Read more on this label in the dedicated section in routing).

# Host networking

When exposing containers that are configured with host networking, the IP address of the host is resolved as follows:

- try a lookup of `host.docker.internal`
- if the lookup was unsuccessful, try a lookup of `host.containers.internal`, (Podman equivalent of `host.docker.internal`)
- if that lookup was also unsuccessful, fall back to `127.0.0.1`

On Linux, for versions of Docker older than 20.10.0, for `host.docker.internal` to be defined, it should be provided as an `extra_host` to the Traefik container, using the `--add-host` flag. For example, to set it to the IP address of the bridge interface (`docker0` by default): `--add-host=host.docker.internal:172.17.0.1`.

# IPv4 & IPv6

When using a docker stack that uses IPv6, Traefik will use the IPv4 container IP before its IPv6 counterpart. Therefore, on an IPv6 Docker stack, Traefik will use the IPv6 container IP.

# Docker API Access

Traefik requires access to the docker socket to get its dynamic configuration.

You can specify which Docker API Endpoint to use with the directive `endpoint`.

!!! warning "Security Note"

```
Accessing the Docker API without any restriction is a security concern:
If Traefik is attacked, then the attacker might get access to the underlying host.
{: #security-note }

As explained in the [Docker Daemon Attack Surface
documentation](https://docs.docker.com/engine/security/#docker-daemon-attack-surface):

!!! quote

    [...] only **trusted** users should be allowed to control your Docker daemon [...]

??? success "Solutions"
```

Expose the Docker socket over TCP or SSH, instead of the default Unix socket file.

It allows different implementation levels of the [AAA (Authentication, Authorization, Accounting) concepts](https://en.wikipedia.org/wiki/AAA_(computer_security)), depending on your security assessment:

- Authentication with Client Certificates as described in ["Protect the Docker daemon socket."](https://docs.docker.com/engine/security/protect-access/)
- Authorize and filter requests to restrict possible actions with [the TecnativaDocker Socket Proxy](https://github.com/Tecnativa/docker-socket-proxy).
- Authorization with the [Docker Authorization Plugin Mechanism](https://web.archive.org/web/20190920092526/https://docs.docker.com/engine/extend/plugins_authorization/)
- Accounting at networking level, by exposing the socket only inside a Docker private network, only available for Traefik.
- Accounting at container level, by exposing the socket on a another container than Traefik's.
- Accounting at kernel level, by enforcing kernel calls with mechanisms like [SELinux](https://en.wikipedia.org/wiki/Security-Enhanced_Linux), to only allows an identified set of actions for Traefik's process (or the "socket exposer" process).
- SSH public key authentication (SSH is supported with Docker > 18.09)
- Authentication using HTTP Basic authentication through an HTTP proxy that exposes the Docker daemon socket.

??? info "More Resources and Examples"

- ["Paranoid about mounting /var/run/docker.sock?"](https://medium.com/@containeroo/traefik-2-0-paranoid-about-mounting-var-run-docker-sock-22da9cb3e78c)
- [Traefik and Docker: A Discussion with Docker Captain, Bret Fisher](https://blog.traefik.io/traefik-and-docker-a-discussion-with-docker-captain-bret-fisher-7f0b9a54ff88)
- [KubeCon EU 2018 Keynote, Running with Scissors, from Liz Rice](https://www.youtube.com/watch?v=ltrV-Qmh3oY)
- [Don't expose the Docker socket (not even to a container)](https://www.lvh.io/posts/dont-expose-the-docker-socket-not-even-to-a-container/)
- [A thread on Hacker News about sharing the `/var/run/docker.sock` file](https://news.ycombinator.com/item?id=17983623)
- [To DinD or not to DinD](https://blog.loof.fr/2018/01/to-dind-or-not-do-dind.html)
- [Traefik issue GH-4174 about security with Docker

```
socket](https://github.com/traefik/traefik/issues/4174)
    - [Inspecting Docker Activity with
Socat](https://developers.redhat.com/blog/2015/02/25/inspecting-docker-activity-with-socat/)
    - [Letting Traefik run on Worker Nodes](https://blog.mikesir87.io/2018/07/letting-traefik-
run-on-worker-nodes/)
    - [Docker Socket Proxy from Tecnativa](https://github.com/Tecnativa/docker-socket-proxy)
```

{% include-markdown "includes/traefik-for-business-applications.md" %}

---

□□□ #2
□ □□□ □□□ 2026-10-31 14:00:38 UTC
□ □□□ □□□ 2026-10-31 14:00:39 UTC